Grundlagen zur Programmierung einer Zeichenfläche für ein CAD-System - Koordinatensysteme, Canvas, Zoom, Rollbalken, Maßstab

Mario Blunk

IMPRESSUM

Mario Blunk
Buchfinkenweg 3
99097 Erfurt
Deutschland

Grundlagen zur Programmierung einer Zeichenfläche für ein CAD-System - Koordinatensysteme, Canvas, Zoom, Rollbalken, Maßstab

Internet
http://www.blunk-electronic.de

Email
info@blunk-electronic.de

Für meine Frau, Geliebte und bester Freund.

In Ihm sind verborgen alle Schätze der Erkenntnis und Weisheit. Die Bibel, Kolosser 2.3

Vorwort

Der Anlaß zum Verfassen dieses Buches war der Wunsch, vollständiges Verständnis über die Mechanismen zu erlagen, die für die Programmierung einer Zeichenfläche nötig sind. Das große Ziel ist ein CAD-System.

Bis dato ist mir keinerlei Literatur zur Programmierung einer Zeichenfläche mit Rollbalken, Möglichkeiten zum Vergrößern, Verkleinern, Maßstab und der zugehörigen Mathematik bekannt. Obwohl hunderte Anwendungen diese Mechanismen beinhalten, ist dazu nichts öffentlich dokumentiert. Im englischen Sprachraum wird in diesem Zusammenhang der Begriff "canvas", zu deutsch "Leinwand", verwendet. Obwohl das deutsche Wort "Leinwand" etwas antiquarisch wirkt, werde ich bei diesem Begriff bleiben.

Die Programmierung einer Leinwand mit all den oben genannten Funktionen erwies sich als äußerst kompliziert und brachte mich gelegentlich an den Rand der Verzweiflung. Das komplexe Uhrwerk aus Vergrößerung, Rollbalken, Verschiebung und Fenstergrößen erschien mir zeitweilig wie ein Zauberwürfel. Ich möchte an dieser Stelle betonen, daß es keine einfache Lösung für diese Aufgabe gibt. Die Hersteller proprietärer CAD-Systeme haben guten Grund, dieses Wissen für sich zu behalten. Das mühevolle Ausarbeiten der nötigen Mechanismen und das Schreiben des Quellcodes für ein kleines Demoprogramm dauerte etwa ein Jahr. Die nötige Mathematik kommt mit Grundrechenarten aus.

Für die Umsetzung verwendete ich die Programmiersprache *Ada*, die zugehörige Anbindung an die *GTK3*-Bibliothek und die darin enthaltenen primitiven *CAIRO*-Routinen. Die in diesem Buch beschriebenen Abläufe und Mechanismen können aber auch mit jeder anderen Programmiersprache, wie C oder C++, reproduziert werden. Auch sollte die Realisierung statt mit *GTK3* auch mit *GTK4* oder *QT* möglich sein. Ich habe versucht, möglichst wenig Funktionen aus *GTK3* zu benutzen, um so wenig wie möglich Abhängigkeit zu erreichen.

Die Werkzeugsammlung in *GTK3* (Datentypen, Widgets, Funktionen und Prozeduren) ist zwar hervorragend dokumentiert, wie man daraus aber schöne graphische Benutzeroberflächen (engl. Graphical User Interfaces - GUI) oder gar ein CAD-System bauen kann, ist kaum allgemeinverständlich erklärt. Im Internet geistern lediglich einige Codefragmente herum, die selten verständlich kommentiert sind und die Kenntniss einer bestimmten Programmiersprache (meist C++) und Erfahrung mit *GTK* oder *QT* voraussetzen. Der gesamte Kontext oder ein allgemeiner Algorithmus fehlt leider immer. Dieses Buch soll dafür eine Hilfe sein.

Was die Umsetzung weiter erschwerte, war die Forderung nach einem System, bei dem die Werte der Y-Achse, dem kartesischen Koordinatensystem entsprechend, von unten nach oben zunehmen. In Bildbearbeitungsprogrammen wie *GIMP* oder dem CAE-Werkzeug *KiCad* steigen die Y-Werte jedoch von oben nach unten. Das ist historisch bedingt und - was die Programmierung angeht - bequem. Y-Werte müssen aber von unten nach oben zunehmen. Für technische Zeichnungen ist das weltweit eine Selbstverständlichkeit.

Das Ziel dieser Abhandlung ist es also, den Einstieg in die Problematik zu erleichtern, dem Leser einen Bauplan in die Hand zu geben, mit dem er dann in der Lage ist, eigene Vorstellungen zu realisieren.

Der Quellcode ist in diesem Buch mit abgedruckt. Das hat den Vorteil, daß man sich damit in den Garten setzen kann und keinerlei elektronische Geräte zum Studium nötig sind. Sie brauchen nur einen Bleistift und Kaffee.

Erfurt, 24. Februar 2025

Inhaltsverzeichnis

Kapitel 1

Aufbau der graphischen Oberfläche (GUI)

Weil es in diesem Buch nur um die grundlegenden Mechanismen einer Zeichenfläche geht, ist das Demoprogramm auf ein Minimum reduziert. Abbildung 1.1 zeigt das Demoprogramm, wie es sich dem Anwender unmittelbar nach dem Start zeigt. Es geht darum, die Grundlagen für die Oberfläche eines zukünftiges CAD-Systemes zu legen. Wir sehen links die Anzeige für Koordinaten, Abstände, Rasterweite, Vergrößerung und Maßstab. Rechts befindet sich die Leinwand auf der wir einen stark vereinfachten Zeichnungsrahmen, Rasterpunkte, einen Cursor (Fadenkreuz) und drei Objekte sehen. Die Objekte sind das sogenannte Modell, also das was wir zeichnen und darstellen wollen. Dieses Beispiel zeigt drei Objekte: Ein Dreieck, ein Rechteck und einen Kreis.

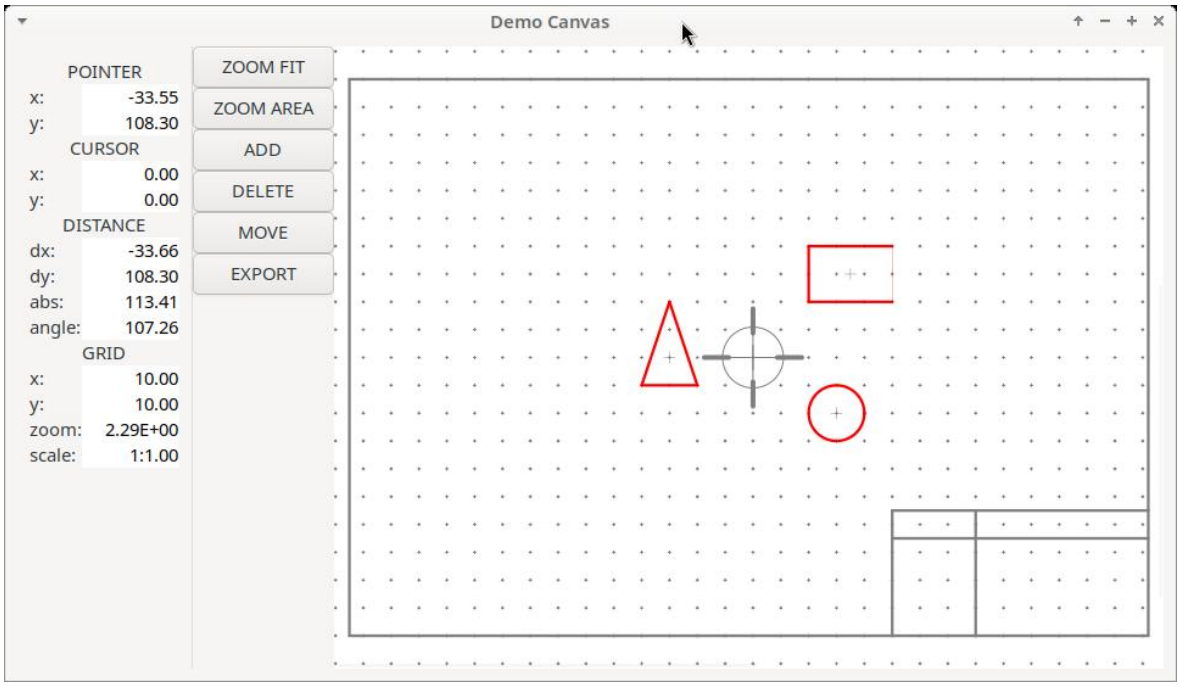

Abbildung 1.1: Demo-GUI

Die in diesem Buch wichtigsten zu behandelnden Elemente der Oberfläche sind in Abbildung 1.2 mittels Pfeilen markiert. Alle anderen Elemente wie die Anzeige der Position des Zeigers (Maus), der Position eines Cursors und Knöpfe[1] sind zweitranging und hier nur zur Inspiration gedacht. Dazu sei auf den Quellcode in den Abschnitten 9.11 und 9.14 verwiesen. In diesem Buch geht es im Wesentlichen um das Hauptfenster[2], Rollfenster[3], die Leinwand[4] und die Rollbalken[5].

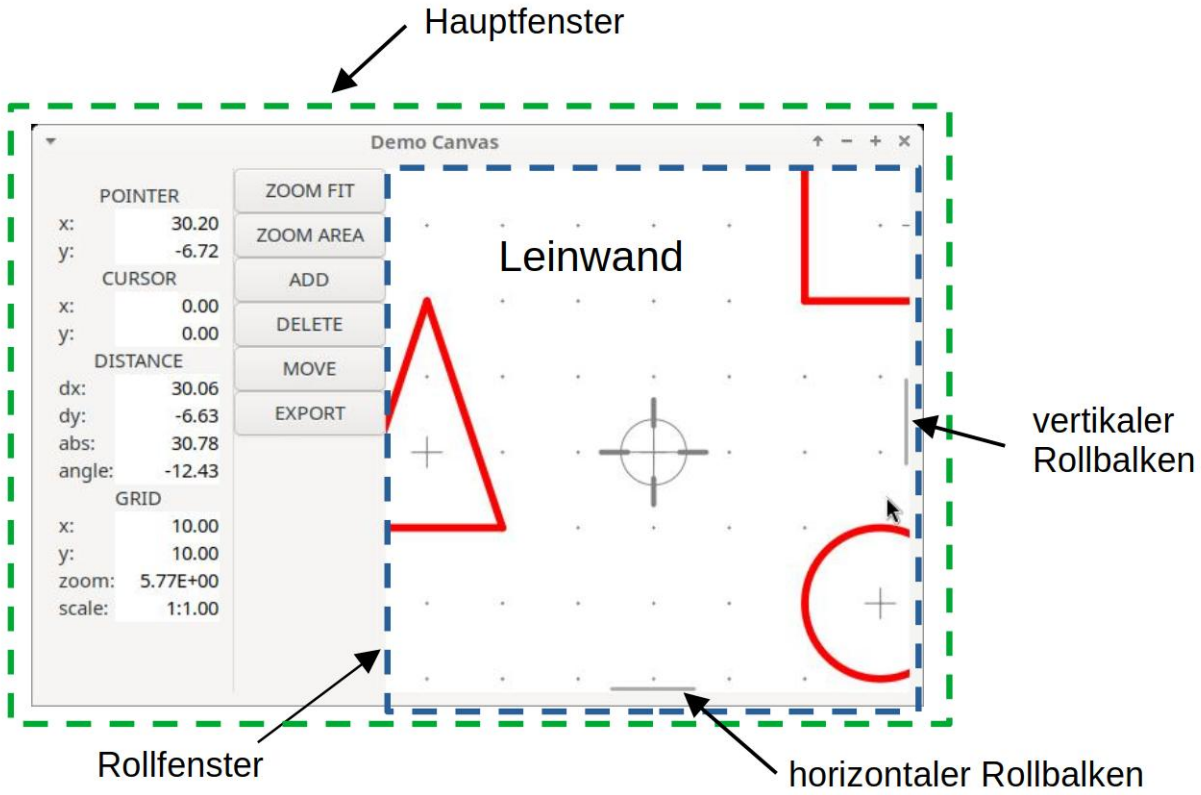

Abbildung 1.2: Demo-GUI beschriftet

Für die Leser, die es nicht abwarten können, das Demoprogramm zu testen, sei auf Abschitt 10 verwiesen.

Eine kurze Bedienungsanleitung des Demoprogrammes ist in Abschnitt 8 zu finden.

[1] engl. buttons
[2] engl. main window
[3] engl. scrolled window
[4] engl. canvas
[5] engl. scrollbars

Durch das Rollfenster sieht der Bediener die Leinwand (graue Fläche) auf welche die Objekte gezeichnet werden. In Abbildung 1.3 ist erkennbar, daß durch das Rollfenster nur ein Teil der Leinwand zu sehen ist. Die Leinwand ist meistens deutlich größer als das Rollfenster. Ohne das Rollfenster müßte man die gesamte Leinwand in der Oberfläche abbilden. Die Abmessungen der Leinwand werden in Pixeln (oder Bildpunkten) ausgedrückt. Im Demoprogramm können die Abmessungen beispielsweise etwa 400.000 mal 200.000 Bildpunkte sein. Diese riesige Fläche kann nur mittels eines Rollfensters sinnvoll gehandhabt werden. Das Rollfenster hat deutlich kleinere Abmessungen, wie z.B. 800 mal 400 Bildpunkte, kann also bequem in das Hauptfenster eingefügt werden. Das Rollfenster wird mittels der Rollbalken vor der Leinwand (hoch, runter, rechts oder links) bewegt sodaß jeder Bereich der Leinwand betrachtet werden kann.

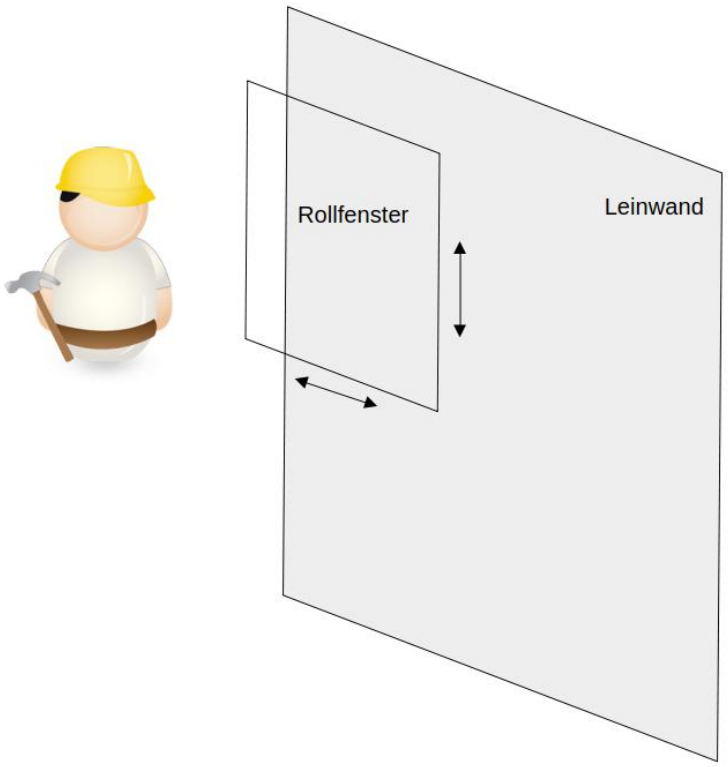

Abbildung 1.3: Rollfenster und Leinwand

Abschließend soll auf die Unterteilung des Hauptfensters eingegangen werden. Dazu soll Abbildung 1.4 helfen. In *GTK* wird das Element Kiste[6] verwendet, um die Oberfläche in kleinere Sektionen zu unterteilen und darin weitere Elemente oder weitere Unterteilungen vorzunehmen. Es gibt vertikale und horizontale Kisten welche unter *GtkAda* vom Typ `gtk_vbox` beziehungsweise `gtk_hbox` sind. Innerhalb einer horizontalen Kiste werden die Elemente nebeneinander platziert wohingegen in einer vertikalen Kiste übereinander platziert wird. Der boolsche Parameter `expand` bestimmt, ob das Element bei Größenveränderung des übergeordneten Elementes sein Größe automatisch anpaßt oder nicht.

Für unser Demoprogramm ist es wichtig, daß die Elemente `box_v1`, `separator` und `box_v0` ihre Größe nicht verändern. Darum ist der Parameter `expand` bei diesen Elementen auf `false` gesetzt. Dem Rollfenster `swin` soll die gesamte verbleibende Fläche des Hauptfensters zuteil werden. Der Parameter `expand` ist demzufolge auf `true` gesetzt (Standardeinstellung). Wenn also das Hauptfenster vom Bediener maximiert wird, paßt sich `box_h` diesem automatisch an. Daraufhin ändert nur das Rollfenster seine Größe. Alle anderen Kisten innerhalb `box_h` bleiben

[6] engl. box

unverändert.

In Kiste box_v1 befindet sich die Koordinatenanzeige. Der Separator trennt box_v1 mit einem vertikalen Balken von box_v0. Letztere beinhaltet die Knöpfe ZOOM FIT, ZOOM AREA, ADD u.s.w. Der gesamte verbleibende Platz auf der rechten Seite wird vom Rollfenster swin eingenommen.

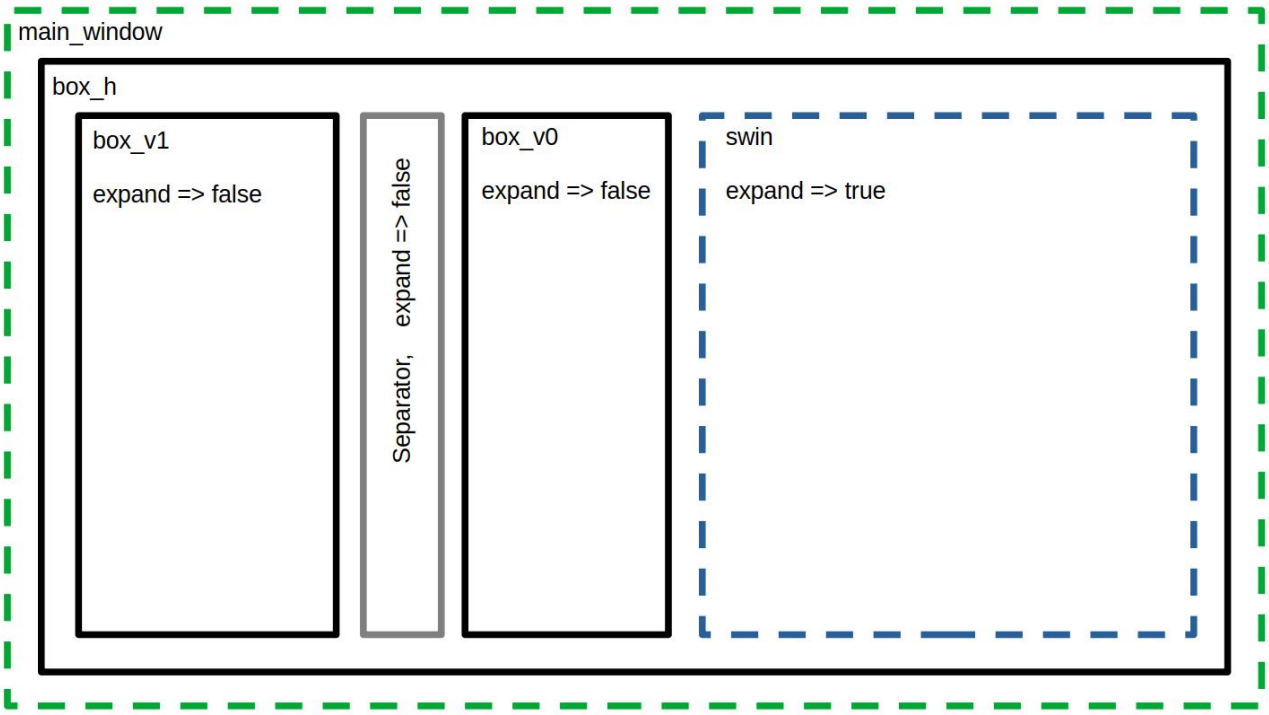

Abbildung 1.4: Gesamter Aufbau der graphischen Oberfläche

Der zugehörige Quellcode ist an folgenden Stellen zu finden:

- Prozedur create_window in Abschnitt 9.9: Dort wird das Hauptfenster main_window erzeugt, die Kiste box_h in das Hauptfenster eingefügt, box_v1 sowie der Separator erzeugt und in box_h eingefügt.
- Prozedur create_buttons in Abschnitt 9.14: In dieser Prozedur wird box_v0 erzeugt und in box_h eingefügt.
- Prozedur create_canvas in Abschitt 9.5: An dieser Stelle wird das Rollfenster swin in box_h eingesetzt.

Kapitel 2

Koordinatensysteme

Es geht in diesem Buch nicht um die Entwicklung eines Malprogrammes, sondern um die Grundlagen für ein echtes CAD-System. Darum müssen wir mit **drei** verschiedenen Koordinatensystemen umgehen können. Das impliziert permanente Konvertierungen zwischen diesen Systemen. Positionen von Objekten, Längen und Breiten müssen von einem Koordinatensystem in ein anderes umgerechnet werden. Jedes Koordinatensystem verwendet eigene Maßeinheiten. Je nachdem welche Programmiersprache zur Anwendung kommt, muß mehr oder weniger viel Sorgfalt der Konvertierung von einem Zahlentyp in einen anderen gewidmet werden (Siehe Abschnitt 11.1). Die Koordinatensysteme sind:

1. Modellkoordinaten (Kurzbezeichnung CS1)
2. Leinwandkoordinaten (Kurzbezeichnung CS2)
3. Koordinaten der graphischen Oberfläche (Kurzbezeichnung CS3)

2.1 Koordinaten des Modells (CS1)

Das Modell ist die Abstraktion der Objekte aus der realen Welt die wir darstellen wollen. Wenn zum Beispiel Gebäude konstruiert werden sollen, besteht das Modell aus Wänden, Treppen, Fenstern und Türen. In der Elektronikkonstruktion geht es um Transistoren und Leiterplatten.

Wenn darzustellenden Objekte sehr groß oder sehr klein sind, kommt noch der Maßstab ins Spiel. Wir kümmern uns für den Moment nicht um den Maßstab. Mehr dazu ist in Abschnitt 3.1 zu finden.

Wichtige Einschränkung: Wir verwenden nur 2D-Koordinaten. Das heißt, es gibt nur Breite und Höhe oder - mit anderen Worten - nur die X- und Y-Achse.

Besonderheit: Das Modell verwendet ein Koordinatensystem in welchem die Werte auf der X-Achse von links nach rechts zunehmen. Die Y-Werte nehmen von **unten nach oben** zu. Das entspricht dem kartesischen Koordinatensystem laut Abbildung 2.1 in welchem fast alle technischen Zeichnungen in der Industrie angefertigt werden. Verwendete Einheiten sind Millimeter, Meter, Zoll, Meilen, ... Es können positive und negative Werte verwendet werden. So befindet sich z.B. die Bodenplatte eines Gebäudes auf einer Höhe von -5 Metern.

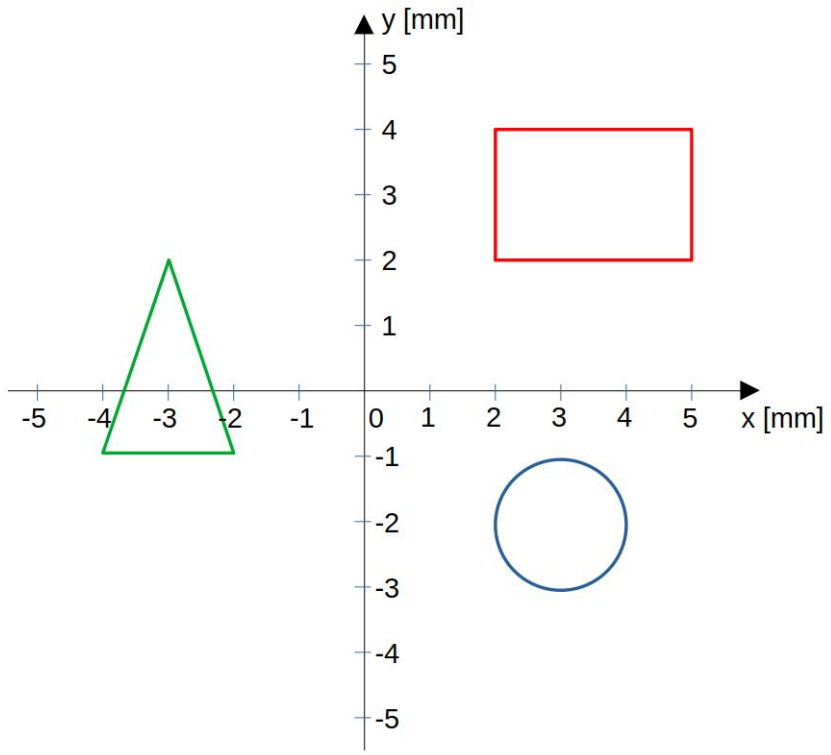

Abbildung 2.1: Kartesisches Koordinatensystem

Ein Punkt P im Modell wird mittels eines sogenannten Vektors ausgedrückt. Das Demoprogramm verwendet den Datentyp `type_vector_model` um Modellkoordinaten zu handhaben (Siehe Quellcode Abschnitt 9.18). So befindet sich im Bild Abbildung 2.1 die Spitze des Dreieckes bei (-3,0/2,0)mm:

$$\vec{P} = \begin{pmatrix} -3,0 \\ 2,0 \end{pmatrix} mm$$

Wir verwenden in diesem Buch für Modellkoordinaten grundsätzlich die Einheit Millimeter (mm).

2.1.1 Weitere Bestandteile des Modells

Neben den eigentlichen darzustellenen Objekten des Modells müssen noch weitere Dinge auf der Leinwand abgebildet werden. In CAD-Systemen ist von großer Bedeutung der Zeichnungsrahmen und das Dokumentationsfeld. Der Rahmen hat Abmessungen, die meistens in Millimetern angegeben werden. Details zum Zeichnungsrahmen folgen in Abschnitt 3.4.

Sofern ein Cursor benötigt wird, ist dieser ebenfalls auf der Leinwand zu sehen. Seine Position wird mit Modellkoordinaten angegeben. Der Cursor wird mittels eines Fadenkreuzes dargestellt und kann mit den Cursortasten bewegt werden. Mehr dazu ist in den Abschnitten 3.12 und 6.6.5 zu finden.

2.1.2 Festkommazahlen oder Fließkommazahlen ?

Die Antwort auf diese Frage kann nicht pauschal gegeben werden. Dieses Thema wird jeden Programmierer früher oder später betreffen.

Wer meint, alle geometrischen und mathematischen Probleme mit Fließkommazahlen[1] von größtmöglicher Genauigket lösen zu können, wird in zahlreiche Fallen treten und viel Zeit mit der Fehlersuche vergeuden. Die katastrophalen Auswirkungen von falsch angewendeten Fließkommazahlen sind in [2] anschaulich beschrieben.

Ebenso ist auch nicht jedes Problem mit Festkommazahlen[2] lösbar. Festkommazahlen werden zudem noch unterschieden in gewöhnliche binäre[3] und dezimale Festkommazahlen[4].

Grundlagen zu den Zahlentypen `float`, `binary fixed` und `decimal fixed` sind in [1] und [3] zu finden.

Wir werden es mit drei Kategorien von Zahlentypen in CS1 zu tun haben:

1. Eingangswerte: Also Längen, Positionen oder Winkel, die vom Menschen vorgegeben werden. Das heißt, der Bediener bestimmt, daß sich ein Objekt auf Position (120,44mm; -23,89mm) bei einem Drehwinkel von 145,11 Grad befindet. Für Distanzen und Positionen in CS1 **müssen** binäre Festkommazahlen verwendet werden, um den relativen Fehler zu begrenzen. Fließkommazahlen sind nicht geeignet, weil der relative Fehler von der Größe der Zahl abhängt. Außerdem addieren sich Rundungsfehler bei mehrfachen, aufeinanderfolgenden Additionsoperationen. So ist es z.B. nicht hinnehmbar, daß ein Objekt nach 1000 Verschiebeoperationen um jeweils 1mm, am Ende mit einer Toleranz von 0,5mm seine endgültige Position findet. Ebenso müssen für Winkel binäre Festkommazahlen verwendet werden. In CAD-Systemen werden Winkel in Grad, also durch Werte von -360,0 bis +360,0 angegeben.

2. Zwischenwerte: Das sind systeminterne Zahlen, die z.B. im Zusammenhang mit Winkelfunktionen verwendet werden. Der hierfür nötige Datentyp ist die Fließkommazahl. Der Wertebereich und die Anzahl der Stellen[5] ist je nach Modell angemessen festzulegen. Wenn z.B. Objekte gedreht, Schnittpunkte von Geraden berechnet oder Polygone verarbeitet werden sind Fließkommazahlen genau die richtige Wahl. Als Beispiel sei die Funktion `get_distance` im Quellcode in Abschnitt 9.18 genannt.

3. Ausgangswerte: Wir befinden uns wieder in CS1. Hier gilt das zu den Eingangswerten Gesagte (siehe oben). Es **müssen** ebenso binäre Festkommazahlen verwendet werden, weil Werkzeugmaschinen wie Bestückautomaten, Plotter oder Bohrmaschinen mit Festkommazahlen arbeiten. Der maximale relative Fehler ist dabei immer begrenzt.

Die Typdeklarationen für diese Zahlentypen sind im Quellcode in Abschnitt 9.18 zu finden.

[1] engl. floating point numbers
[2] engl. fixed point numbers
[3] engl. ordinary binary fixed point numbers
[4] engl. decimal fixed point numbers
[5] engl. digits

2.1.2.1 Vergleichsoperationen mit Fließkommazahlen

Ergänzend sei noch auf ein anderes Problem im Zusammenhang mit Fließkommazahlen hingewiesen, die für Zwischenwerte verwendet werden. Das Problem wird uns in diesem Buch zum Glück nicht beschäftigen. Wenn der Leser aus dem Demoprogramm aber ein echtes CAD-System machen möchte, wird sich folgende Problematik offenbaren: Fließkommazahlen können eine Vielzahl von Nachkommastellen aufweisen. Das bringt eine oft viel zu große Genauigkeit mit sich, die bei einer Vergleichsoperationen wie der folgenden hinderlich ist:

A und B seien zwei Fließkommazahlen, die eine Entfernung in Metern ausdrücken. Es soll mit hinreichender Genauigkeit festgestellt werden, ob beide gleich groß sind:

```
if A = B then
  put_line ("gleich");
else
  put_line ("ungleich");
end if;
```

Die Verwendung des Standard-Vergleiches mit "=" bedeutet, daß A und B exakt bis auf die letzte Nachkommastelle miteinander verglichen werden. Wenn die beiden Operanden A und B sich auch nur in der dreiunzwanzigsten Nachkommastelle unterscheiden, fällt der Test negativ aus (A ist ungleich B). Es muß also die Standard-Vergleichsfunktion "=" neu definiert werden[6]. In der vernünftigen Neudefinition in unserem Beispiel wird dann festgelegt, daß z.B. nur bis zur zweiten Nachkommastelle verglichen wird. Alle nachfolgenden Nachkommastellen werden ignoriert. Jede Programmiersprache hat ihre eigene Art, mit diesem Problem umzugehen.

Die jeweilige Anwendung oder Situation bestimmt, mit welcher Genauigkeit der Vergleich stattfinden muß.

Folgende geometrische Operationen sind von dieser Problematik betroffen:

- Ermittlung des Schnittpunktes zweier Geraden
- Ermittlung der Schnittpunkte von Geraden und Kreisen
- Operationen mit Polygonen (Expansion, Zuschneiden, Überlappung, ...)

[6] engl. redefining

2.2 Koordinaten der Leinwand (CS2)

Die Leinwand ist etwas virtuelles und wird vom Graphiksystem (*GTK* oder *QT*) erzeugt. Es ist etwas, auf dem der Anwender zeichnen kann. *GTK3* verwendet für die Koordinaten Fließkommazahlen des Types `gdouble`[7]. In der *GTK3*-Terminologie wird hier von sogenannten `logischen Pixeln` gesprochen. Wir verwenden für Maße in Zusammenhang mit der Leinwand von nun an die Einheit `lp`.

Um eine saubere Trennung von der *GTK3*-Bibliothek `glib` zu erhalten, wurde im Demoprogramm vom Typ `gdouble` der neue Datentyp `type_logical_pixels` abgeleitet. Auf diesem basiert wiederum der Datentyp `type_logical_pixels_vector` um mittels der Vektorrechnung Leinwandkoordinaten auszudrücken und zu handhaben.

Der Typ `gdouble` wie auch der davon abgeleitete Typ `type_logical_pixels` schließt negative Zahlen ein, was aber im Widerspruch zu Leinwandkoordinaten steht, denn **die Leinwand kennt nur positive Koordinaten** ! Solange Leinwandkoordinaten nur berechnet werden, ist das kein Problem. Oft entstehen innerhalb des Prozesses der Berechnung der Leinwandkoordinaten sogar negative Werte. Wenn jedoch am Ende aller Rechnerei auf die Leinwand gezeichnet wird, ist zu beachten, daß dann nur positive Koordinaten an die primitiven *CAIRO*-Routinen übergeben werden. Aus diesem Grund verwendet das Demoprogramm dafür den Subtyp `type_logical_pixels_positive`. Siehe Quellcode in Abschnitt 9.19.

Auf der Leinwand steigen die Werte der X-Achse von links nach rechts. Die Werte der Y-Achse **steigen von oben nach unten**, was gewöhnungsbedürftig aber leider zu akzeptieren ist. In Abbildung 2.2 ist dies erkennbar. Dieser unangenehme Umstand ist historisch bedingt, stammt aus den Anfängen der Fernsehtechnik[8], verursacht aber im Zusammenhang mit CAD mehr Probleme als Nutzen und wird uns in diesem Buch noch oft beschäftigen.

Es gibt auf der Leinwand, wie oben gesagt, keine negativen, sondern nur positive Koordinaten. So befindet sich der Ursprung der Leinwand, die Koordinate (0,0;0,0) in der oberen linken Ecke.

[7] *QT* verwendet dafür ganze Zahlen.

[8] Damals war die obere linke Ecke des Bildschirmes der Startpunkt beim Aufbau des Bildes.

Ein Punkt C auf der Leinwand wird mittels eines Vektors ausgedrückt. Die Spitze des Dreiecks im Bild unten befindet sich z.B. auf der X-Achse bei 2,0lp und auf der Y-Achse bei 4,0lp:

$$\vec{C} = \begin{pmatrix} 2,0 \\ 4,0 \end{pmatrix} lp$$

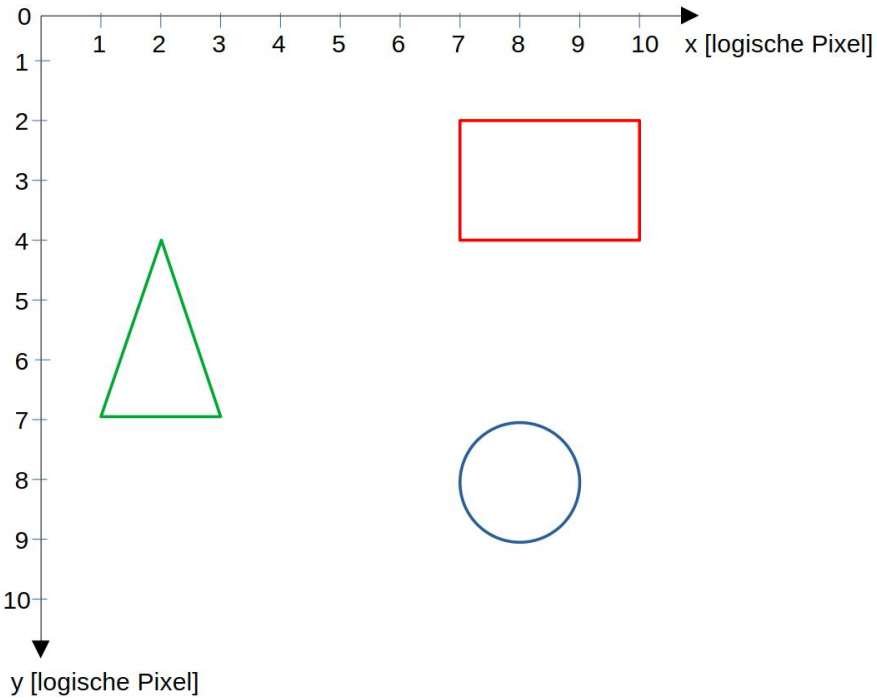

Abbildung 2.2: Koordinatensystem der Leinwand

Die Werte in diesem Beispiel sind sehr klein. Typische Leinwandkoordinaten bewegen sich bis weit in die Hundertausende hinein.

2.3 Koordinaten der graphischen Oberfläche (CS3)

Abmessungen und Positionen von Fenstern[9], Knöpfen oder die der Leinwand werden mit ganzen Zahlen des Types `gint` angegeben.

Auch hier haben wir wieder einen Konflikt, denn die Positionen und Abmessungen sind immer positive ganze Zahlen. Der Typ `gint` schließt aber negative Zahlen mit ein !

Im Zusammenhang mit GTK3 spricht man hier von sogenannten `Gerät-Pixeln`[10]. Wir verwenden dafür in diesem Buch die Einheit `dp`.

Zu beachten ist auch hier wieder der leidige Umstand, daß die obere linke Ecke eines Fensters die Koordinate (0;0) ist und der Y-Wert nach unten steigt[11]. Die Leinwand hat eine Breite[12] und eine Höhe[13].

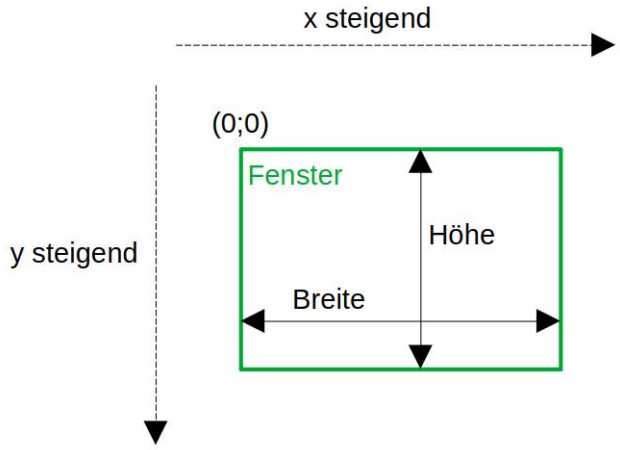

Abbildung 2.3: Abmessungen eines Fensters

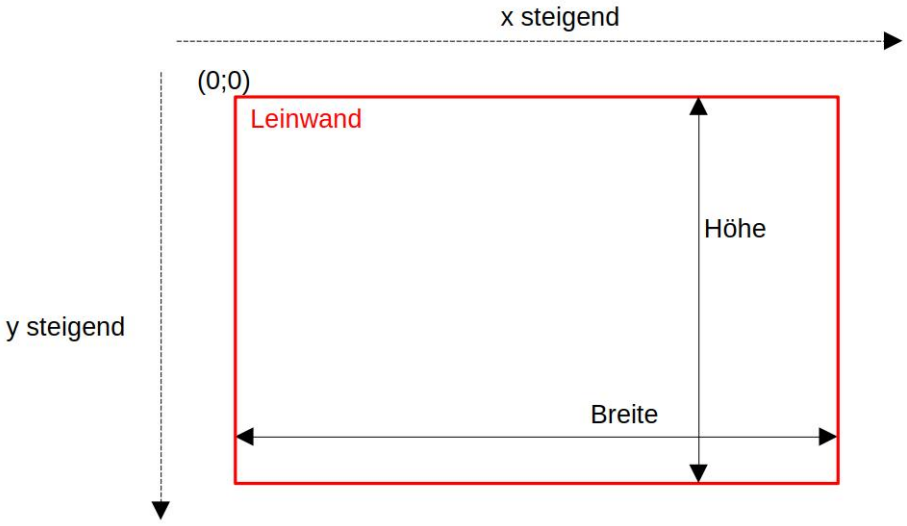

Abbildung 2.4: Abmessungen der Leinwand

[9] engl. windows
[10] engl. device pixels
[11] engl. increasing
[12] engl. width
[13] engl. height

Kapitel 3

Zentrale Größen und Begriffe

Es soll in diesem Kapitel zunächst nur um Größen und Begiffe gehen, ohne auf deren Zusammenhänge näher einzugehen. Als Vorschau seien diese zentralen Größen genannt:

- Maßstab[1] M und Papiergröße[2]
- Vergrößerungsfaktor[3] S
- Begrenzungsrechteck B
- Basis-Offset F
- Translation-Offset T

3.1 Maßstab und Papiergröße

Die darzustellenden Objekte sind oft viel größer als die verfügbare Zeichenfläche. Die Zeichenfläche wird meist durch die verwendete Papiergröße wie A4 oder A3 bestimmt. Ob tatsächlich auf Papier gedruckt wird, spielt keine Rolle. Die Objekte müssen also verkleinert dargestellt werden. Umgekehrt, wenn die Objekte sehr klein sind, müssen diese vergrößert werden, damit Details erkennbar sind. Das nennt man "Maßstäbliche Darstellung". Der Maßstab M wird dabei als das Verhältnis von dargestellter Größe zu wirklicher Größe in der Realität ausgedrückt.

So bedeutet z.B. ein Maßstab M von 1:100, daß 1mm in der Zeichnung 100mm in der Wirklichkeit entsprechen. Wir haben es also mit einer **Verkleinerung** um Faktor 100 zu tun.

Der Maßstab M von 10:1 bedeutet: 10mm in der Zeichnung entsprechen 1mm in der Wirklichkeit. Statt 10:1 kann man auch schreiben 1:0,1. Die Zeichnung stellt also die Objekte mit 10facher **Vergrößerung** dar.

Wenn keine maßstäbliche Darstellung gefordert ist, kann das Thema Maßstab vorerst ignoriert werden. **Wir nehmen also im Folgenden an, daß das Modell im Maßstab 1:1 auf der Leinwand dargestellt wird.**

Erst in dem späteren Kapitel 7 wird das Thema mit all seinen Herausforderungen und Details aufgegriffen. Es würde im Folgenden nur unnötig Verwirrung stiften.

[1] engl. scale
[2] engl. paper size
[3] engl. zoom factor

3.2 Vergrößerungsfaktor

Der Vergrößerungsfaktor S drückt aus, in welchem Maß die Zeichnung dem Bediener auf der Leinwand vergrößert oder verkleinert dargestellt wird. Der Vergrößerungsfaktor S ist **nicht** zu verwechseln mit dem Maßstab M (siehe vorherigen Abschnitt) ! Es handelt sich hier um zwei völlig verschiedene Größen.

Der für S verwendete Datentyp ist die Fließkommazahl welche größer als Null sein muß. S drückt aus, wieviele logische Bildpunkte pro Millimeter verwendet werden. Die Einheit ist also:

$$S = \frac{lp}{mm} \tag{3.1}$$

Für die Umrechnung einer Länge L im Modell zur Länge C auf der Leinwand gilt:

$$C = L \cdot S \tag{3.2}$$

Beispiel: 2mm im Modell entsprechen bei einer Vergrößerung von 10 auf der Leinwand einer Länge von 20,0lp:

$$20,0lp = 2,0mm \cdot 10,0\frac{lp}{mm} \tag{3.3}$$

Umgekehrt gilt für die Umrechnung einer Länge C auf der Leinwand zur Länge L im Modell:

$$L = \frac{C}{S} \tag{3.4}$$

S ist eine globale Variable. Siehe Quellcode in Abschnitt 9.12 für Details.

3.2.1 Minimaler und maximaler Vergrößerungsfaktor

Desweiteren gibt es einen maximalen Vergrößerungsfaktor S_{max} und einen minimalen Vergrößerungsfaktor S_{min}. Beide sind Systemgrenzen und werden durch die Natur des Modelles festgelegt. Die Anwendung entscheidet, in welchem Bereich sich der Vergrößerungsfaktor bewegen soll.

Beispiel:

Ein Rohr hat einen Durchmesser von von 100mm. Bei einem Vergrößerungsfaktor S von 1,0 wird dieses Rohr mit einer Breite von 100lp auf der Leinwand gezeichnet.

Wenn die Vorgabe ist, daß ein Rohr diesen Durchmessers bei maximaler Vergrößerung mit höchstens 10.000lp dargestellt werden soll, dann ist S_{max} mit 100 festzulegen:

$$S_{max} = \frac{C}{L} \tag{3.5}$$

$$S_{max} = \frac{10.000lp}{100mm} \tag{3.6}$$

$$S_{max} = 100 \tag{3.7}$$

Wenn bei kleinster Vergrößerung 5lp für das Rohr ausreichen, ist S_{min} mit 0,05 zu fixieren:

$$S_{min} = \frac{C}{L} \tag{3.8}$$

$$S_{min} = \frac{5lp}{100mm} \tag{3.9}$$

$$S_{min} = 0,05 \tag{3.10}$$

3.2.2 Multiplikator

Beim Vergrößern oder Verkleinern muß S stufenweise geändert werden (Siehe Abschnitt 8.1.2). Eine angemessene Schrittweite ist mit Hilfe eines Multiplikators S_m erreichbar. Im Demoprogramm ist S_m eine Konstante mit dem Wert 1,2.

Wenn vergrößert[4] wird, z.B. mittels des Mausrades, erfolgt eine Multiplikation des gegenwärtigen Vergrößerungsfaktors S_1 mit dem Multiplikator um den neuen Vergrößerungsfaktor S_2 zu erhalten:

$$S_2 = S_1 \cdot S_m \tag{3.11}$$

Umgekehrt, beim Verkleinern wird durch den Multiplikator geteilt:

$$S_2 = \frac{S_1}{S_m} \tag{3.12}$$

Diese Funktionen werden mittels der Prozeduren
`increase_zoom_factor` und `decrease_zoom_factor` realisert. Die Umsetzung ist im Quellcode in Abschnitt 9.12 zu finden.

[4] engl. zoom-in

3.3 Objekte

Bisher haben wir nur verallgemeinert von Objekten gesprochen. Es ist nun zu diskutieren, was wir im Zusammenhang mit CAD als Objekt bezeichnen.

Wir werden von nun an unterscheiden zwischen:

- primitiven Objekten und
- komplexen Objekten

Details zur Typdeklaration von Objekten und zugehörigen Unterprogrammen sind im Quellcode in Abschnitt 9.3 zu finden.

3.3.1 Primitive Objekte

Ein einfaches, sogenanntes primitives Objekt ist ein Kreis, ein Kreisbogen oder eine Linie. Aus diesen läßt sich jedes übergeordnete komplexe Objekt zusammenbauen.

Das Demoprogramm kennt nur diese Kategorien primitiver Objekte. Auf Kreisbögen wurde also verzichtet:

- Linien, modelliert durch Linienbreite, Start- und Endpunkt. Siehe Abbildung 3.1.
- Kreise, modelliert durch Linienbreite, Radius und Mittelpunkt. Siehe Abbildung 3.2.

Primitive Objekte werden in einem CAD-System nicht direkt vom Bediener ausgewählt und auf der Leinwand platziert[5]. Sie sind immer nur Teil eines komplexen Objektes, welches wir später definieren.

[5] Innerhalb von sogenannten Bauteilbibliotheken werden komplexe Objekte mittels primitiver Objete zusammengesetzt.

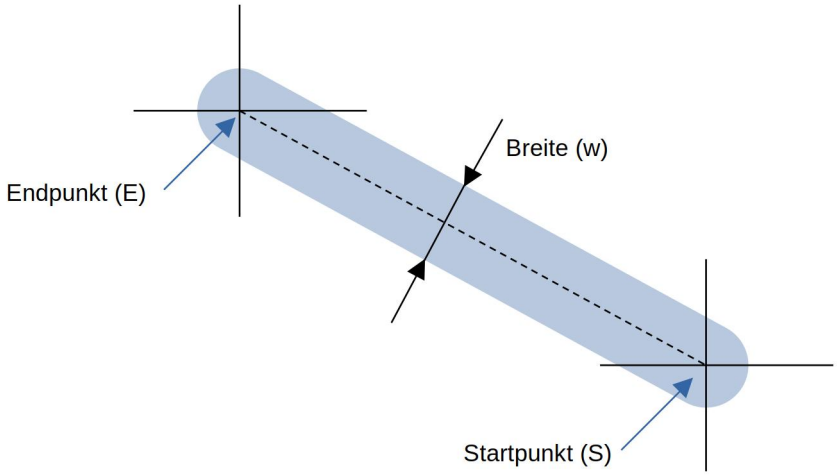

Abbildung 3.1: Linie

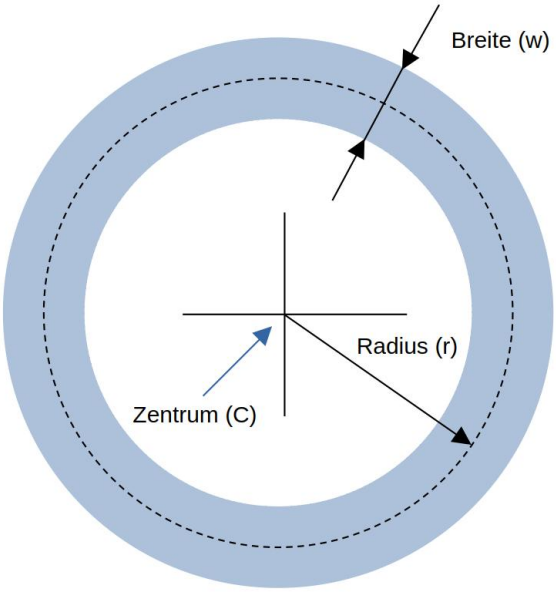

Abbildung 3.2: Kreis

3.3.1.1 Start- und Endpunkte

Die Start- und Endpunkte von Linien (Abbildung 3.1) und Kreisbögen müssen abgerundet sein. In GTK3 gibt es dafür den Befehl `set_line_cap`. Er ist in Prozedur `cb_draw` im Quellcode in Abschnitt 9.8 zu finden. Das hat den Vorteil, daß beim Aneinanderreihen von Linien und Kreisbögen sich immer kantenfreie, sauber Übergangsstellen bilden. Der Radius r am Ende oder Anfang einer Linie oder eine Kreisbogens entspricht immer der halbe Linienbreite w.

$$r = \frac{w}{2} \tag{3.13}$$

3.3.1.2 Besonderheiten zur Linienbreite

Zur Breite von Linien oder dem Umfang eines Kreises oder Kreisbogens (Abbildungen 3.1 und 3.2) sollen hier einige Anmerkungen festgehalten werden.

Die Breite w einer Linie spielt in der Realität oft eine Rolle. Zum Beispiel wird die Breite einer Leiterbahn auf einer Leiterplatte in Millimetern angegeben. Sie ist also fertigungstechnisch absolut relevant. In diesem Fall wird beim Vergrößern und Verkleinern (zoom) die Breite mit skaliert.

Es kann sich aber auch um eine abstrakte Linie handeln, die z.B. eine Fräskontur darstellt. Hier macht die Angabe einer Linienbreite keinen Sinn. In der Praxis wird dann einfach die Breite Null angegeben, was aber eigentlich nur ein Provisorium ist. Beim Darstellen einer solchen Linie oder eines solchen Kreisbogens auf der Leinwand muß aber dennoch eine bestimmte Breite in logischen Pixeln (lp) angegeben werden, weil ansonsten das Objekt nicht sichtbar wäre. Diese Breite ist konstant, also unabhägig vom Vergrößerungsfaktor, zu halten[6].

Die entscheidende Frage ist also, ob das primitive Objekt eine Kontur darstellt oder nicht.

Im Demoprogramm wurde diese Unterscheidung nicht vorgenommen, muß aber in einem echten CAD-System unbedingt gemacht werden.

Bei der Typedeklaration einer Line, bietet sich in der Programmiersprache *Ada* der sogenannte `parametrierte Typ`[7] an. Der steuernde Parameter ist z.B. dann der Parameter `contour`. Die unten stehende Typdeklaration bedeutet in Worten: Wenn der Parameter `contour` auf `true` gestellt ist, handelt es sich um eine Kontur bei der es keine Breite w gibt. Ist `contour` auf `false` gesetzt, kann eine Breite w angegeben werden:

```
type type_line_controlled (contour : boolean) is record
     s, e : type_vector_model;
     case contour is
          when TRUE =>
               null;
          when FALSE =>
               w : type_distance_model_positive := 0.1;
     end case;
end record;
```

Somit gibt es eine klare Trennung zwischen Konturen und Nicht-Konturen.

3.3.1.3 Besonderheiten zu gefüllten Kreisen

Der Gedanke, einen Kreis zu füllen kommt unweigerlich auf, ist aber an dieser Stelle verfrüht. Es gibt jedoch noch andere Konturen, die man mit einer Farbe oder einem Muster füllen kann. Das können Rechtecke und unregelmäßige geschlossene Formen (Polygone) sein. Die Modellierung von Konturen findet auf der Ebene der komplexen Objekte statt. Somit ist auch das Füllen von Kreisen auf dieser Ebene nicht angebracht.

[6] Beim Darstellen des Cursors haben wir es mit einer ähnlichen Problematik zu tun. Siehe Abschnitt 6.6.5

[7] engl. parameterized type

3.3.2 Komplexe Objekte

In CAD-Systemen haben wir es gewöhnlich mit weitaus komplexeren Objekten zu tun als Linien und Kreisen. Aus diesem Grund sprechen wir jetzt von komplexen Objekten. Diese wiederum bestehen aus primitiven Objekten. Der Bediener holt die komplexen Objekte aus Bauteilbibliotheken, platziert und verschiebt diese auf der Leinwand oder löscht sie von dort wieder. So setzt sich z.B. das Symbol für einen Transistor, eine Schraube oder ein Zahnrad aus vielen einzelnen Linien, Kreisen und Kreissegmenten zusammen. Dazu seien hier zwei Beispiele aufgeführt. Abbildung 3.4 zeigt das Schaltplansymbol eines Transistors. Es setzt sich zusammen aus acht Linien und zwei Textmarken für den Namen (T1) und seinen Wert (BC548C). In der Mitte ist der Ursprung[8] des Objektes durch ein Kreuz erkennbar. Der Ursprung kennzeichnet später in der Zeichnung die x/y-Position des komplexen Objektes. Diese Positionsmarke ist für das Auffinden von Objekten in der Zeichnung und für z.B. maschinelle Montageprozesse (CAM) äußerst wichtig. Details zum letztendlichen Darstellen des Ursprunges sind in Abschnitt 6.6.9.1 zu finden.

Die Position eines komplexen Objektes ist es eine Frage der Definition. Man kann die untere linke Ecke eines Objektes als Referenz für die Position benutzen oder seine Mitte (siehe Abbildung 3.3). Bei unregelmäßigen, nicht-symmetrischen Objekten ist die geometrische Mitte oder der Schwerpunkt eines Objektes besser geeignet.

Relativ zur dieser Position werden später die primitiven Objekte auf die Leinwand gezeichnet (siehe Abschitt 6.6.9.2).

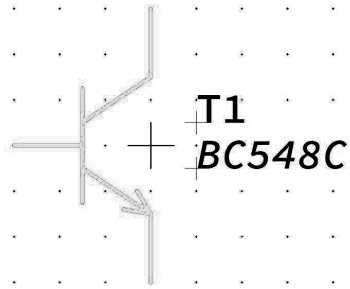

Abbildung 3.3: Schaltplansymbol eines Transistors

Bild 3.3 zeigt das Gehäuse eines Transistors. Es setzt sich zusammen aus 4 Linien und drei gefüllten Rechtecken. Dazu kommen - wie beim Schaltplansymbol - zwei Textmarken für den Namen (T1) und seinen Wert (BC548C). Hier ist der Ursprung in Form eines Kreuzes mittig im rechteckigen Gehäuse zu sehen.

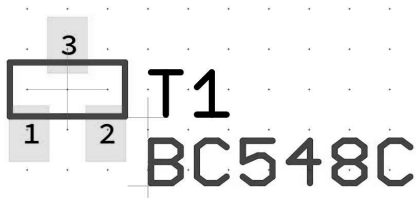

Abbildung 3.4: Gehäuse eines Transistors

[8] engl. origin

Ein komplexes Objekt ist gekennzeichnet durch:

1. seinen Ursprung oder seine Position (x;y).
2. die Seitennummer (nur bei Zeichnungen mit mehreren Seiten).
3. Drehwinkel in Grad um seinen **eigenen** Ursprung.
4. Spiegelung an x oder y-Achse seines **eigenen** Ursprungs.
5. eine mehr oder weniger große Anzahl primitiver Objekte, mit denen auch Kontouren zusammengesetzt sind.
6. Textmarken, Platzhalter, ...

3.3.2.1 Modellierung und Bibliotheken

Die Modellierung eines komplexen Objektes wird üblicherweise in einer Bibliothek vorgenommen. In unserem Demoprogramm wurde auf eine echte Bibliothek verzichtet, um die Dinge überschaubar zu halten.

In diesem Buch soll die Handhabung und Realisierung von Bibliotheken nicht betrachtet werden um den Rahmen nicht zu sprengen.

WICHTIG: Alle primitiven Objekte des komplexen Objektes werden in Bibliotheken so angelegt, als befände sich das komplexe Objekt mit seinem Ursprung auf Position (0;0).

Die finale Position des komplexen Objektes in der Zeichnung wird durch seinen Selektor (oder seine Eigenschaft) `position` bestimmt. Ein Zusatz der Position ist üblicherweise auch der Drehwinkel (oder die Rotation) um den **seinen eigenen** Ursprung. Der Drehwinkel wird in der Bibliothek meist mit Standardwert 0 Grad festgelegt. Optional kann die Spiegelung des Objektes noch die Position ergänzen. Bei der Berechnung des Begrenzungsrechtecks B und beim Zeichnen auf der Leinwand ist das zu beachten. Betreffend des Begrenzungsrechtecks sind Details in Abschnitt 5.1.1.4 zu finden. Was das Zeichnen auf der Leinwand angeht, so sei auf Abschnitt 6.6.9 und insbesondere 6.6.9.2 verwiesen.

Das Demoprogramm verwendet - wie oben schon gesagt - keine echte Bibliothek. Der Einfachheit halber sind drei komplexe Objekte fest programmiert. Beim Start des Programmes werden diese: ein Rechteck, ein Dreieck und ein Kreis in die Datenbank geladen und dargestellt (Siehe Abbildung 3.5). Ein viertes Objekt, ein Quadrat, kann durch Drücken des Knopfes `ADD` hinzugefügt werden. In einem echten CAD-System würde sich an dieser Stelle eine Bibliothek öffnen, aus welcher der Bediener dann ein Objekt heraussucht.

Die Datenbank ist in der Variablen `objects_database` enthalten. Die Prozedur `make_database_1` erzeugt die drei genannten komplexen Objekte und speichert sie in `objects_database`.

Alternativ stehen noch die Prozeduren `make_database_2`, `make_database_3` und `make_database_4` zur Verfügung, um andere Objekte zu erzeugen. Darauf soll hier aber zunächst nicht eingegangen werden. Mehr Informationen sind im Quellcode in Abschnitt 9.3 zu finden.

3.4 Zeichnungsrahmen und Dokumentationsfeld

Ein Zeichnungsrahmen (ZR) und ein Dokumentationsfeld (DF) unterscheiden ein CAD-System **elementar** von einem Malprogramm. Der Leser wird sich eventuell fragen, warum ZR und DF hier so viel Aufmerksamkeit erhalten. Eine Antwort ist: Der ZR und das DF werden wie ein Teil des Modelles behandelt. Alle ihre Einzelbestandteile existieren in CS1 und müssen wie Objekte des Modells behandelt werden, sind aber vom Maßstab M **nicht** betroffen. Die Bestandteile des ZR und DF sind im Demoprogramm Linien einer bestimmten Breite, also primitive Objekte. Bei einem echten CAD-System kommen noch Texteinträge dazu. Weitere Gründe, warum ZR und DF **zwingend nötig** sind:

- Festlegung der Größe der Seite z.B. A4 oder A3 (Durch die maßstabsgetreue Darstellung (Siehe Abschnitt 3.1) lassen sich sehr kleine oder sehr große Objekte im ZR einpassen.)
- Grob-Lokalisierung von Objekten mittels der Quadrantenleisten am Rand
- Neben der x- und y-Koordinate braucht es eine Zuordnung der Seitennummer zu Objekten, sofern eine Zeichnung aus mehreren Seiten besteht. Je nach Anwendung kann dieser Grundsatz zur Geltung kommen: Wenn Zeichnungen sich aus mehreren Seiten zusammensetzen, wird die Seitennummer ein Zusatz zur Position eines Objektes. Diese Methode hat sich im Bereich der Entwicklung von Elektronik bewährt.
- Dokumentation und Ordnung

Das Demoprogramm verwendet einen Zeichnungsrahmen sowie ein Dokumentationsfeld nur in einfachster Form so wie es in Abbildung 3.5 zu sehen ist.

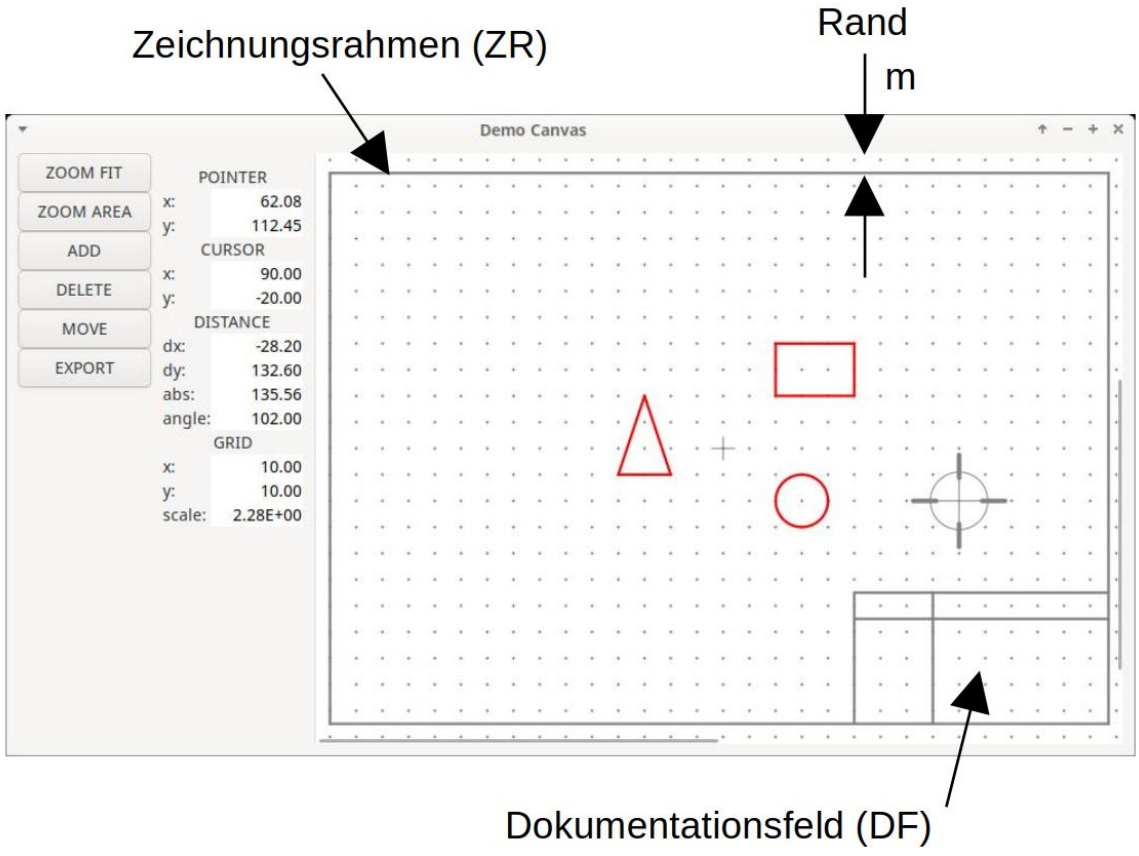

Abbildung 3.5: Zeichnungsrahmen des Demoprogrammes

Einen weitaus detaillierteren Zeichnungsrahmen mit Quadrantenleisten am Rand zeigt Abbildung 3.6. Die Quadrantenleisten erlauben es, ein Objekt grob zu lokalisieren. Beim Suchen eines Objektes kann das z.B. das Ergebnis lauten: "Widerstand R5 befindet sich auf Seite 4 in Quadrant (8/3)".

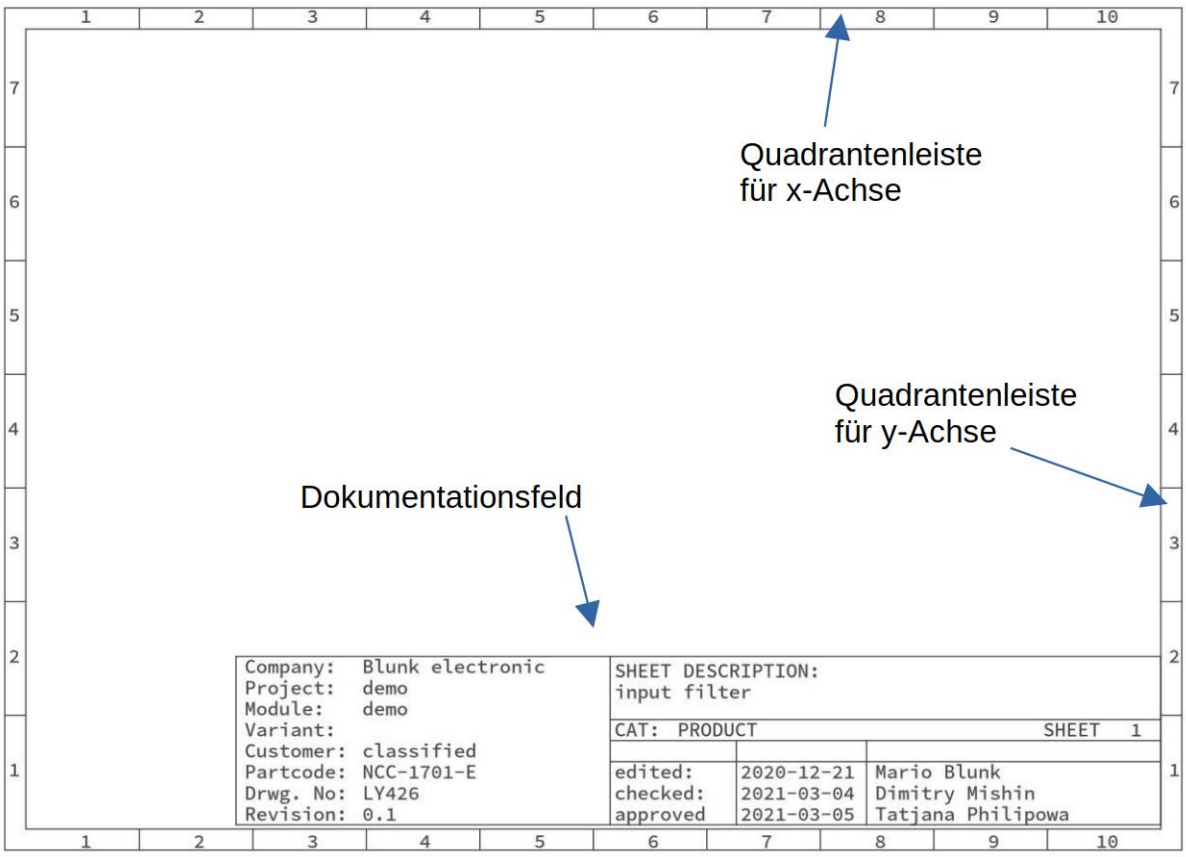

Abbildung 3.6: Detaillierter Zeichnungsrahmen

Innerhalb des ZR befindet sich das Dokumentationsfeld[9] (DF) in welchem folgende Informationen untergebracht werden **müssen**:

- Datum der letzten Änderung, Prüfung und Freigabe [10].
- Name des Zeichners, Prüfers und Freigebers
- Name der Firma, Institution, Projektname, Name des Kunden, ...
- Zeichnungsnummer, Materialnummer, ...
- Maßstab (im obigen Beispiel nicht enthalten)
- Bei mehrseitigen Plänen: Seitennummer, Bezeichnung des sichtbaren Funktionsblocks oder Submodules
- Sonstiges wie z.B. ein Firmenlogo

Für die oben genannten Informationen werden üblicherweise Platzhalter verwendet, um das Dokumentationsfeld universell zu halten.

[9] engl. title block

[10] Detailliertere Angaben zu Änderungen sollten hier nicht stehen. Statt dessen sei auf die hervorragenden Vorteile einer Versionskontrolle hingewiesen. Dazu siehe Abschnitt 11.3.

Das Dokumentationsfeld (DF) befindet sich meistens in der unteren rechten Ecke des Zeichnungsrahmens. Das DF kann aber auch eine eigene Position haben. Dadurch kann es innerhalb des ZR an beliebiger Stelle platziert werden.

3.4.1 Modellierung

Komfortable CAD-Systeme erlauben es, den ZR und das DF als Bibliothekselement anzulegen oder in einer speziellen Datei zu definieren (Klartext oder xml).

Der Einfachheit halber ist in unserem Demoprogramm der Zeichnungsrahmen und das Dokumentationsfeld kodiert in der Variablen `drawing_frame`. Die Prozedur `make_drawing_frame` generiert die primitiven Elemente des DF, des ZR, setzt die Position des ZR und speichert das alles in der Variablen `drawing_frame`. Siehe Quellcode in Abschnitt 9.2 für Details.

WICHTIG: Die Modellierung der Linien des ZR und DF in einer zukünftigen Bibliothek ist allgemein zu halten und zwar so, als wäre die Position des ZR und DF bei (0;0). Mit anderen Worten: Der ZR und das DF befinden sich mit ihrer linken unteren Ecke am Zeichnungsursprung[11].

Im Stadium der Modellierung des ZR/DF ist aber nicht immer klar, wo in der Zeichnung ZR/DF sich befinden werden. Der Zeichnungsursprung (0;0) muß nicht zwangsläufig mit der unteren linken Ecke des ZR/DF übereinstimmen (Siehe Abbildung 3.5). Die finale Position des ZR wird durch den Selektor `position` der Variablen `drawing_frame` bestimmt. Bei kreisförmigen Strukturen des Modells besteht mitunter die Forderung, daß sich die Zeichnungsmitte, also (0;0) in der Mitte des Objektes befinden soll. Das Beispiel in Abbildung 3.7 zeigt diese Situation, in der der ZR um (-103;-99) Millimeter nach links unten verschoben ist. Im Demoprogramm ist dafür die globale Variable `drawing_frame_position` vorgesehen.

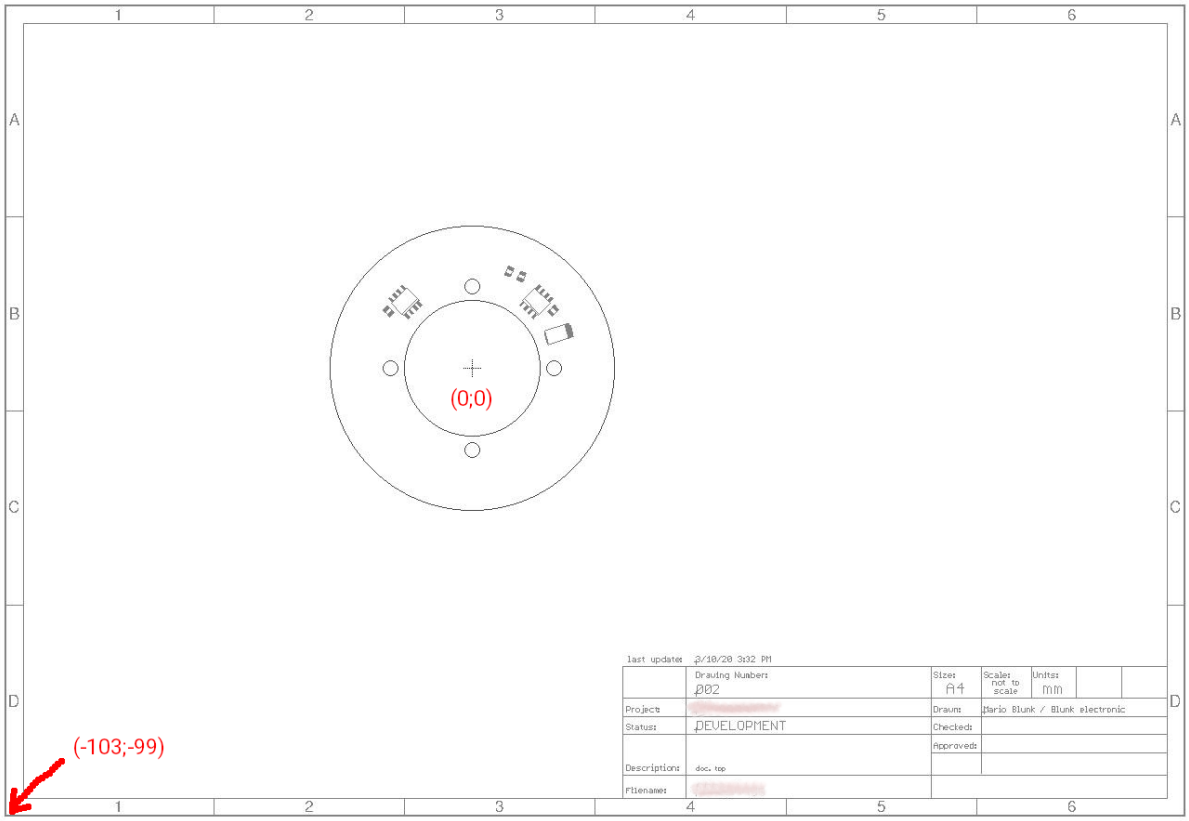

Abbildung 3.7: Zeichnungsrahmen versetzt nach links-unten

[11] Das Prinzip ist ähnlich dem der Modellierung komplexer Objekte, wie in Abschitt 3.3.2.1 beschrieben wurde.

Bei der Berechnung des Begrenzungsrechtecks B und beim Zeichnen auf der Leinwand ist das zu beachten. Betreffend des Begrenzungsrechtecks sind Details in Abschnitt 5.1.1.3 zu finden. Was das Zeichnen auf der Leinwand angeht, so sei auf Abschnitt 6.6.9.2 verwiesen.

Grundsätzlich gilt: Der ZR und das DF werden wie ein Teil des Modelles behandelt. Der Maßstab M hat auf die Position des ZR und DF **keinen** Einfluß !

3.5 Begrenzungsrechteck im Allgemeinen

In diesem Buch wird das Wort "Begrenzungsrechteck" sehr oft vorkommen. Im englischen Sprachraum und unter Programmierern von graphischen Oberflächen gibt es dafür die weit verbreitete Bezeichnung "Bounding-Box". Es handelt sich dabei um ein imaginäres Rechteck, welches ein oder mehrere Objekte umfasst. Begrenzungsrechtecke sind wichtig, um die benötigte Zeichenfläche von Objekten auf der Leinwand zu ermitteln. Abbildung 3.8 zeigt abstrahiert diese Fläche. Ein Begrenzungsrechteck ist eine Fläche der Breite w und der Höhe h. Seine Position ist dort, wo seine untere linke Ecke ist, also am Punkt P.

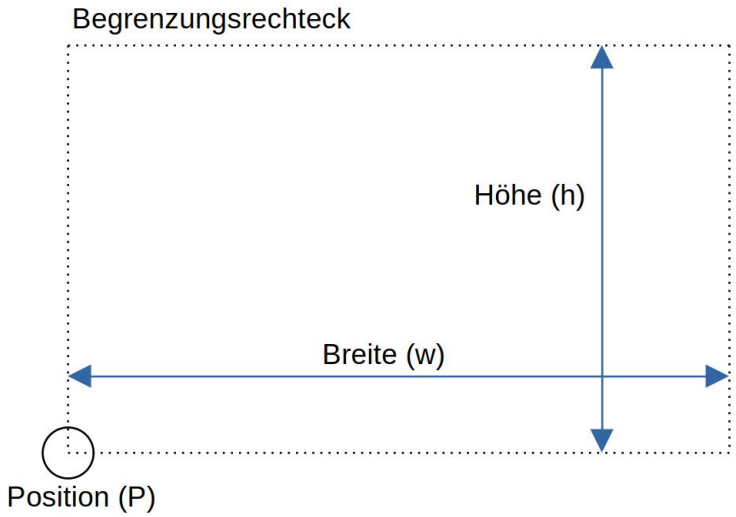

Abbildung 3.8: Begrenzungsrechteck abstrakt

Ein Begrenzungsrechteck im allgemeinen basiert auf den in Anspruch genommenen Koordinaten eines oder mehrerer Objekte:

1. der weitesten Ausdehnung nach links, also dem kleinsten verwendeten x-Wert
2. der weitesten Ausdehnung nach rechts, also dem größten verwendeten x-Wert
3. der weitesten Ausdehnung nach unten, also dem kleinsten verwendeten y-Wert
4. der weitesten Ausdehnung nach oben, also dem größten verwendeten y-Wert

Die folgende Abbildung 3.9 zeigt als Beispiel das Begrenzungsrechteck eines einzelnen Objektes, nämlich einer Linie mit einer bestimmten Breite. Beachte: Die Breite der Linie geht in die Berechnung mit ein !

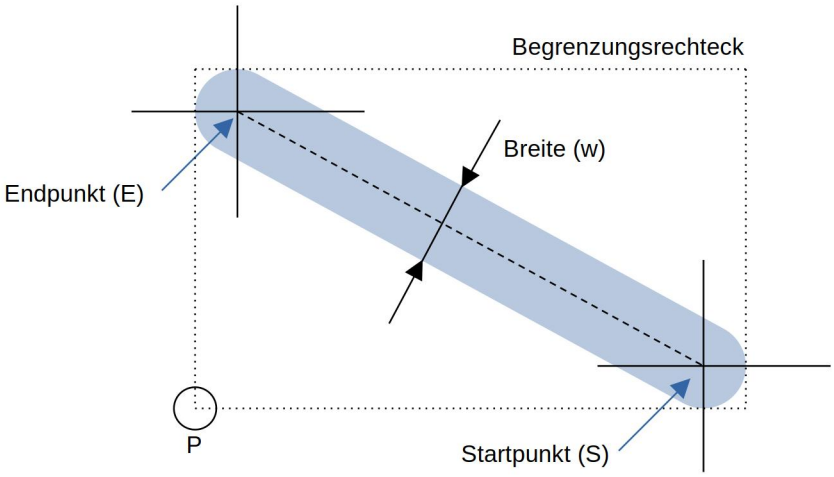

Abbildung 3.9: Begrenzungsrechteck einer Linie

Wenn mehrere Objekte von einem Begrenzungsrechteck umfaßt werden, sieht das so aus wie in Abbildung 3.10. Wird das Begrenzungsrechteck zudem noch um einen Rand[12] der Breite m ausgedehnt wird, unterscheiden wir zwischen innerem und äußerem Begrenzungsrechteck. Wir verwenden dafür die Abkürzungen IB[13] und OB[14].

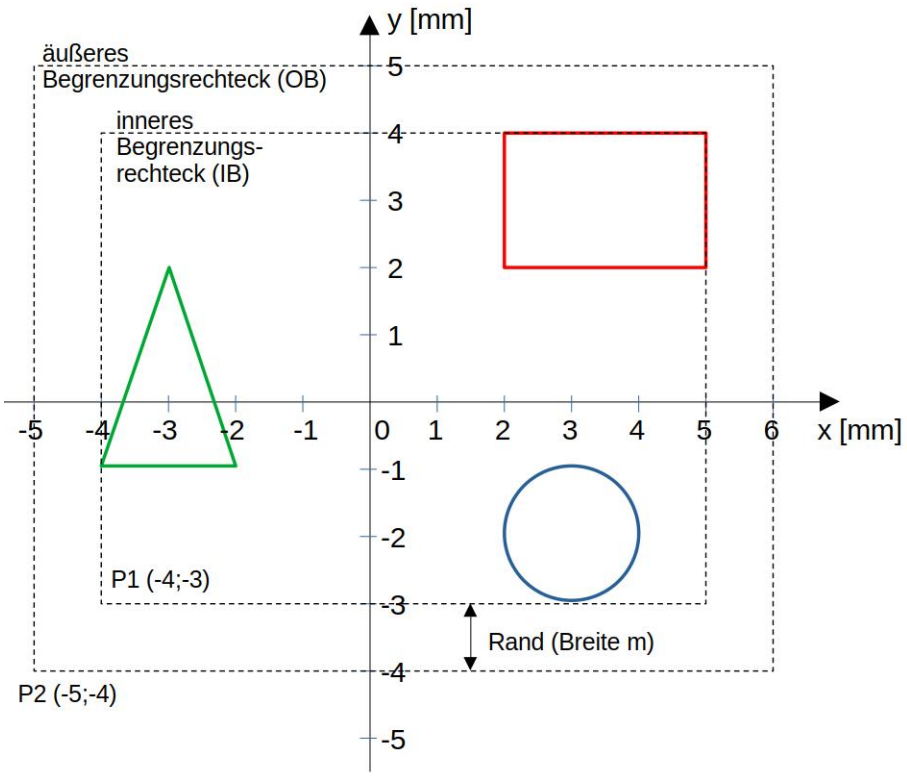

Abbildung 3.10: Begrenzungsrechteck mehrerer Objekte

Die Breite der darzustellenden Objekte in Abbildung 3.10 erstreckt sich von -4mm bis +5mm was 9mm ergibt. Die Höhe geht von -3mm bis +4mm, also sind das 7mm.

[12] engl. margin
[13] engl. inner bounding-box
[14] engl. outer bounding-box

3.6 Begrenzungsrechteck des Modells

Ein spezielles Begrenzungsrechteck ist jenes, welches **alle** darzustellenden Objektes des Modelles umfasst. Es ist für viele der in diesem Buch beschriebenen Mechanismen grundlegend und spielt eine große Rolle bei fast allem, was mit Vergrößerungseffekten zu tun hat.

Wenn vom Begrenzungsrechteck des Modelles - also aller Objekte - die Rede ist, wird der Buchstabe B als Abkürzung und Formelzeichen verwendet.

Vom Begrenzungsrechteck eingeschlossen ist außerdem noch

1. Der Zeichnungsrahmen (ZR). Dazu ist mehr in Abschitt 3.4 zu finden.
2. Der Zeichnungsrand der Breite m. Dieser ist beim Darstellen aller Objekte gut erkennbar[15] Es handelt sich dabei um den nicht bedruckbaren Bereich um den ZR herum. Das Demoprogramm verwendet für die Breite des Randes die globale Variable `margin`. Siehe Quellcode in Abschnitt 9.2.

Wie wird das Begrenzungsrechteck B gebildet ?

Wir betrachten Abbildung 3.11. Zunächst haben wir das innere Begrenzungsrechteck (IB), das die Objekte des Modells und den Zeichnungsrahmen umfaßt. Wird nun das IB um die Breite m des Randes erweitert, ergibt sich das äußere Begrenzungsrechteck (OB). Das OB ist nun das Begrenzungsrechteck B.

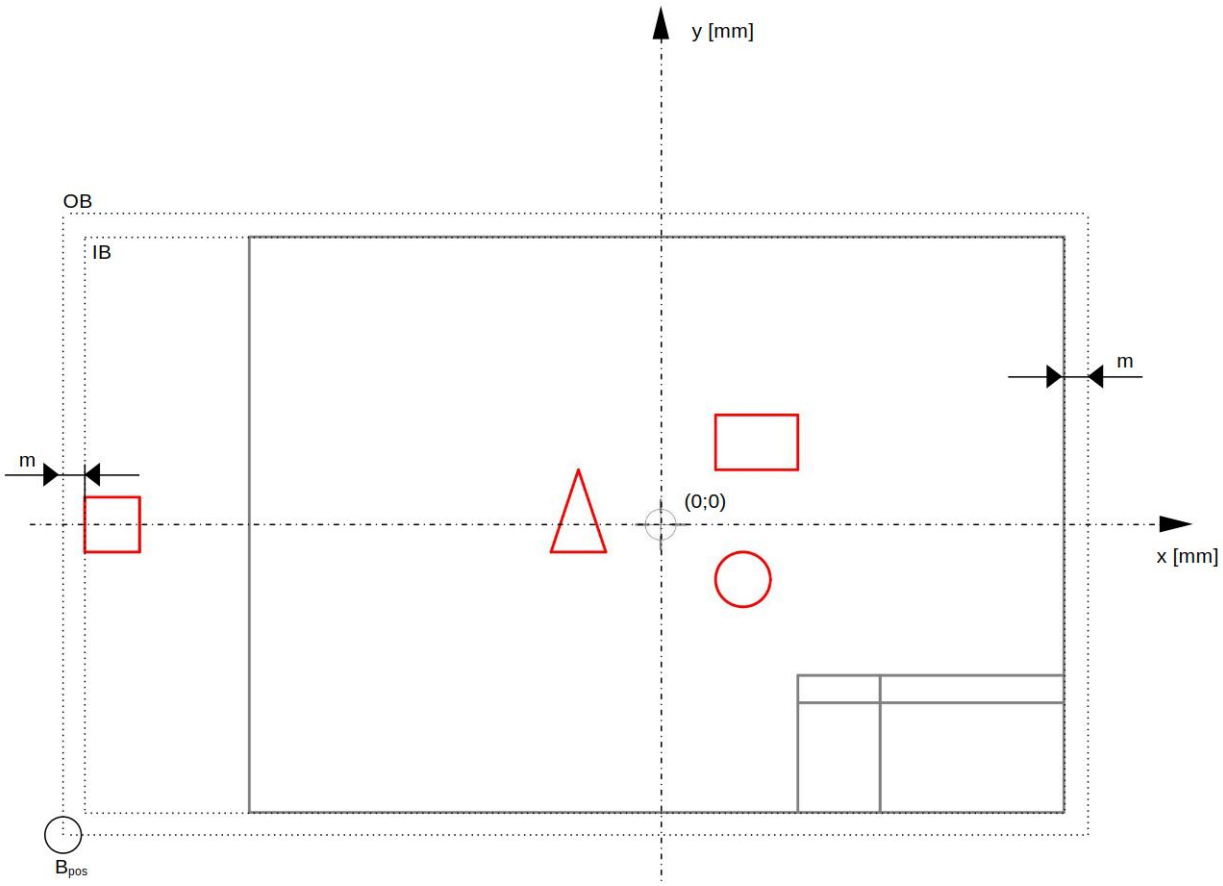

Abbildung 3.11: Begrenzungsrechteck B

Der Leser könnte nun fragen, ob es ausreichen würde, das Begrenzungsrechteck allein aufgrund des Zeichnungsrahmens zu berechnen, weil ja sowieso alle Objekte innerhalb des

[15] Dahinter steckt der Vorgang des sogenannten Einpassens aller Objekte.

Zeichnungsrahmens liegen. Das ist leider nicht immer der Fall. Solange eine Zeichnung noch nicht fertig ist, dürfen Objekte vorübergehend durchaus auch außerhalb des Zeichnungsrahmens sein. Um diesen Fall abzudecken, ist links vom Zeichnungsrahmen demonstrativ ein Objekt, nämlich ein Quadrat eingezeichnet.

Details zur Ermittlung des Begrenzungsrechteckes B sind in Abschitt 5.1 zu finden. Siehe auch den Quellcode in Abschnitt 9.6.

3.6.1 Abmessungen des Begrenzungsrechtecks

Das Begrenzungsrechteck B basiert auf den vom Modell beanspruchten Koordinaten, dem Zeichnungsrahmen (ZR) und auf dem Zeichnungsrand. Diese Werte sind für die Ermittlung der Abmessungen von B nötig:

1. kleinste verwendete x-Position
2. größte verwendete x-Position
3. kleinste verwendete y-Position
4. größte verwendete y-Position

Die Breite der darzustellenden Objekte ist der Wert I_w. In Abbildung 3.11 ist das die Breite des inneren Begrenzungsrechtecks (IB). Die Höhe der darzustellenden Objekte ist der Wert I_h. Das entspricht der Höhe des inneren Begrenzungsrechteckes. Der Zeichnungsrand hat die Breite m.

B hat also die Gesamtbreite

$$B_{width} = I_w + 2m \tag{3.14}$$

und die Gesamthöhe

$$B_{height} = I_h + 2m \tag{3.15}$$

Die Dimensionen von B sind Vorausetzung für:

1. die Berechnung der initialen Größe des Rollfensters, welches einen Blick auf die Leinwand ermöglicht. Siehe Abschitte 3.8 und 5.4.1 für Details.
2. die Festlegung der Anschläge[16] des vertikalen und horizontalen Rollbalkens. Mehr dazu ist in den Abschnitten 3.10 und 5.3 zu finden.

3.6.2 Grenzen des Begrenzungsrechtecks

Es ist eine maximale Breite B_{w_max} sowie eine maximale Höhe B_{h_max} als Systemgrenze festzulegen. Beide hängen von der konkreten Anwendung ab. Es geht also um die Abmessungen der realen Welt - also des Modelles - das wir abbilden wollen. Bei diese Abmessungen kann man sich an gebräuchlichen Papiergrößen wie A4, A3, A2, u.s.w. orientieren. So hat beispielsweise A3 die Abmessungen 297 $\times$ 420 mm im Hochformat. Das Demoprogramm verwendet die Abmessungen 2000 $\times$ 1000mm, also Querformat.

Im Zusammenhang mit der Bestimmung der Größe der Leinwand sind diese Grenzwerte wichtig. Wie die Größe der Leinwand berechnet wird ist Thema in Abschnitt 5.5.1.2.

[16] engl. limits

3.6.3 Position des Begrenzungsrechtecks

Das Begrenzungsrechteck hat außerdem auch eine Position. Das ist die **untere linke Ecke** - ausgedrückt in Modellkoordinaten. Wir bezeichnen diese Position mit B_{pos}. Der kleinste verwendete x-Wert und der kleinste verwendete y-Wert der darzustellenden Objekte (ZR eingeschlossen) bestimmen die Position des Begrenzungsrechtecks B.

Im Beispiel in Abbildung 3.12 ist die Position des Begrenzungsrechtecks B_{pos} unten-links markiert. Weil der Rand mit der Breite m innerhalb des Begrenzungsrechtecks liegt, ist B_{pos} um m nach links-unten verschoben. Die X- und Y-Achse des Koordinatensystems ist jeweils mit Strich-Punkt-Linie gezeichnet. Die Zeichnungsmitte, der Ursprung ist mit (0;0) markiert[17].

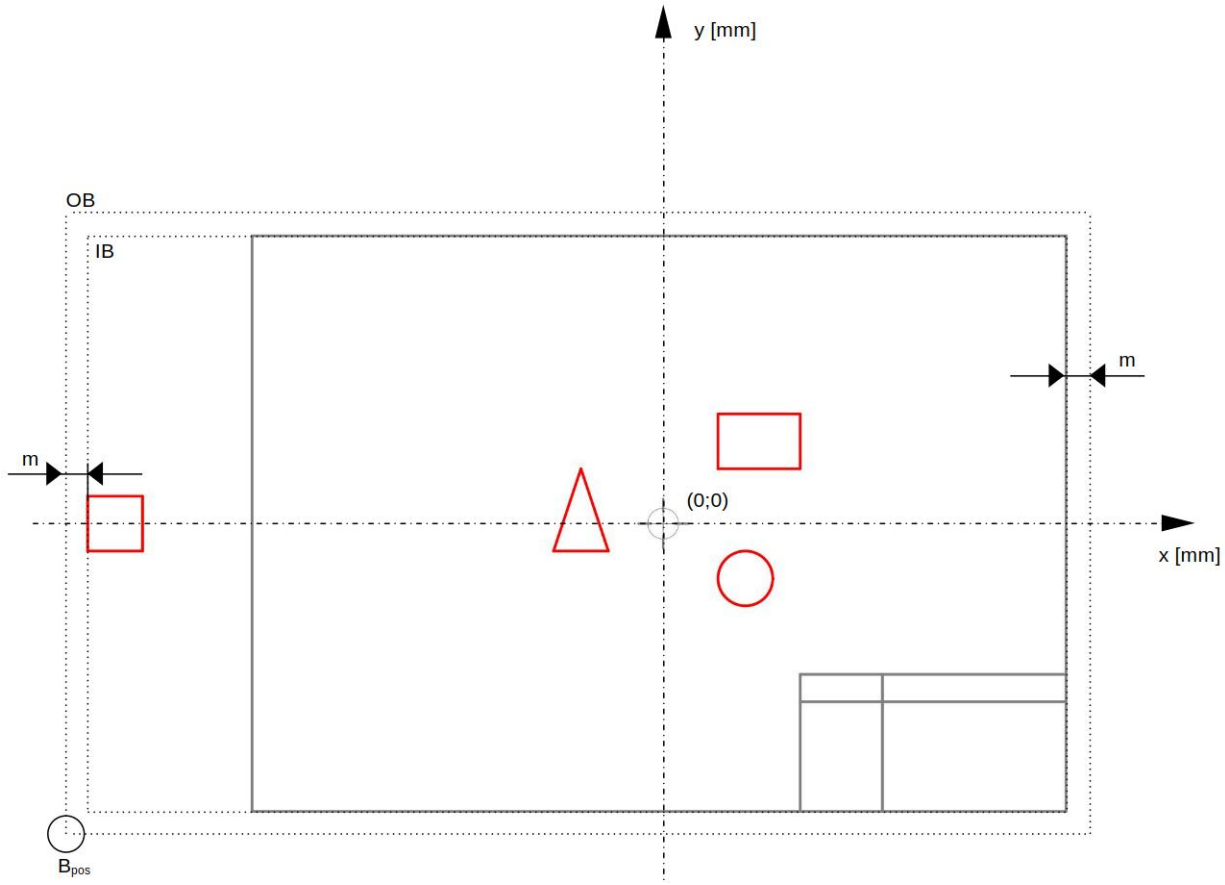

Abbildung 3.12: Position des Begrenzungsrechtecks

Warum ist die Position B_{pos} wichtig ?

Im Modell können auch negative Koordinaten existieren. So befindet sich z.B. in Abbildung 3.12 der Kreis auf einer negativen y-Position. Allein weil die Zeichnungsmitte (0;0) etwa in der Mitte des Zeichnungsrahmens liegt, ergeben sich negative x- und y-Werte für einige Objekte (z.B. für den Kreis und das Dreieck) sowie für die untere Hälfte des Zeichnungsrahmens.

Wir brauchen für die Umrechnung von Modellkoordinaten in Leinwandkoordinaten als Zwischenschritt eine **Normierung**. Diese verschiebt alle Modellkoordinaten um den Offset B_{offs} in den 1.Quadranten. Der 1.Quadrant befindet sich rechts der Y-Achse und oberhalb der X-Achse. Nur dort sind alle Werte positiv.

Der Grund für diese Normierung ist, daß die nachfolgende Umrechnung in Leinwandkoordinaten (CS2) nur mit positiven Zahlen funktioniert.

[17] Zudem befindet sich der Cursor auch auf (0;0). Er ist an dem kleinen Kreis um den Ursprung herum zu erkennen.

Im Ergebnis der Normierung sind dann alle Koordinaten positive Zahlen, wie in Abbildung 3.13 gut zu sehen ist.

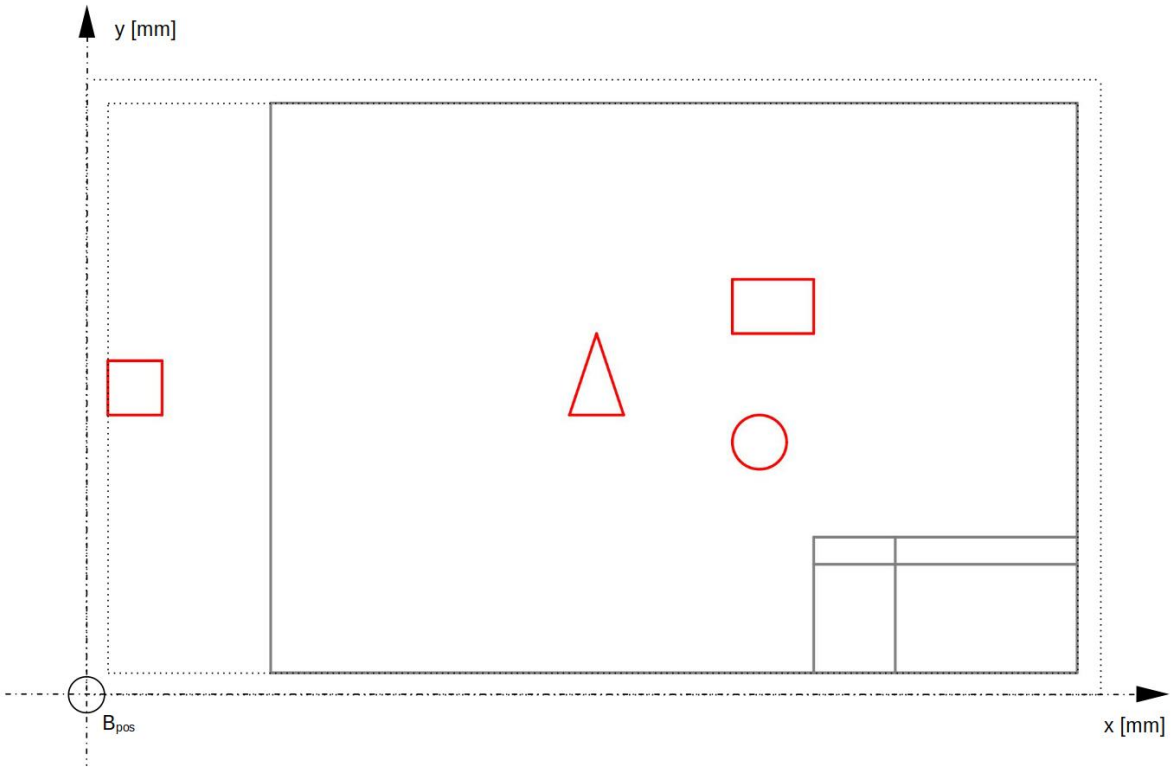

Abbildung 3.13: Modellkoordinaten normiert

Um den benötigten Offset zu erhalten, gilt diese einfache Beziehung:

$$\vec{B}_{offs} = -1 \cdot \vec{B}_{pos} \tag{3.16}$$

In Worten: B_{offs} ergibt sich, indem wir das Vorzeichen der x- und y-Position des Begrenzungsrechtecks einfach umdrehen.

Um den Offset B_{offs} sind also alle Modellkoordinaten zu verschieben, bevor in Leinwandkoordinaten umgerechet wird. Umgekehrt, wenn von Leinwandkoordinaten nach Modellkoordinaten umgerechet wird, sind die Modellkoordinaten um B_{pos} zurück zu verschieben.

Durch die Normierung erhalten wir virtuelle (oder normierte) Modellkoordinaten. Wir müssen nun also die Modellkoordinaten unterscheiden in:

1. **reale** Koordinaten (ursprüngliche, tatsächliche Position von Objekten)
2. **virtuelle** Koordinaten (bedingt durch Verschiebung um B_{offs} in den 1. Quadranten)

Für die Umrechnung eines realen Modellpunktes P_r in einen virtuellen Modellpunkt P_v gilt also vereinfacht:

$$\vec{P}_v = \vec{P}_r - \vec{B}_{pos} \tag{3.17}$$

Das Umrechnen eines virtuellen Modellpunktes P_v zurück in einen realen Modellpunkt P_r erfolgt nach dieser Beziehung:

$$\vec{P}_r = \vec{P}_v + \vec{B}_{pos} \tag{3.18}$$

Zur direkten Umrechnung zwischen realen und virtuellen Modellkoordinaten siehe auch Abschnitt 4.2. Die Konvertierung von Modellpunkten nach Leinwandpunkten und zurück beinhaltet ebenso diesen Vorgang. Dazu siehe Abschitte 4.3 und 4.4.

3.7 Basis-Offset

Zwischen virtuellen Modellkoordinaten und Leinwandkoordinaten besteht ein sogenannter Basis-Offset[18] F in x und y. Seine Aufgabe ist es, das Modell so auf der Leinwand zu positionieren, daß es selbst bei maximalem Vergrößerungsfaktor S und jedem beliebigen Zentrum der Vergrößerung vollständig auf die Leinwand paßt. Abbildung 3.14 soll den Zusammenhang anschaulich machen.

Es ist zu bedenken, daß bei jedem Vergrößern oder Verkleinern oder bei jeder Betätigung der Rollbalken **alle** Objekte auf die Leinwand gemalt werden.

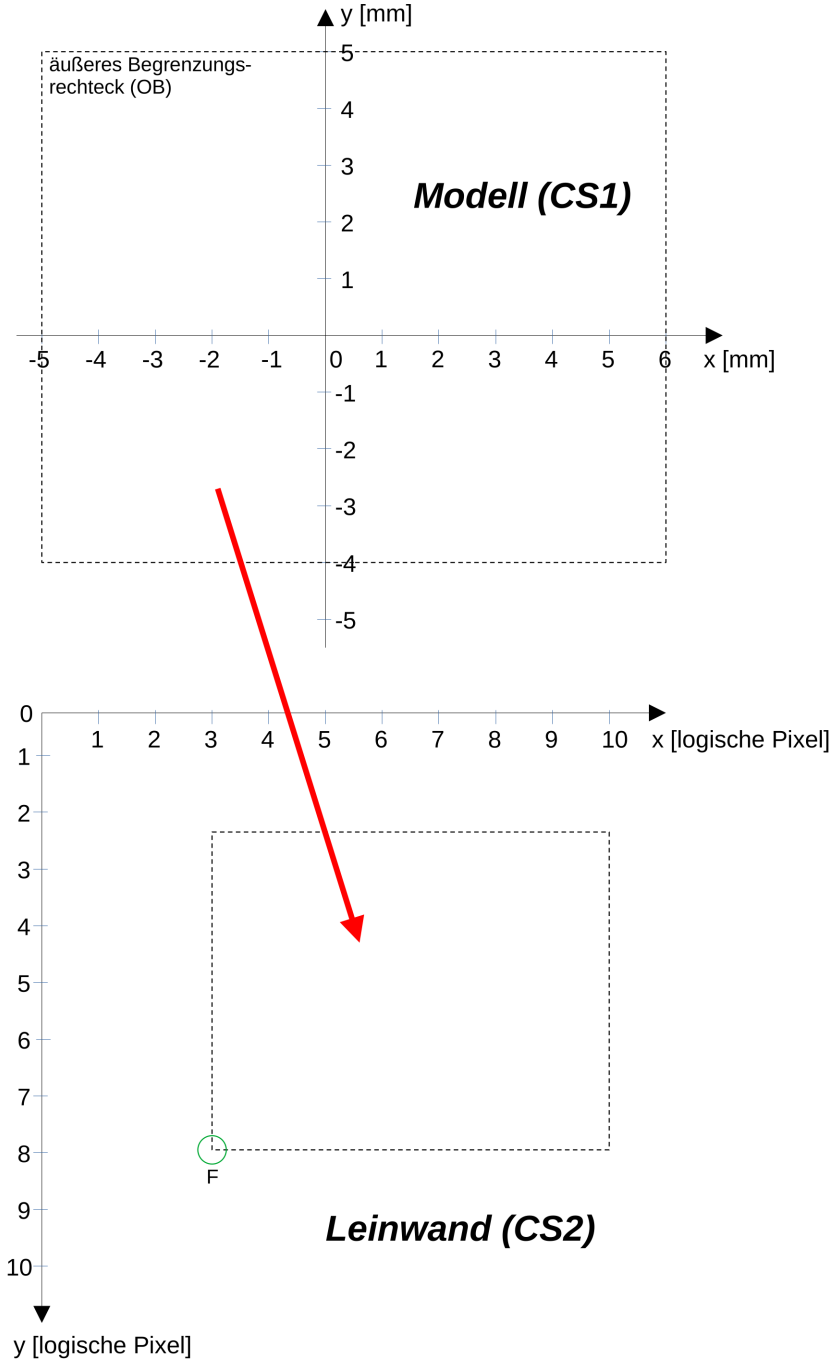

Abbildung 3.14: Basis-Offset

[18] engl. base-offset

Die Einheit des Basis-Offsets F ist dp, also logische Pixel. F ist abhängig von:

1. der Breite und Höhe des Begrenzungsrechtecks B,

2. dem maximalen Vergrößerungsfaktor S_{max}.

Der Basis-Offset F ist ein grundlegender Parameter für die Umrechnung zwischen Modell- und Leinwandkoordinaten. Seine Berechnung wird in Abschnitt 5.2 erläutert.

Im Zusammenhang mit Größenveränderungen des Hauptfensters erfährt der Basis-Offset eine weitere Bedeutung. Es sei an dieser Stelle auf Abschnitt 6.5 verwiesen. Siehe Quellcode in Abschnitt 9.7 für Details.

3.8 Initiale Größe des Rollfensters

In Abbildung 3.15 ist das Rollfenster zu sehen. In *GTK3* heißt dieses Fenster "scrolled window" (in QT "QScrollArea" ?). Hinter dem Rollfenster befindet sich die Leinwand. Das Rollfenster erlaubt den Blick auf die Leinwand, die durchaus viel größer sein kann als das Rollfenster. Siehe ergänzend auch Abbildung 1.3.

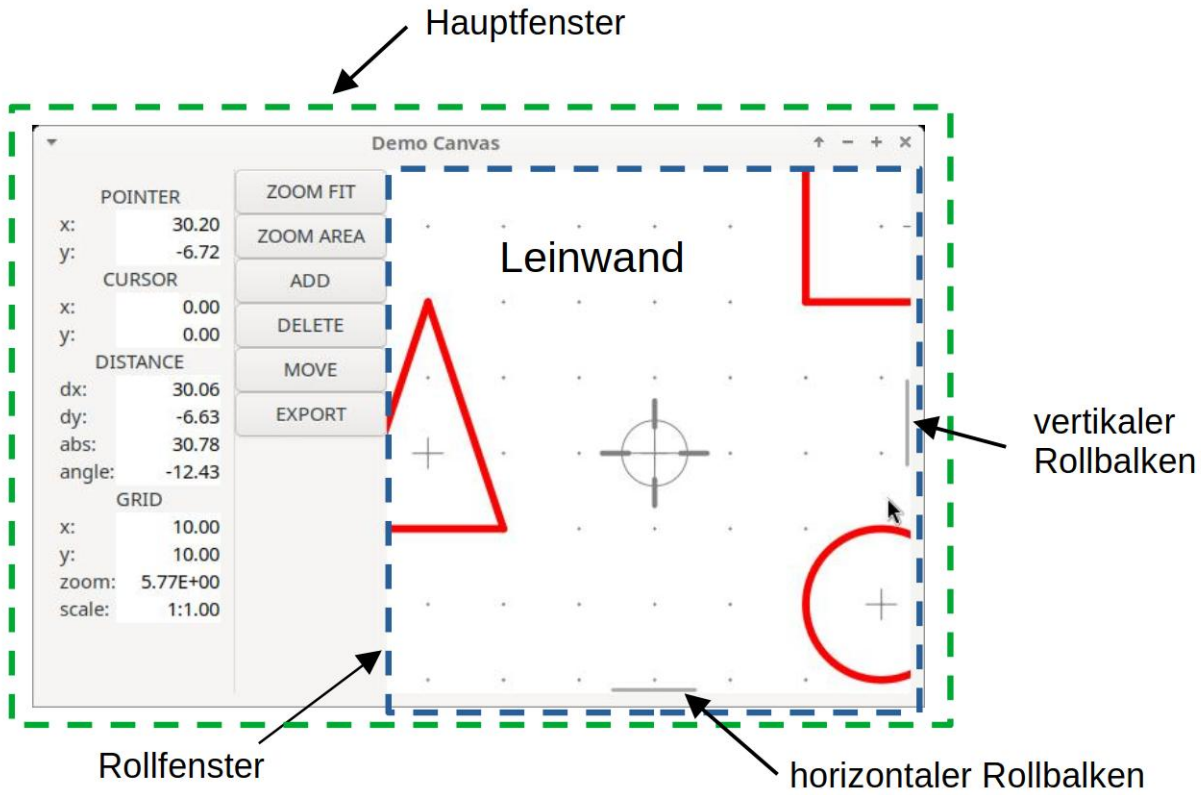

Abbildung 3.15: Initialgröße des Rollfensters

Die Dimensionen des Rollfensters werden in ganzzahligen Gerät-Bildpunkten, also dp, angegeben (CS3). Seine initialen Abmessungen können auf zwei Wegen festgelegt werden:

1. Variabel: Die initiale Breite und Höhe entspricht der Breite und Höhe des Begrenzungsrechtecks B bei einem Vergrößerungsfaktor S von 1,0. Nach dem Systemstart sind also alle Objekte sichtbar. Der Nachteil dieser Methode ist, daß schwer vorhersehbar ist, wie groß das Rollfenster wird, weil die Abmessungen von B unbekannt sind. Das wirkt sich wiederum auf die Größe des Hauptfensters aus. Dieser Ansatz wurde verworfen.

2. Statisch: Werden die initialen Abmessungen - also Höhe und Breite - starr vorgegeben, dann ist die Größe des Rollfenster im Voraus bekannt. Somit ist dann auch die Größe des Hauptfensters indirekt beherrschbar. Um alle Objekte durch das Rollfenster bei Systemstart zu sehen, wird mittels des Vergrößerungsfaktors S entsprechend vergrößert oder verkleinert. Der Vergrößerungsfaktor ist nach Systemstart also nicht zwingend 1,0. Dieser Ansatz wurde weiterverfolgt und erwies sich als der Zweckmäßigere. In Abschnitt 5.4.1 wird diese Lösung genauer erklärt.

Egal welche Methode verwendet wird: Es muß eine minimale Höhe und Breite eingehalten werden, damit der Bediener überhaupt etwas von der Leinwand sieht. Diese Initialgröße wird beim Systemstart verwendet. Im selben Moment wird, entsprechend der statischen Methode, die gesamte Zeichnung in das Rollfenster eingepaßt. Im Demoprogramm sind diese minimalen und zugleich initialen Abmessungen in der globalen Konstante `swin_size_initial` festgehalten. Siehe Quellcode in Abschnitt 9.10 für Details.

Achtung! Bei der Festlegung der minimalen Abmessungen ist außerdem eine weitere Forderung zu erfüllen. Anhand Abbildung 3.16 soll das Problem erklärt werden: Die minimale Höhe h_c des Rollfensters muß größer sein als die Summe h_e der Höhen aller neben dem Rollfenster platzierten Elemente[19]. Ebenso muß die minimale Breite w_c des Rollfensters größer sein als die Summe aller Breiten der Elemente über oder unter dem Rollfenster. In der Abbildung befinden sich oberhalb oder unterhalb des Rollfensters keine Elemente, sodaß die Breite w_e nicht existiert. Anderenfalls müßte darauf geachtet werden.

Wenn diese Forderungen nicht erfüllt sind, kann die Andordnung der Widgets im Hauptfenster gestört werden, die Leinwand "einfrieren" und somit keine Signale mehr senden. Letzterer Effekt wurde unter *GTK3* beobachtet.

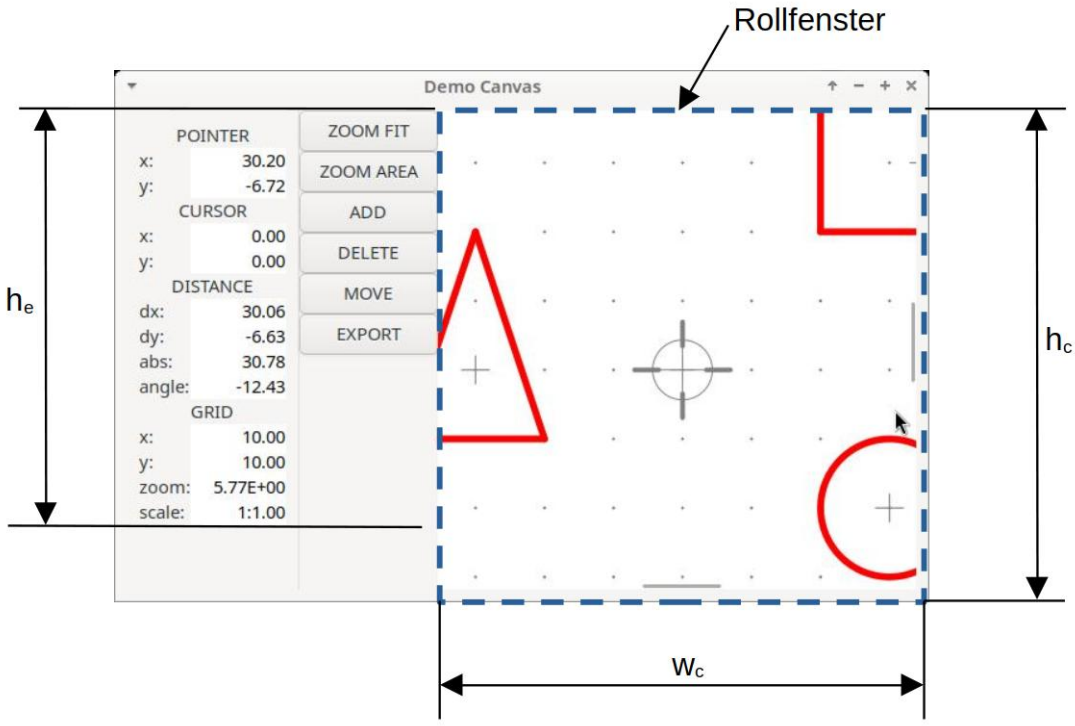

Abbildung 3.16: Minimalabmessungen des Rollfensters

[19] engl. widgets

3.9 Sichtbarer Bereich des Modells

Der sichtbare Bereich[20] ist eine Fläche des Modelles, die für den Bediener durch das Rollfenster sichtbar ist. Es handelt sich um ein Rechteck. Seine Position ist dort, wo seine untere linke Ecke ist. Es hat eine Breite W und eine Höhe H. Die Position und seine Abmessungen werden in Modellkoordinaten angegeben (CS1). Siehe Quellcode in Abschnitt 9.13 für Details.

Der sichtbare Bereich ist wichtig, um sämtliche Zeichenoperationen auf der Leinwand auf ein Minimum zu reduzieren. Ziel ist es, nur das zu zeichnen, was sichtbar ist. Davon betroffen ist:

1. das Raster[21]
2. die eigentlichen Objekte des Modells
3. der Zeichnungsrahmen (ZR) und Dokumentationsfeld (DF)

Ohne diese Minimierung würde jedes Objekt - unabhägig davon ob es sichtbar ist oder nicht - auf die Leinwand gezeichnet werden. Die Folge wäre ein erheblicher Zeitverlust beim Zeichnen und Aktualisieren des Bildes. Weil der sichtbare Bereich ein Rechteck ist, besteht er also aus diesen Komponenten:

- Position A_{pos}
- Breite A_{width}
- Höhe A_{height}

Der sichtbare Bereich ist abhängig von

1. dem Einstellwert V des horizontalen und vertikalen Rollbalkens,
2. dem Vergrößerungsfaktor S,
3. der Breite W und Höhe H des Rollfensters (nicht die Seitenbreite P der Rollbalken !)

Zur Berechnung des sichtbaren Bereiches siehe Abschnitt 6.6.1.

Der sichtbare Bereich kann auch zum Auffinden von Objekten genutzt werden. Wenn der Bediener nach dem Objekt X sucht, könnte das Ergebnis der Suche eine Nachricht wie: "Das gesuchte Objekt X befindet sich außerhalb des sichtbaren Bereiches auf Position (x;y)."

[20] engl. visible area
[21] engl. grid

3.10 Rollbalken

Rollbalken[22] sind mit *GTK3* nur mittels eines Rollfensters realisierbar. Sie machen auch nur durch ein Rollfenster Sinn. Die Rollbalken am Rand des Rollfensters erlauben dem Bediener, das Rollfenster vor der Leinwand in horizontaler wie auch vertikaler Richtung zu verschieben. Auf diese Weise kann jeder Teil der Leinwand, auch bei maximaler Vergrößerung, angezeigt werden. Eine weitere Forderung an die Rollbalken ist, daß unabhägig vom Zentrum einer Vergrößerung oder Verkleinerung jeder Teil der Leinwand erreichbar sein muß. Diese Funktionen sind ein Selbstverständlichkeit in vielen Anwendungen. Die nötigen Mechanismen im Hintergrund jedoch sind weit kompexer als es scheint.

Zunächst einige Erklärungen zu den verwendeten Begriffen und Abkürzungen. In Abbildung 3.10 sind die Rollbalken abstrahiert zu sehen. Der Spanne des Verschiebebereiches beginnt an einem unteren Anschlag L[23] und endet an einem oberen Anschlag U[24]. Der Einstellwert V[25] ist die momentane Position des Rollbalkens. Ab diesem Wert beginnt der sichtbare Bereich der Leinwand. Der sichtbare Bereich P[26] endet dort, wo der Rollbalken endet [27].

Dieses Schema wird in GTK3 und QT verwendet. In GTK wird für L, U, V und P der Zahlentyp `gdouble` verwendet. Es handelt sich hier - wie auch bei den Leinwandkoordinaten (CS2) - um sogenannte `logische Pixel`. Die Einheit ist also `lp`. QT verwendet stattdessen den Typ `integer`, also ganze Zahlen.

Weil für L, U, V und P nur positive Zahlen erlaubt sind, verwendet das Demoprogramm den dafür zugeschnittenen Datentyp `type_logical_pixels_positive`. Siehe dazu den Quellcode in den Abschnitten 9.19 und 9.10.

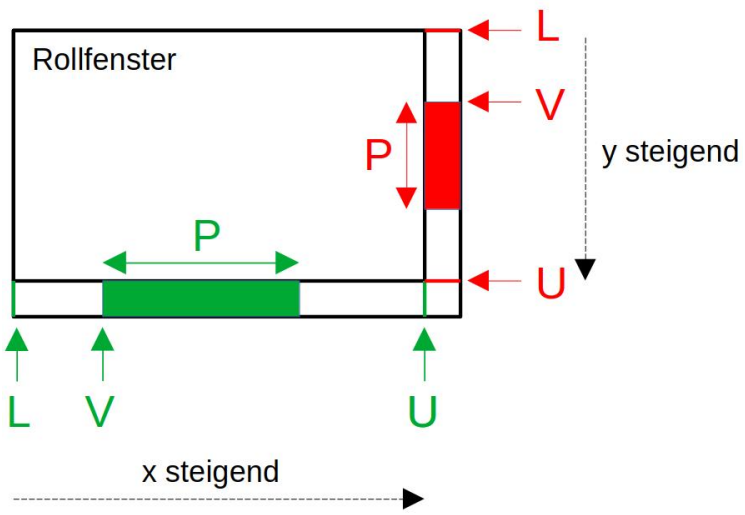

Abbildung 3.17: Rollbalken

Achtung: Betreffend des vertikalen Rollbalkens ist wieder der leidige Umstand zu beachten, daß die Werte der y-Achse von oben nach unten steigen. Der untere Anschlag des vertikalen Rollbalkens ist also oben !

[22] engl. scrollbars
[23] engl. lower limit
[24] engl. upper limit
[25] GTK verwendet das englische Wort `value`
[26] GTK verwendet die Bezeichnung `page size`
[27] Der sichtbare Bereich der Leinwand ist nicht dasselbe wie der sichtbare Bereich des Modells (siehe 3.9) !

3.11 Zeigerposition (Mausposition)

Wenn der Zeiger[28] sich innerhalb des Rollfensters befindet, kann dessen Position auf der Leinwand im Zahlentyp `gint` - also mit ganzen Zahlen - angefragt werden[29].

Es handelt sich um Gerät-Bildpunkte, also ist die Einheit `dp`. Die Position des Zeigers in Form von Gerät-Bildpunkten ist im Zusammenhang mit CAD-Systemen nicht brauchbar[30]. Statt dessen muß die Zeigerposition fortlaufend in reale Modellkoordinaten (CS1) umgerechnet und angezeigt werden. Das Demoprogramm hat dafür die Koordinatenanzeige links des Rollfensters (Siehe Abbildung 1.1).

Im Demoprogramm ist dieser Vorgang anschaulich in der Funktion `cb_canvas_mouse_moved` zu sehen. Siehe Quellcode in Abschnitt 9.8.

3.12 Cursorposition

Der Cursor ist für ein CAD-System nicht zwingend nötig, jedoch ist er eine hilfreiche Sache. Manche Systeme unterscheiden nicht zwischen Mauszeiger und Cursor. Die langjährige Erfahrung mit CAD-Systemen ließ mich die Notwendigkeit und den Sinn eines separaten Cursors erkennen. Das hier vorgestellte Demoprogramm hat einen separaten Cursor der mittels der Pfeiltasten (rechts, links, hoch, runter) bewegt wird. Die Vorteile sind:

1. Einfache und sichere Platzierung von Objekten, insbesondere, wenn keine Maus sondern nur ein Touchpad zur Verfügung steht. Wer mit einem Notebook in einem Zug arbeiten muß, kennt das Problem.

2. Einfache Entfernungsmessungen zwischen Cursor und Mauszeiger

3. Vergrößern und Verkleinern an der Position des Cursors

Die Position des Cursors wird in Modellkoordinaten (CS1) angegeben.

Wenn ein Cursor verwendet wird, sollte dieses Verhalten realisert werden: Sobald der Bediener irgendwo auf die Leinwand klickt, wird der Cursor auf den Rasterpunkt gesetzt, der der Zeigerposition am nahesten ist. Es handelt sich hierbei um eine Art von Rundungsoperation zum nächstgelegenen Vielfachen des Rasters hin[31]. Ausnahme: Wenn eine ausgewählte Fläche vergrößert werden soll (Siehe Abschitt 8.1.4), bleibt der Cursor an seiner Position.

Das Demoprogramm zeigt die Position des Cursors links vom Rollfenster an (Siehe Abbildung 1.1). Komfortable Systeme erlauben sogar, den Cursor mittels Kommandozeile auf einen genau spezifizierten Punkt zu setzen, was allein mit dem Mauszeiger sehr mühsam ist. Im Gegensatz zum Mauszeiger darf sich der Cursor auch außerhalb des sichtbaren Bereiches befinden. Wie der Cursor auf die Leinwand gezeichnet wird, ist in Abschnitt 6.6.5 nachzulesen. Die Handhabung des Cursors ist in Abschnitt 8.1.2 beschrieben.

Abbildung 3.18 zeigt ein Beispiel, wie der Cursor aussehen könnte. Er ist eindeutig unterscheidbar vom Mauszeiger und vom Zeichnungsursprung.

Siehe Quellcode in Abschnitt 9.16 für Details.

---

[28] engl. pointer

[29] Korrekterweise dürfte der Typ `gint` nicht verwendet werden, weil dieser auch negative ganze Zahlen einschließt. Es gibt aber keine negativen Zeigerkoordinaten ! Statt dessen müßte der Zahlentyp `natural` - in der Programmiersprache `Ada` üblich - benutzt werden.

[30] In Bildbearbeitungsprogrammen hingegen - wie z.B. `Gimp` - wird die Zeigerposition dem Anwender direkt angezeigt, sofern der Anwender das wünscht.

[31] Im englischen Sprachgbrauch wird das `snap to grid` genannt

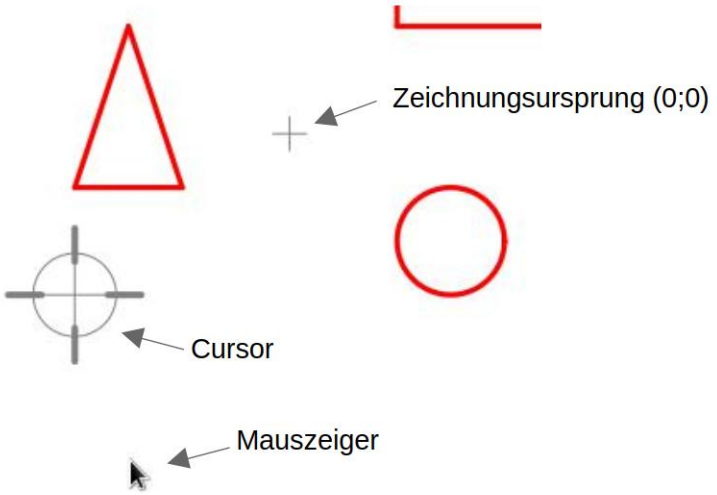

Abbildung 3.18: Cursor, Ursprung, Mauszeiger

3.13 Skalierpunkt

Der Skalierpunkt ist der Punkt im Modell oder auf der Leinwand, an dem der Bediener eine Vergrößerung[32] oder Verkleinerung[33] vornimmt. Es handelt sich um das Zentrum der Skalierung. Um diesen Punkt herum dehnt sich Bild aus oder schrumpft es. Der Skalierpunkt kann festgelegt werden durch:

1. die Cursorposition in CS1 (sofern ein Cursor verwendet wird)
2. den Mauszeiger im CS2

In Abschnitt 8.1.2 ist beschrieben, wie im Demoprogramm vergrößert oder verkleinert wird.

[32] engl. zoom-in
[33] engl. zoom-out

3.14 Translation-Offset

Der Translation-Offset[34] T ist eine der **wichtigsten** Größen. Er ist im Zusammenhang mit Skalierung - also Vergößern und Verkleinern - elementar. Er ist bedeutend für folgende Operationen:

- Umrechnung zwischen Modell- und Leinwandkoordinaten (CS1 und CS2) (Siehe Abschnitte 4.3 und 4.4).
- Verkleinerung von einen bestimmten Punkt P weg oder Vergrößerung zu einem bestimmten Punkt P hin. Mittels T werden alle Objekte beim Zeichnen auf die Leinwand (CS2) so versetzt, daß der Bediener den Eindruck bekommt, das Bild würde sich um diesen Punkt P herum ausdehnen oder schrumpfen (Siehe Details in Abschnitt 6.3).
- Verschiebung des Bildes innerhalb des Rollfensters - unabhägig von den Einstellungen der Rollbalken. Im Zusammenhang mit dem Einpassen von Objekten ist das wichtig (Siehe Abschnitt 5.4.2).

Die Einheit für T ist `lp`, also logische Pixel. Weil x und y auch negativ sein können, wird für T der Datentyp `type_logical_pixels_vector` verwendet. Siehe Quellcode in Abschnitt 9.22.

Die Translation findet folglich nur im Koordinatensystem 2 statt (CS2).

Beispiel: Gegeben sei ein Translation-Offset T

$$\vec{T} = \begin{pmatrix} 2,0 \\ -10,0 \end{pmatrix} lp \tag{3.19}$$

Wird z.B. der ursprüngliche Punkt C1 (10,0;5,0) um den Wert T versetzt, so wird er zu C2:

$$\vec{C2} = \vec{C1} + \vec{T} \tag{3.20}$$

$$\vec{C2} = \begin{pmatrix} 10,0 \\ 5,0 \end{pmatrix} lp + \begin{pmatrix} 2,0 \\ -10,0 \end{pmatrix} lp \tag{3.21}$$

$$\vec{C2} = \begin{pmatrix} 12,0 \\ -5,0 \end{pmatrix} lp \tag{3.22}$$

Siehe Quellcode in Abschnitt 9.22 für Details.

[34] engl. translate-offset

Kapitel 4

Umrechnung zwischen Koordinatensystemen

Positionen, Breiten oder Längen werden in allen drei Systemen CS1, CS2 und CS3 gehandhabt. Zentrale Bedeutung hat die Konvertierung zwischen den Systemen. Das ist der Fall wenn:

- der Bediener die Ansicht vergrößert oder verkleinert,
- der Bediener das Rollfenster verschiebt[1],
- der Bediener die Größe des Hauptfensters ändert (Minimieren, Maximieren, Höhe oder Breite ändern),
- der Bediener den Mauszeiger bewegt und fortlaufend die zugehörigen Modellkoordinaten angezeigt werden.
- die Entfernung zwischen Cursor und Mauszeiger berechnet und angezeigt wird.

Darüber hinaus ist zu beachten, daß hier ständig zwischen den Datentypen die von CS1, CS2 und CS3 verwendet werden, konvertiert werden muß. Je nach verwendeter Programmiersprache ist also mehr oder weniger Sorgfalt geboten (Siehe Abschnitt 11.1).

Details zu den im Folgenden beschriebenen Unterprogrammen sind im Quellcode in Abschnitt 9.15 zu finden.

4.1 Entfernungen und Längen zwischen CS1 und CS2

Die zugehörige simple Theorie um Entfernungen und Längen von Modell nach Leinwand und zurück zu konvertieren, wurde in Abschnitt 3.2 behandelt. Realisiert wurde das in den beiden überladenen Funktionen `to_distance`. Die Funktionen unterscheiden sich in den Zahlentypen ihrer Argumente und Rückgabewerte.

[1] engl. to scroll

4.2 Umrechnung zwischen realen und virtuellen Modellkoordinaten

Achtung! Der Begriff "real" hat hier nichts mit dem Maßstab aus Abschnitt 3.1 zu tun !

Der Unterschied zwischen realen und virtuellen Modellkoordinaten kommt durch die Normierung zustande. Dazu wurden bereits in Abschnitt 3.6.3 die nötigen Überlegungen dargelegt.

Ein Beispiel anhand Abbildung 4.1 soll die zu lösende Aufgabe anschaulich machen.

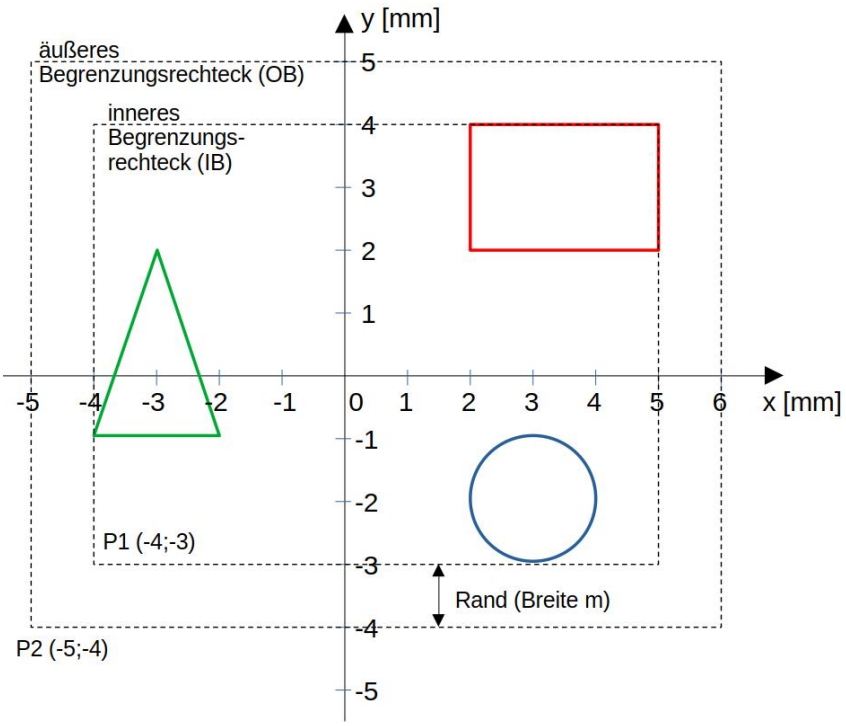

Abbildung 4.1: reale und virtuelle Modellkoordinaten

Es befindet sich das Zentrum des Kreises auf Position (3;-2). Bedingt durch die Position B_{pos} des Begrenzungsrechtecks B bei (-5;-4) wird der Kreis auf einen virtuellen Punkt P_v (8;2) verschoben. Wenn eine Abfrage der wirklichen, realen Positon vorgenommen wird, muß P_v wieder zurück verschoben werden um den Offset (-5;-4). Das Ergebnis ist dann wieder der reale Punkt P_r bei (-2;3).

Ausgehend von den Überlegungen in Abschnitt 3.6.3 sollen nun die für die Umrechnung nötigen Funktionen beschrieben werden.

4.2.1 Von real nach virtuell

Die Funktion `to_virtual` nimmt als Argument einen realen Modellpunkt P_r und liefert den entsprechenden virtuellen Modellpunkt P_v zurück:

$$\vec{P}_v = \vec{P}_r - \vec{B}_{pos} \tag{4.1}$$

4.2.2 Von virtuell nach real

Die Funktion `to_real` nimmt als Argument einen virtuellen Modellpunkt P_v und liefert den entsprechenden realen Modellpunkt P_r zurück:

$$\vec{P}_r = \vec{P}_v + \vec{B}_{pos} \tag{4.2}$$

4.3 Von Modell nach Leinwand

Es wird von CS1 nach CS2 konvertiert, also von `mm` nach `lp`. Gegeben sei ein realer Modellpunkt M_1. Gesucht ist der zugehörige Punkt C_2 auf der Leinwand.

1. M_1 muß zuerst verschoben werden um die invertierte Position des Begrenzungsrechtecks (siehe Abschnitt 3.6.3) was dann den virtuellen (oder normierten) Punkt M_2 ergibt:

$$\vec{M}_2 = \vec{M}_1 - \vec{B}_{pos} \tag{4.3}$$

   Diese Verschiebung ist in der Funktion `to_virtual` verborgen.

2. Die Werte für x und y müssen nun mit separaten Formeln unter Einbeziehung des Basis-Offsets F und des Vergrößerungsfaktors S umgerechnet werden (Siehe Abschnitt 3.7):

$$C_{1x} = M_{2x} \cdot S + F_x \tag{4.4}$$

$$C_{1y} = -(M_{2y} \cdot S + F_y) \tag{4.5}$$

   In Gleichung 4.5 ist die Umwandlung von nach oben steigenden y-Werten in CS1 zu nach unten steigenden y-Werten in CS2 verborgen.

3. Zum Schluß muß der Punkt C_1 um den Translation-Offset T verschoben werden, um den gesuchten Punkt C_2 zu erhalten:

$$\vec{C}_2 = \vec{C}_1 + \vec{T} \tag{4.6}$$

Die Gleichungen 4.4, 4.5 und 4.6 wurden in der Funktion `virtual_to_canvas` realisert. Siehe Quellcode in Abschnitt 9.15.

4.4 Von Leinwand nach Modell

Es wird von CS2 nach CS1 konvertiert, also von `lp` nach `mm`. Gegeben sei ein Leinwandpunkt C_1. Gesucht ist der zugehörige reale Punkt M_2 im Modell.

Der in Abschitt 4.3 dargelegte Ablauf wird nun rückwärts durchlaufen.

1. Zuerst wird C_1 um den Translation-Offset T zurückverschoben.

   Dann werden die Werte für x und y mit separaten Formeln umgerechnet. Dabei ist wieder der Basis-Offset F und der Vergrößerungsfaktor S einzubeziehen (Siehe Abschnitt 3.7):

   $$M_{1x} = \frac{C_{1x} - T_x - F_x}{S} \tag{4.7}$$

   $$M_{1y} = \frac{-(C_{1y} - T_y) - F_y}{S} \tag{4.8}$$

   In Gleichung 4.8 wird von nach unten steigenden y-Werten in CS2 zu nach oben steigenden y-Werten in CS1 umgewandelt.

   Die Gleichungen 4.7 und 4.8 liefern einen virtuellen Modellpunkt ausgehend von einem gegebenen Leinwandpunkt und sind in der Funktion `canvas_to_virtual` verborgen. Siehe Quellcode in Abschnitt 9.15.

2. Abschließend wird der virtuelle Punkt M_1 um die Position des Begrenzungsrechtecks zurückverschoben, um den realen Modellpunkt M_2 zu erhalten. Das entspricht der in Abschnitt 3.6.3 beschriebenen Entnormierung:

   $$\vec{M}_2 = \vec{M}_1 + \vec{B}_{pos} \tag{4.9}$$

   Dafür wird die Funktion `to_real` verwendet.

Der oben geschilderte Mechanismus wurde in der Funktion `canvas_to_real` realisert. Siehe Quellcode in Abschnitt 9.15.

Kapitel 5

Initialisierung

In diesem Kapitel soll der Zusammenhang der zentralen Größen untereinander sowie grundlegende Mechanismen dargelegt werden, die für die Initialisierung nötig sind.

Die Vorgänge im operativen Betrieb basieren auf den hier dargelegten Überlegungen und werden in Kapitel 6 behandelt.

Für den Einstieg soll zunächst der Initialisiervorgang anhand Abbildung 5.1 gezeigt werden. Sie zeigt die wichtigsten Zusammenhänge beim Systemstart.

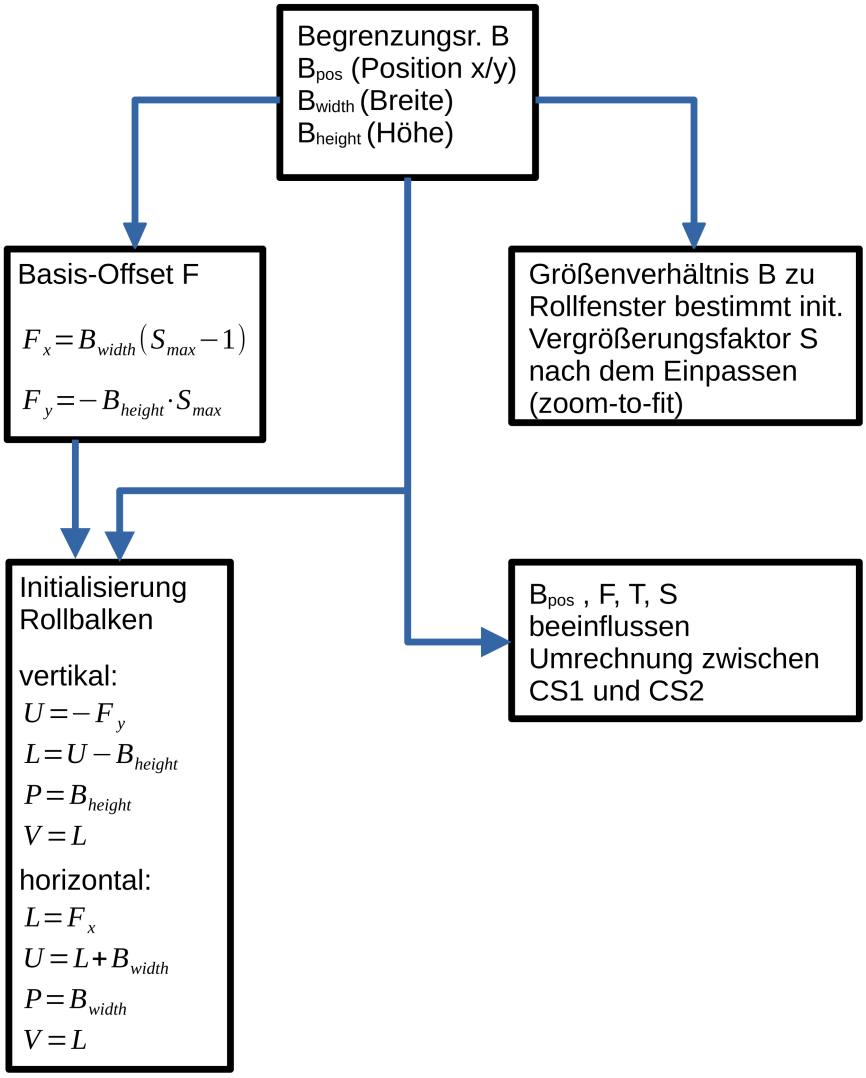

Abbildung 5.1: Zusammenhänge zentraler Größen bei Systemstart

Der Ausgangspunkt ist das Begrenzungsrechteck B (Bounding-Box). Von den Abmessungen des Begrenzungsrechtecks ist der Basis-Offset F abhängig.

Basis-Offset F und Begrenzungsrechteck B bestimmen die Initialisierung der Rollbalken.

Aus dem Größenverhältnis des Begrenzungsrechtecks zur initialen Größe des Rollfensters wird der initiale Vergrößerungsfaktor berechnet.

Außerdem sind die Position B_{pos} des Begrenzungsrechtecks, der Basis-Offset F, der Translation-Offset T und der Vergrößerungsfaktor S Grundlage für die Umrechnungen zwischen Modellkoordinaten und Leinwandkoordinaten.

In der Reihenfolge der nachfolgenden Unterabschnitte ist die Initialisierung vorzunehmen. Im Demoprogramm ist diese Initialisierung in der Hauptprozedur[1] `demo` zu finden. Siehe Quellcode in Abschnitt 9.1.

Die Initialisierung erfolgt grob in dieser Abfolge von Einzelschritten:

1. Begrenzungsrechteck B des Modells ermitteln. Siehe Abschnitt 3.6.
2. Basis-Offset F berechnen. Siehe Abschnitt 5.2.
3. Initialwerte des horizontalen und vertikalen Rollbalkens berechnen (aber noch nicht anwenden !). Siehe Abschnitt 5.3.1 und 5.3.2.
4. GTK initialisieren
5. Hauptfenster initialisieren. Siehe Abschnitt 3.8.
6. Rollfenster und Rollbalken initialisieren
7. Leinwand initialisieren. Siehe Abschnitt 5.5.
8. Leinwand in Rollfenster einfügen[2]
9. Rollfenster in Hauptfenster einfügen
10. Alle Fenster und Steuerelemente anzeigen.
11. Initialwerte des horizontalen und vertikalen Rollbalkens anwenden. Siehe Punkt 3 oben.
12. Alle Objekte in das Rollfenster einpassen[3].

[1] engl. main procedure
[2] In der GTK-Terminologie wird die Leinwand ein Kind (engl. child) des Rollfenster.
[3] engl. zoom to fit

5.1 Berechnung des Begrenzungsrechtecks

Der erste Schritt ist die Berechnung des Begrenzungsrechtecks des Modells und des Zeichnungsrahmens. Dies impliziert, daß alle betroffenden Objekte auf ihre Position und Größe untersucht werden müssen. Das ist vor dem erstmaligen Start der graphischen Oberfläche zwingend nötig.

Das Begrenzungsrechteck muß außerdem auch im Laufe des operativen Betriebes von Zeit zu Zeit neu berechnet werden. Dies ist Bestandteil des sogenannten Einpassens[4]. Gründe dafür sind:

1. Hinzufügen oder Entfernen von Objekten die sich außerhalb des gegenwärtigen Begrenzungsrechtecks befinden.

2. Verschieben von Objekten aus dem Bereich des gegenwärtigen Begrenzungsrechtecks heraus.

3. Löschen von Objekten.

4. Änderungen am Zeichnungsrahmen (z.B. von A4 nach A3), denn dieser wird wie ein Teil des Modells behandelt.

Zum Begrenzungsrechteck ist bereits in Abschnitt 3.6 einiges gesagt worden. Weil vom Begrenzungsrechteck viele Größen abhängig sind, ist seiner Berechnung einige Sorgfalt zu widmen. So muß zum Beispiel sichergestellt werden, daß die maximalen Abmessungen des Begrenzungsrechtecks nicht überschritten werden. Diese Grenzen sind die maximale Breite B_{w_max} und die maximale Höhe B_{h_max} (siehe Abschnitt 3.6.2). Was die Berechnung zeitaufwändig macht, ist der Umstand, daß **alle** komplexen Objekte des Modelles auf ihre Position und Dimensionen untersucht werden müssen. Das impliziert, daß für die primitiven Objekte eines jeden komplexen Objektes kleine Begrenzungsrechtecke gebildet und diese zu einem gesamten Begrenzungsrechteck verschmolzen werden müssen. Außerdem ist auch der Zeichnungsrahmenen und der Zeichnungsrand in die Berechnung einzubeziehen. Die Datenbank des Modells muß also bei jeder Neuberechnung des Begrenzungsrechteckes B vollständig durchforstet[5] werden. Abbildung 3.11 vermittelt einen ersten Eindruck über das Ausmaß dieses Vorganges. Dort sind nur vier komplexe Objekte skizziert. Eine echte Zeichnung kann hunderte bis tausende komplexe Objekte behinhalten.

Die Prozedur `compute_bounding_box` ist vorgesehen, um das globale Begrenzungsrechteck B zu ermitteln (Quellcode in Abschitt 9.6). Sie modifizert folgende weitere globale Variablen. Diese Variablen sind global angelegt, damit die zahlreichen anderen Unterprogramme diese lesen können:

- `bounding_box`. Das ist das Begrenzungsrechteck B an sich.

- `bounding_box_changed`. Es handelt sich um ein Flag, welches anzeigt, ob sich das Begrenzungsrechteck B geändert hat oder nicht.

- `bounding_box_error`. Darin ist die Aussage enthalten, ob die Abmessungen des Begrenzungsrechtecks, also B_{w_max} oder B_{h_max} überschritten wurden. Weiterhin ist dort in Fehlerfall die festgestellte Breite und Höhe vermerkt.

Die Prozedur kann durch optionale Argumente diese weiteren Aufgaben erfüllen:

[4] engl. zoom to fit
[5] engl. to parse

1. **Abbruch bei erstem Fehler**: Normalerweise werden alle Objekte der Datenbank in die Berechnung des Begrenzungsrechtecks einbezogen und erst am Schluß die Abmessungen geprüft. In diesem Fall wird schon bei der ersten Überschreitung des Grenzwertes für Breite B_{w_max} oder Höhe B_{h_max} abgebrochen. **Im gegemwärtigen Stand des Demoprogrammes ist diese Funktion noch nicht implementiert.**

2. **Ignorieren von Fehlern**: Das bedeutet, im Fehlerfall wird ein Begrenzungsrechteck berechnent, daß die zulässigen Abmessungen überschreiten darf. Diese Option ist interessant um den Fall zu simulieren, bei dem die Abmessungen der Leinwand nicht ausreichen (siehe Abbildung5.12).

3. **Reiner Test**: Test, ob das Begrenzungsrechteck fehlerfrei berechnent werden kann. Das Begrenzungsrechteck, also die globale Variable `bounding_box`, wird in diesem Fall **nicht** verändert. Das Flag `bounding_box_changed` wird zurückgesetzt. Die globale Variable `bounding_box_error` wird im Fehlerfall entsprechend modifizert.

5.1.1 Details zur Berechnung

Die Bestimmung des Begrenzungsrechtecks B mittels der Prozedur `compute_bounding_box` soll nun näher beschrieben werden. Jedoch können nicht alle Feinheiten der Prozedur hier behandelt werden. Darum sei auf den Quellcode in Abschitt 9.6 verwiesen.

Hier eine Übersicht der auszuführenen Teilaufgaben:

1. Zeichnungsrahmen (ZR) durchforsten. Das Dokumentationsfeld kann übersprungen werden, weil es gesichert innerhalb des Zeichnungsrahmens liegt.
2. Datenbank der komplexen Objekte durchforsten.
3. Primitive Objekte innerhalb der komplexen Objekte durchforsten.
4. Für jedes primitive Objekt ein separates Begrenzungsrechteck G berechnen. Die Linienbreite des primitiven Objektes geht in das Begrenzungsrechteck G ein. Dabei ist die Position des übergeordneten komplexen Objektes sowie dessen Rotation und evtl. Spiegelung zu berücksichtigen.
5. Einzelne Begrenzungsrechtecke der primitiven Objekte (G) miteinander zu einem großen Begrenzungsrechteck H verschmelzen [6].
6. Begrenzungsrechteck H um die Breite m des Zeichnungsrandes ausdehnen und um seine Breite m nach links und nach unten verschieben.

Zu Beginn der Prozedur wird ein temporäres lokales Begrenzungsrechteck `bbox_new` auf Position (0;0) mit null Breite und null Höhe angelegt. Im weitern Verlauf werden alle aus primitiven Objekten entstehenden Begrenzungsrechtecke mit `bbox_new` verschmolzen. Das temporäre Begrenzungsrechteck `bbox_new` wird also im Laufe der Prozedur größer und größer. Am Schluß der Prozedur enthält `bbox_new` das Begrenzungsrechteck aller darzustellenden Objekte.

[6] engl. to merge

5.1.1.1 Primitive Objekte

In Abschnitt 3.3.1 wurde bereits erklärt, was ein primitives Objekt ist.

Da diese einfachen Strukturen mit einer bestimmten Linienbreite gezeichnet werden, ist auch die Linienbreite in die Berechnung einzubeziehen. Die Berechnung des Begrenzungsrechteckes für Linien und Kreise gestaltet sich einfach. Aufwändiger wird es bei Kreisbögen. Da es aber hier nur um ein überschlägiges Begrenzungsrechteck geht, könnte man einen Kreisbogen wie einen Kreis betrachten.

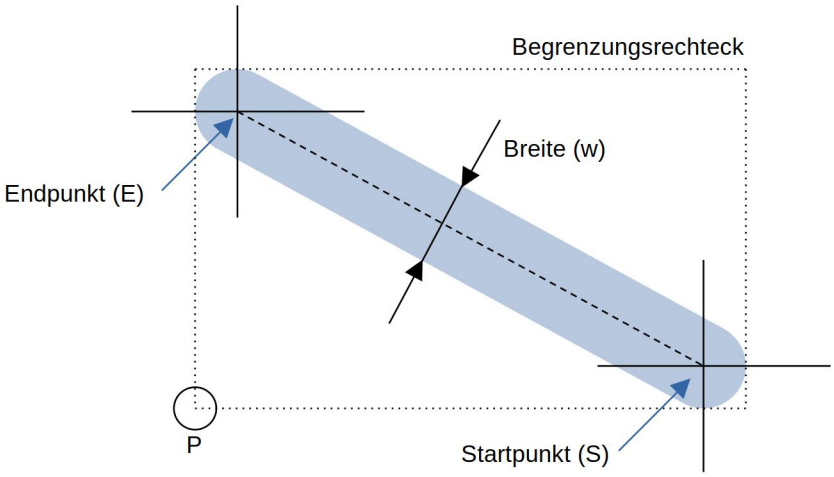

Abbildung 5.2: Begrenzungsrechteck einer Linie

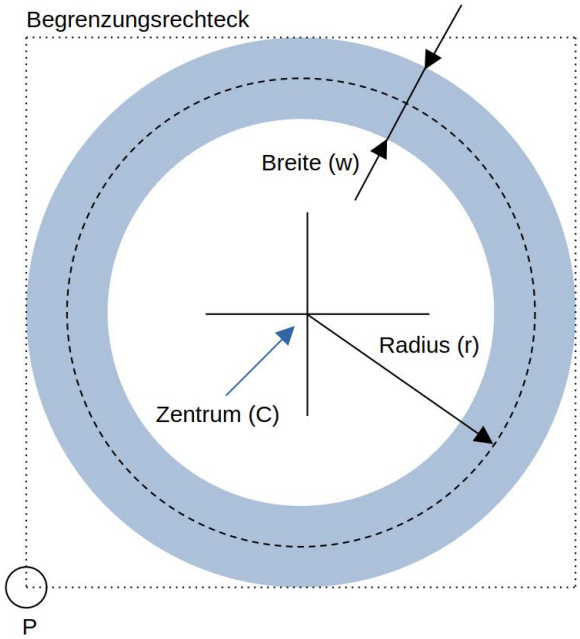

Abbildung 5.3: Begrenzungsrechteck eines Kreises

Im Demoprogramm werden zwei überladene Funktion `get_bounding_box` verwendet, um das Begrenzungsrechteck einer Linie und eines Kreises zu erhalten. Siehe Quellcode in Abschitt 9.3.

5.1.1.2 Verschmelzen von Begrenzungsrechtecken

Die Begrenzungsrechtecke der im Demoprogramm verwendeten Kategorien primitiver Objekte sind in Abbildung 5.2 und 5.3 abstrahiert zu sehen. Aus diesen primitiven Objekten ist der Zeichnungsrahmen (ZR), das Dokumentationsfeld (DF) und ein jedes komplexes Objekte zusammengesetzt.

Im Laufe der Berechnung des Begrenzungsrechteckes B für die gesamte Zeichnung müssen ständig die einzelnen kleineren Begrenzungsrechtecke der primitiven Objekte miteinander verschmolzen werden.

Wenn die Begrenzungsrechtecke zweier Objekte miteinander verschmolzen werden, sieht das Ergebnis so aus wie in Abbildung 5.4. Das Gesamt-Begrenzungsrechteck ist mit einer Strichellinie eingezeichnet.

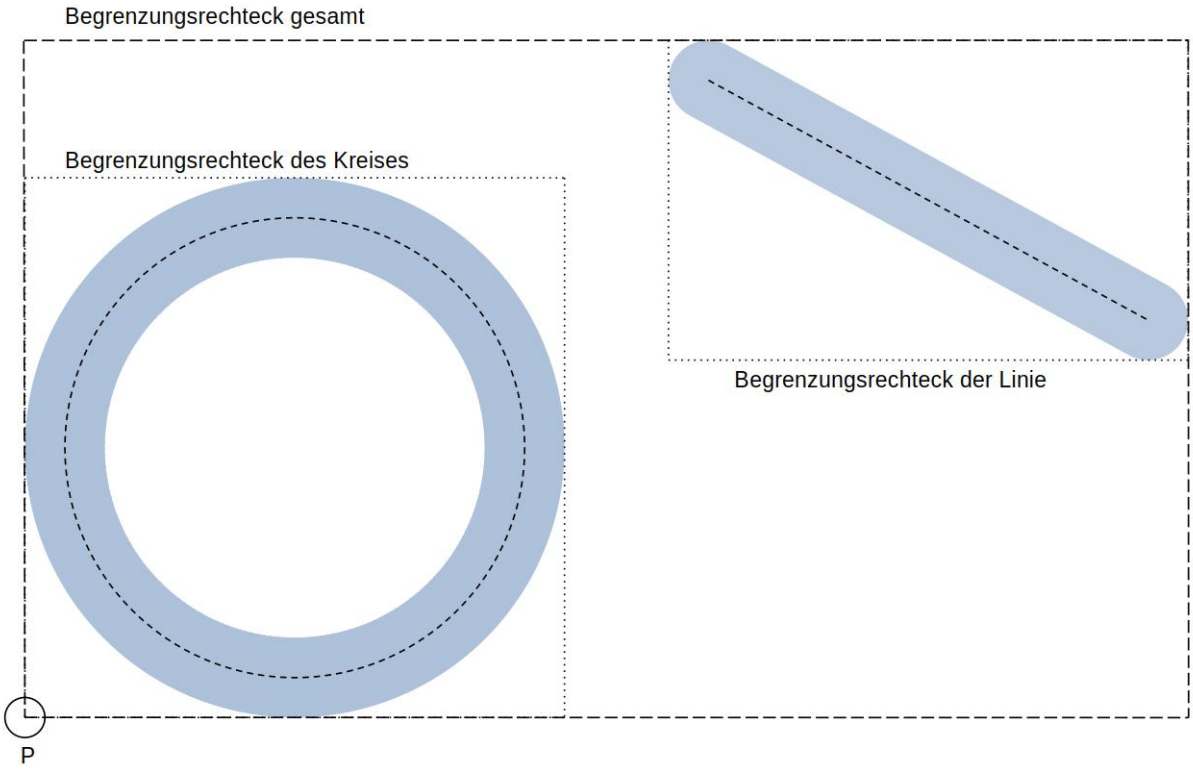

Abbildung 5.4: Verschmelzen zweier Begrenzungsrechtecke

Für das Verschmelzen zweier Begrenzungsrechtecke ist die Prozedur `merge_areas` vorgesehen. Siehe Quellcode in Abschnitt 9.18.

5.1.1.3 Zeichnungsrahmen

Die Prozessierung des ZR besteht aus einem sogenannten Iterator, welcher die in der Variablen `drawing_frame` gespeicherten Linien - eine nach der anderen - aufgreift. Für jede Linie wird zunächst ein eigenes Begrenzungsrechteck G unter Einbeziehung der Linienbreite ermittelt. Dieses wird danach um die Position des ZR verschoben. In Abschnitt 3.4.1 wurden die Gründe für diesen Mechanismus bereits dargelegt. Das Begrenzungsrechteck G wird abschließend mit dem temporären Begrenzungsrechteck `bbox_new` verschmolzen. Die primitiven Objekte des Dokumentationsfeldes (DF) brauchen nicht behandelt werden, weil sich das DF gesichert innerhalb des ZR befindet.

5.1.1.4 Komplexe Objekte

Im Folgenden ist zu beachten, daß in Bibliotheken die komplexen Objekte in ihrer **Urform** modelliert sind (Siehe Abschnitt 3.3.2.1). Es gelten also Standardwerte (oder Vorgaben) für:

- x/y-Position
- Drehwinkel
- Spiegelung (optional)

Wenn ein komplexes Objekt in der Zeichnung platziert[7] wird, gelten diese Standardwerte nicht mehr, denn der Bediener bestimmt zu diesem Zeitpunkt die eigentlichen Position, Drehwinkel und evtl. die Spiegelung. Diese Daten müssen bei der Bestimmung des Begrenzungsrechteckes der primitiven Objekte einbezogen werden[8].

Die komplexen Objekte sind in einer Datenbank gespeichert. Im Demoprogramm ist das die globale Variable `objects_database` (Siehe Quellcode in Abschnitt 9.3). Ein Iterator greift ein Objekt nach dem anderen auf und übergibt es an eine Prozedur, welche das einzelne Objekt behandelt. Diese Prozedur ist entsprechend benannt als `query_object` und kümmert sich nun um die primitiven Objekte des gegebenen komplexen Objetkts. In dieser Prozedur durchforstet ein Iterator alle Linien, ein zweiter alle Kreise und eventuell ein möglicher dritter alle Kreisbögen. Alle Iteratoren übergeben das jeweilige primitive Objekt an ein spezielles Unterprogramm, welches das Objekt weiterbehandelt. Im Demoprogramm sind das die Prozeduren `query_line` und `query_circle`.

Die vollständige Behandlung des primitiven Objektes sollte nach diesem Schema erfolgen:

1. Eine Kopie C des Objektes erzeugen.
2. Falls mit Spiegelung komplexer Objekte gearbeitet wird, kann C hier gespiegelt werden. Je nach Anwendung kann auch später gespiegelt werden (siehe unten).
3. C um den Drehwinkel des übergeordneten komplexen Objektes drehen. Das Zentrum der Drehung ist der **Ursprung** des komplexen Objektes.
4. Je nach Anwendung kann C jetzt gespiegelt werden (siehe oben).
5. C um die Position des übergeordneten komplexen Objektes verschieben.
6. Für C das Begrenzungsrechteck G ermitteln (Linienbreite beachten !) und mit `bbox_new` verschmelzen.

[7] Programmierer verwenden in diesem Zusammenhang das Wort "Instanzierung".
[8] Auch beim Zeichnen auf der Leinwand ist dieser Umstand zu bedenken (Siehe Abschnitt 6.6.9.2)

Im Demoprogramm wurde jedoch ein vereinfachtes Verfahren angewendet, weil darin nicht mit Drehwinkeln und Spiegelung gearbeitet wird:

1. Begrenzungsrechteck G des primitiven Objektes ermitteln (Linienbreite beachten !).
2. G um die Position des übergeordneten komplexen Objektes verschieben.
3. G mit `bbox_new` verschmelzen.

5.1.1.5 Zeichnungsrand

Um den Zeichnungsrand der Breite m wird das temporäre Begrenzungsrechteck `bbox_new` ausgedehnt. In Abschnitt 3.6 wurden dazu bereits die nötigen Grundgedanken formuliert.

Die Breite und Höhe des temporären Begrenzungsrechtecks `bbox_new` wird um das Doppelte von m vergrößert. Dann wird `bbox_new` um m nach links-unten verschoben.

Nun ist die Berechnung des Begrenzungsrechtecks B abgeschlossen.

5.2 Berechnung des Basis-Offsets

Der Basis-Offset F ist eine globale Variable und wird ausgehend von der Breite und Höhe des gegenwärtigen Begrenzungsrechteckes B und dem maximalen Vergrößerungsfaktor S_{max} errechnet. Wenn die Abmessungen des Begrenzungsrechtecks sich ändern, z.B. durch Einpassen, dann muß der Basis-Offset ebenso neu berechnet werden. Siehe Abschnitt 5.1. Der Basis-Offset muß so bemessen sein, daß nach oben und nach links ausreichend Raum auf der Leinwand besteht. Dieser Raum ist nötig, um auch bei maximalem Vergrößerungsfaktor alle Objekte auf der Leinwand unterzubringen.

Merke: Beim Vergrößern wird mehr Leinwandfläche gebraucht. Oberhalb der X-Achse und links der Y-Achse gibt es keine Leinwand ! Die Berechnung erfolgt separat für die Y- und Y-Achse.

Im Demoprogramm ist für die Berechnung des Basis-Offsets F die Prozedur `set_base_offset` vorgesehen. Siehe Quellcode in Abschnitt 9.7 für Details.

5.2.1 Y-Achse

Der y-Anteil des Basis-Offsets ist von Bedeutung um genug Raum nach **oben** bei maximaler Vergrößerung zu erhalten wenn das Zentrum einer Vergrößerung im Punkt F liegt. Abbildung 5.5 soll die Platzverhältnisse auf der Leinwand veranschaulichen.

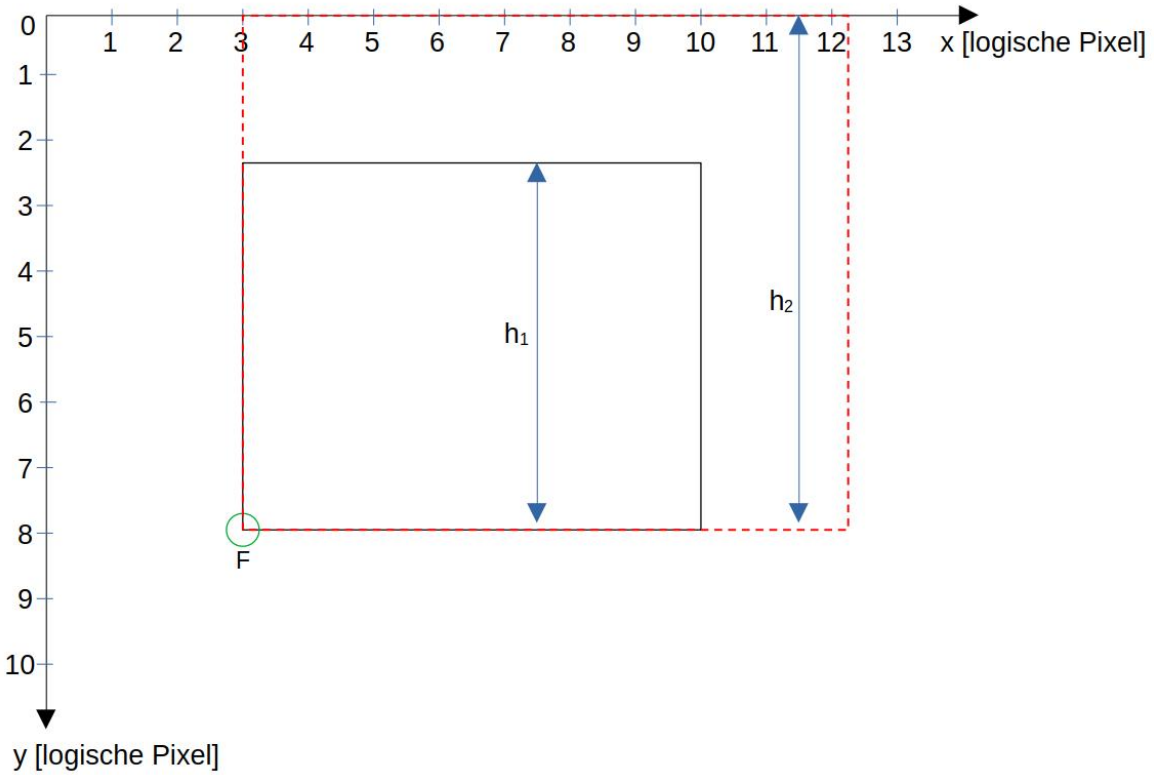

Abbildung 5.5: Basis-Offset Berechnung y-Anteil

h_1 entspricht der Höhe B_h des Begrenzungsrechtecks bei Vergrößerungsfaktor 1,0. Die zugehörige Fläche ist mit durchgezogener Linie dargestellt.

h_2 steht für die Höhe des Begrenzungsrechtecks bei maximaler Vergrößerung. Bei maximaler Vergrößerung mit Zentrum F dehnt sich das Bild nach rechts-oben aus. Die resultierende dafür benötigte Fläche ist gestrichelt eingezeichnet.

$$h_2 = h_1 \cdot S_{max} \tag{5.1}$$

Gleichung (4.5) muß nach F_y umgestellt werden:

$$F_y = -C_y - M_y \cdot S \tag{5.2}$$

Für M ist die Höhe des Begrenzungsrechtecks und für S die maximale Vergrößerung einzusetzen:

$$F_y = -C_y - B_h \cdot S_{max} \tag{5.3}$$

Bei maximaler Vergrößerung mit Zentrum F erreicht C_y den oberen Rand der Leinwand, nimmt den Wert Null an und kann somit weggelassen werden. Der y-Anteil des Basis-Offsets wird also schlußendlich berechnet nach:

$$F_y = -B_h \cdot S_{max} \tag{5.4}$$

Daraus ergibt sich, daß die y-Komponente des Basis-Offsets eine negative Zahl ist.

5.2.2 X-Achse

Der x-Anteil des Basis-Offsets schafft genug Raum nach **links** bei maximaler Vergrößerung wenn das Zentrum im Punkt Z liegt. Zum Verständis ist Abbildung 5.6 gedacht.

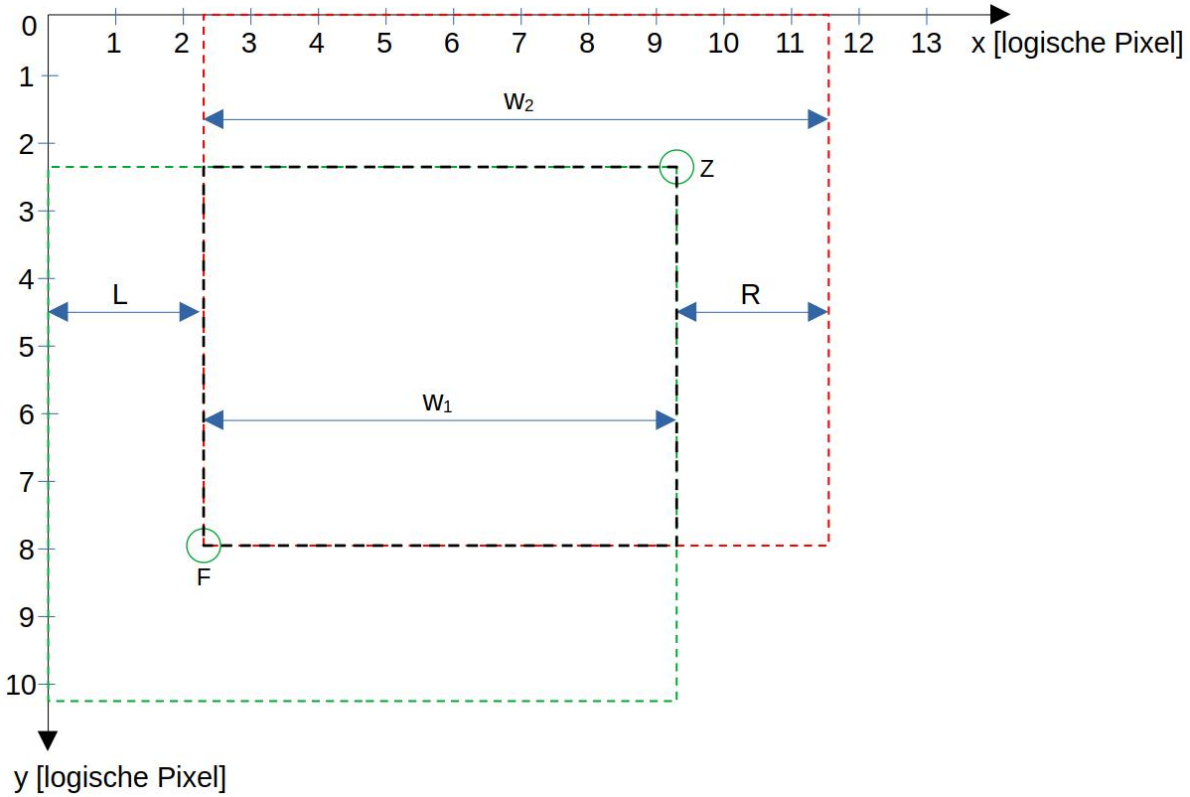

Abbildung 5.6: Basis-Offset Berechnung x-Achse

W_1 entspricht der Breite des Begrenzungsrechtecks (auf der Leinwand) welches in der Mitte der Abbildung zu sehen ist.

Bei maximaler Vergrößerung mit Zentrum F dehnt sich das Bild nach rechts-oben. Der resultierende Bereich ist gestrichelt eingetragen. Es gilt:

$$W_2 = W_1 \cdot S_{max} \tag{5.5}$$

Rechts ist also eine zusätzliche Leinwandbreite R vorzusehen:

$$R = W_2 - W_1 \tag{5.6}$$

Die selbe Breite L ist nun auch links gefordert. Wenn das Zentrum der Vergrößerung im Punkt Z ist, dehnt sich das Bild nach links-unten aus. Der sich ergebende Bereich ist mit Strichellinie gezeichnet. Somit ergibt sich:

$$F_x = R \tag{5.7}$$

Zusammenfassend gilt also:

$$F_x = B_w \cdot S_{max} - B_w \tag{5.8}$$

Die Vereinfachung der Gleichung liefert schlußendlich für die x-Komponente des Basis-Offsets:

$$F_x = B_w(S_{max} - 1) \tag{5.9}$$

5.3 Rollbalken

Aus Abbildung 5.1 geht hervor, daß die Initialisierung der Rollbalken auf dem Begrenzungsrechteck B und dem Basis-Offset F beruht. Zur Erinnerung sei an dieser Stelle nochmal das Schema der Rollbalken in Abbildung 5.7 gezeigt.

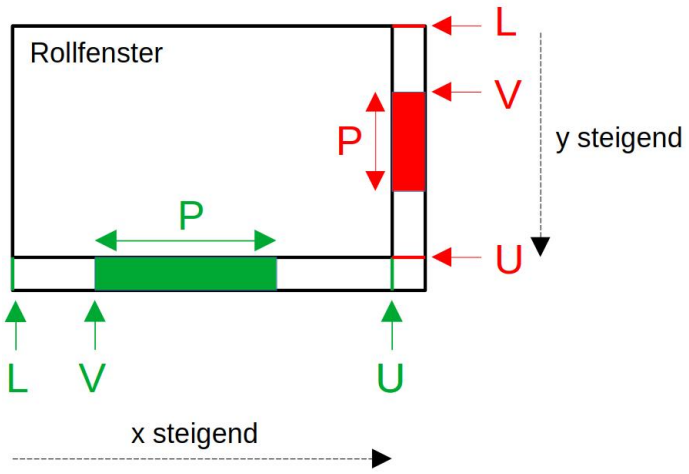

Abbildung 5.7: Rollbalken

Die praktische Umsetzung der Überlegungen in den folgenden Unterabschnitten ist in der Prozedur `set_initial_scrollbar_settings` zu finden. Siehe Quellcode in Abschnitt 9.10.

5.3.1 Vertikaler Rollbalken

Für das Verständis der nachfolgenden Überlegungen soll uns Abbildung 5.8 eine Hilfe sein.

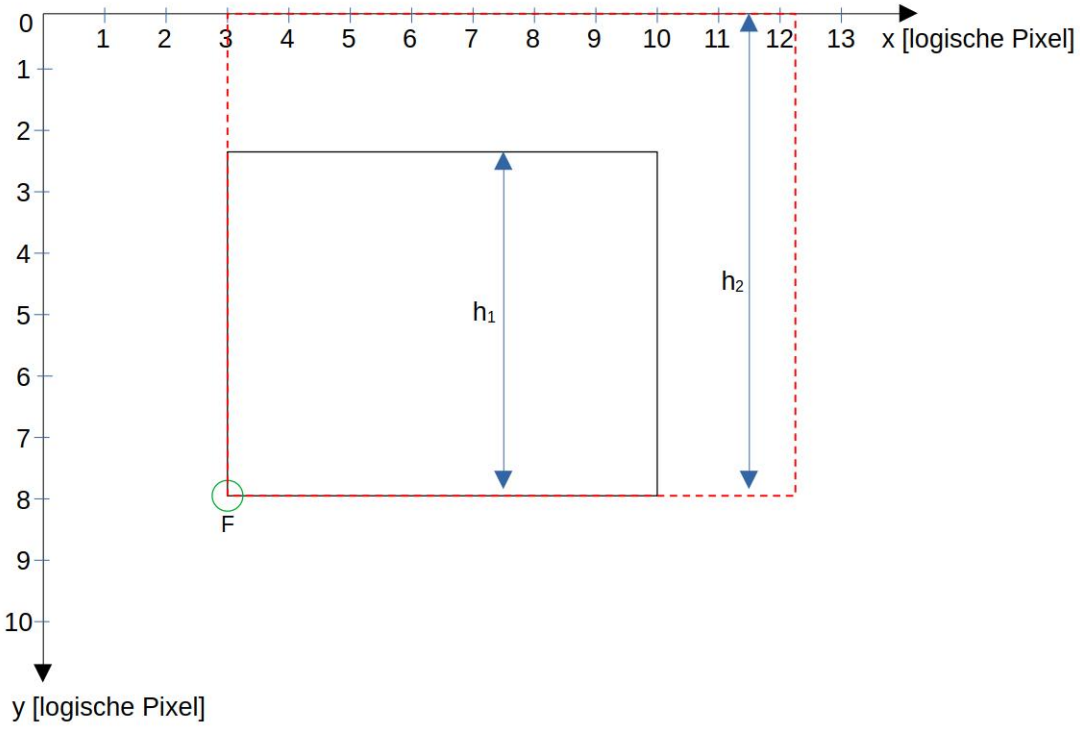

Abbildung 5.8: Initialisierung des vertikalen Rollbalkens

5.3.1.1 Oberer Anschlag

Achtung: Der obere Anschlag U befindet sich am **unteren** Rand des Rollfensters. Wenn der Inhalt des Begrenzungsrechtecks gezeichnet wird, entspricht seine untere linke Ecke auf der Leinwand dem Punkt F. Somit ist der obere Anschlag U des Rollbalkens:

$$U = -F_y \tag{5.10}$$

Weil F_y eine negative Zahl ist, muß mit -1 multipliziert werden um einen positiven Wert für U zu erhalten. Bemerkung: Es handelt sich hier nur um einen Initialwert. Im Laufe des weiteren Betriebes kann U auch kleinere Werte annehmen !

5.3.1.2 Unterer Anschlag

Achtung: Der untere Anschlag L befindet sich am **oberen** Rand des Rollfensters. h_1 entspricht der Höhe B_h des Begrenzungsrechtecks bei einem Vergrößerungsfaktor S von 1,0. Es gilt also:

$$L = U - h_1 \tag{5.11}$$

oder

$$L = U - B_{height} \cdot S \tag{5.12}$$

Bemerkung: Es handelt sich hier nur um einen Initialwert. Im Laufe des weiteren Betriebes kann L auch größere Werte annehmen !

5.3.1.3 Seitenbreite

Die Seitenbreite P, also die Höhe des sichtbaren Bereiches, entspricht der Höhe des Begrenzungsrechtecks bei Vergrößerungsfaktor S von 1,0:

$$P = h_1 \tag{5.13}$$

oder

$$P = B_{height} \cdot S \tag{5.14}$$

5.3.1.4 Einstellwert

Die Position oder Einstellung V entspricht dem Wert des unteren Anschlages:

$$V = L \tag{5.15}$$

In vertikaler Richtung - von oben nach unten - sehen wir von der Leinwand den Bereich von V bis V+P, also die gesamte Höhe des Begrenzungsrechtecks.

5.3.2 Horizontaler Rollbalken

Die Initialisierung des horizontalen Rollbalkens verläuft analog der des vertikalen Rollbalkens. Zum Verständis der nachfolgenden Überlegungen dient nun Abbildung 5.9.

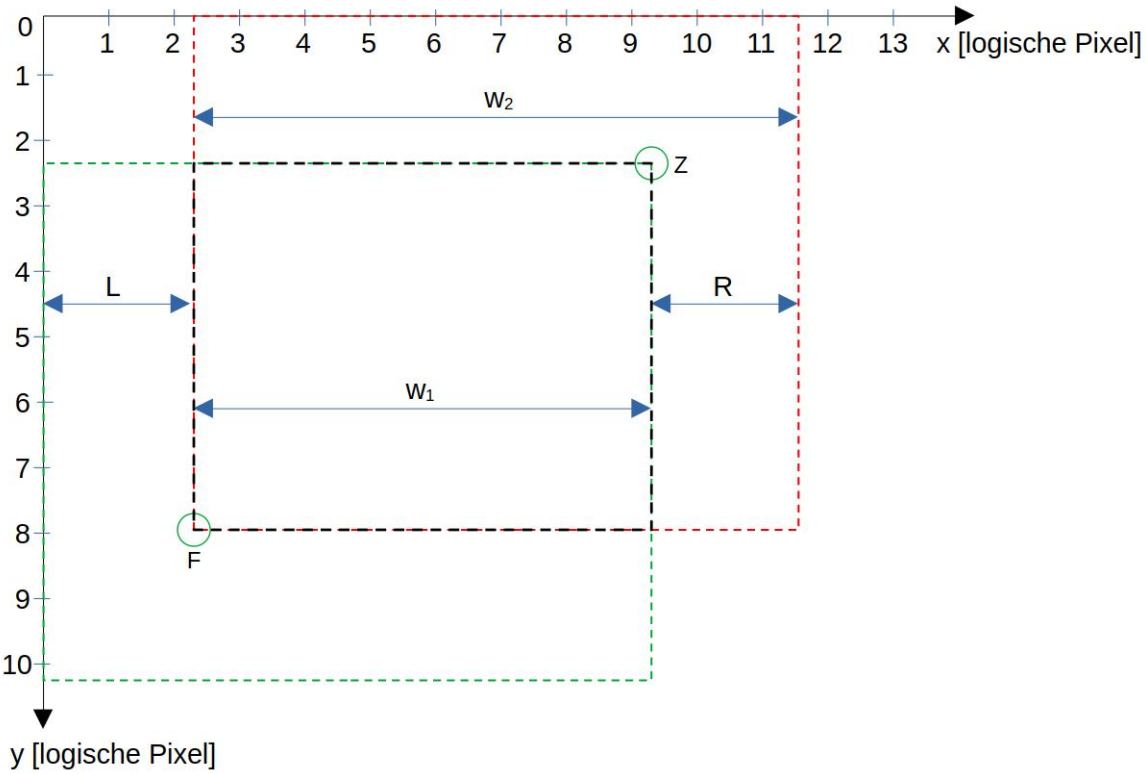

Abbildung 5.9: Initialisierung horizontaler Rollbalken

5.3.2.1 Linker Anschlag

Links im Rollfenster befindet sich der untere Anschlag L. Wenn der Inhalt des Begrenzungsrechtecks gezeichnet wird, befindet sich seine untere linke Ecke auf der Leinwand bei Punkt F. Somit ist der untere Anschlag L des horizontalen Rollbalkens:

$$L = F_x \tag{5.16}$$

Bemerkung: Es handelt sich hier nur um einen Initialwert. Im Laufe des weiteren Betriebes kann L auch größere Werte annehmen !

5.3.2.2 Rechter Anschlag

Rechts im Fenster befindet sich der obere Anschlag U. w_1 steht für die Breite des Begrenzungsrechtecks B_w bei einem Vergrößerungsfaktor S von 1,0. Der obere Anschlag befindet sich um w_1 weiter rechts von L:

$$U = L + w_1 \tag{5.17}$$

oder

$$U = L + B_{width} \cdot S \tag{5.18}$$

Bemerkung: Es handelt sich hier nur um einen Initialwert. Im Laufe des weiteren Betriebes kann U auch kleinere Werte annehmen !

5.3.2.3 Seitenbreite

Die Seitenbreite, also die Weite des sichtbaren Bereiches, entspricht der Breite des Begrenzungsrechtecks bei einem Vergrößerungsfaktor S von 1,0:

$$P = w_1 \tag{5.19}$$

oder

$$P = B_{width} \cdot S \tag{5.20}$$

5.3.2.4 Einstellwert

Die Position oder Einstellung V entspricht dem Wert des unteren Anschlages:

$$V = L \tag{5.21}$$

In horizontaler Richtung - von links nach rechts - sehen wir von der Leinwand den Bereich von V bis V+P, also die gesamte Breite des Begrenzungsrechtecks.

5.4 Objekte einpassen (Zoom-to-Fit)

Dieser Vorgang bedeutet, daß alle Objekte und der Zeichnungsrahmen im Rollfenster dargestellt werden um einen Gesamtüberblick zu bekommen. Beim Start des CAD-Systems wird das Begrenzungsrechteck erstmalig berechnet. Dieser Vorgang basiert auf der Position der Objekte (im Modell) so wie sie in der Projektdatenbank eingetragen sind. Er basiert ebenso auf dem Zeichnungsrahmen. Auf diese Weise ist in der erstmaligen Ansicht jedes Objekt und der gesamte Zeichnungsrahmen dem Bediener sichtbar.

Der Vorgang des Einpassen wird in zwei Situationen ausgelöst:

1. Beim Start des Systems, also initial, müssen alle Objekte samt Zeichnungsrahmen angezeigt werden.
2. Im operativen Betrieb verschiebt der Bediener laufend Objekte, löscht welche oder fügt andere hinzu. Um von Zeit zu Zeit zwischendurch einen Gesamtüberblick zu erhalten löst der Bediener den Vorgang aus. Das ist aber nicht Bestandteil der Initialisierung sondern des operativen Modus, weshalb hier auf Abschnitt 6.1 verwiesen sei.

Diese Vorgang beinhaltet zwei Kernaufgaben:

- Ermittlung eines Vergrößerungsfaktors welcher eine gegebene Fläche so verkleinert oder vergrößert, daß sie in das Rollfenster paßt
- Zentrierung einer gegebenen Fläche im Rollfenster

Ablauf

1. Aus den vorangegangenen Schritten ist das Begrenzungsrechteck B und der Basis-Offset F bekannt. Der Translation-Offset T ist initial gleich (0;0).
2. Da das Rollfenster eine vorgegebene Initialgröße hat, muß ein Vergrößerungsfaktor gefunden werden, bei dem alle Objekte und der geamte Zeichnungsrahmen durch das Rollfenster sichtbar sind. Wir haben es in diesem Fall mit zwei rechteckigen Flächen zu tun: Die Fläche des Rollfensters und die des Begrenzungsrechteckes B. Mittels der Funktion `get_ratio` wird das Größenverhältnis beider Flächen ermittelt und der Vergrößerungsfaktor S entsprechend gesetzt. Dieser Vorgang wird in Abschnitt 5.4.1 genauer behandelt.
3. Nun muß der Translation-Offset T so gesetzt werden, daß der Inhalt des Begrenzungsrechteckes B zentriert im Rollfenster erscheint. Das wird mit der Prozedur `center_to_visible_area` bewerkstelligt. In Abschnitt 5.4.2 sind dazu die Details zu finden.

In der Prozedur `zoom_to_fit` wurde dieser Mechanismus umgesetzt. Siehe Quellcode in Abschnitt 9.12.

5.4.1 Größenverhältnis Rollfenster zu einer rechteckigen Fläche

Im Zusammenhang mit dem Einpassen aller Objekte oder dem Vergrößern eines beliebigen Bereiches ist es nötig, Größenverhältnisse von rechteckigen Flächen zueinander zu ermitteln. Daraus soll dann der nötige Vergrößerungsfaktor S ermittelt werden, um die gegebene Fläche in das Rollfenster einzupassen. Diese Situationen sind betroffen:

- Start des Systems. Die gesamte Zeichnung - also das Begrenzungsrechteck B - muß in das in das Rollfenster eingepaßt werden.
- Vergrößerung eines ausgewählten Bereiches und Einpassen dessen in das Rollfenster. Dieser Vorgang wird im operativen Modus gestartet. Dazu sei auf Abschnitt 6.2 verwiesen.

Die Fläche des Rollfensters bezeichnen wir mit D. Die in D einzupassende Fläche nennen wir E. Es stellt sich nun die Frage, ob anhand des Breiten- oder Höhenverhältnisses zwischen D und E skaliert werden soll. Dazu bestimmen wir zunächst das Verhältnis der Breiten und Höhen separat voneinander:

$$S_w = \frac{D_{width}}{E_{width}} \tag{5.22}$$

Ist D breiter als E, wird S_w größer 1,0.

$$S_h = \frac{D_{height}}{E_{height}} \tag{5.23}$$

Ist D höher als E, wird S_h größer 1,0.

Der kleinere Wert von S_w und S_h bestimmt am Ende den neuen globalen Vergrößerungsfaktor S:

$$S = min(S_w, S_h) \tag{5.24}$$

In der Funktion `get_ratio` wurde dieser Mechanismus umgesetzt. Ihr Argument ist eine Fläche, die in das gegenwärtige Rollfenster eingepaßt werden soll. Das Ergebnis ist der dafür nötige Vergrößerungsfaktor S. Siehe Quellcode in Abschnitt 9.10 für Details.

5.4.2 Fläche über sichtbarem Bereich zentrieren

Dieser Vorgang ist ebenfalls für das Einpassen aller Objekte oder das Vergrößern eines beliebigen Bereiches nötig. Eine gegebene rechteckige Fläche C soll so verschoben werden, daß die Koordinaten ihrer geometrische Mitte mit der des sichtbaren Bereiches übereinstimmen. Der für die Verschiebung benötigte Translation-Offset T soll berechnet werden. Die Forderung ist, daß dabei die Rollbalken **unverändert** bleiben sollen.

Abbildung 5.10 zeigt die Ausgangslage. Das gewünschte Ergebnis ist in Abbildung 5.11 zu sehen. Die gegebene Fläche C hat die Breite w_2, die Höhe h_2 und die Ausgangsposition bei $(x_0;y_0)$.

Der sichtbare Bereich V hat die Breite w_1 und die Höhe h_1. Ebenso hat V eine Position, die untere linke Ecke, mit den Koordinaten $(x_1;y_1)$. Zur Ermittlung des sichtbaren Bereiches siehe Abschnitt 6.6.1.

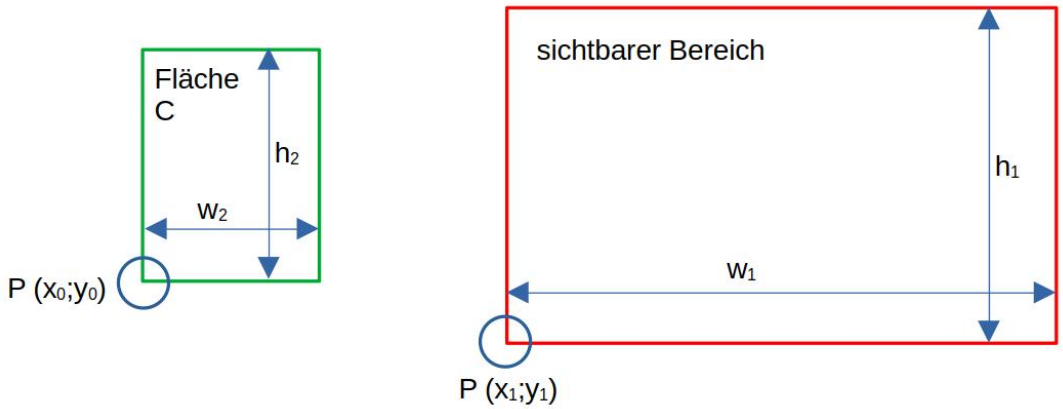

Abbildung 5.10: Fläche zentrieren, vorher

Das Ziel ist, Fläche C innerhalb des sichtbaren Bereiches zu zentrieren, wie in Abbildung 5.11 zu sehen ist. Es ist also der Punkt P bei $(x_2;y_2)$ gesucht.

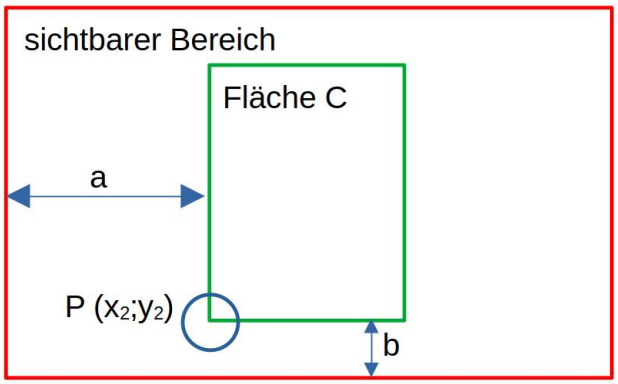

Abbildung 5.11: Fläche zentrieren, nachher

Die Zielposition der Fläche C ist bei (x_2;y_2). Links von C gibt es den Abstand a. Unterhalb von C besteht der Abstand b.

Für die horizontale Verschiebung gilt:

$$a = \frac{w_1 - w_2}{2} \tag{5.25}$$

$$x_2 = x_1 + a \tag{5.26}$$

Die Verschiebung im CS1 ist somit:

$$dx = x_2 - x_0 \tag{5.27}$$

Um die zugehörige x-Verschiebung - im CS2 - zu erhalten, muß mit dem gegenwärtigen Vergrößerungsfaktor multipliziert werden. Für die x-Komponente des Translation-Offsets T gilt:

$$T_x = dx \cdot S \tag{5.28}$$

Für die vertikale Verschiebung der Fläche C gilt:

$$b = \frac{h_1 - h_2}{2} \tag{5.29}$$

$$y_2 = y_1 + b \tag{5.30}$$

Die y-Verschiebung in CS1 ist somit:

$$dy = y_2 - y_0 \tag{5.31}$$

Um die zugehörige y-Verschiebung - in CS2 - zu erhalten, muß mit dem gegenwärtigen Vergrößerungsfaktor multipliziert werden. Zu beachten ist, daß auf der Leinwand die y-Werte nach unten hin steigen. Darum muß dy mit -1 multipliziert werden. Für die y-Komponente des Translation-Offsets T gilt also:

$$T_y = -dy \cdot S \tag{5.32}$$

Der beschriebene Ablauf funktioniert auch dann, wenn die Fläche C breiter oder höher ist als der sichtbare Bereich.

Die Prozedur *center_to_visible_area* erfüllt die Aufgabe, den nötigen Translation-Offset T so zu berechnen, daß eine gegebene Fläche im sichtbaren Bereich zentriert wird. Siehe Quellcode in Abschnitt 9.13.

5.5 Abmessungen der Leinwand

Grundsätzlich ist die Leinwand ist so zu bemessen, daß bei maximaler Vergrößerung und maximaler Größe des Rollfensters alle Objekte einschließlich des Zeichnungsrahmens darauf Platz finden. Die Abmessungen der Leinwand sind abhängig von:

1. dem maximalen Vergrößerungsfaktor S_{max}
2. der Größe des Begrenzungsrechtecks B

Eine Möglichkeit: Die Abmessungen der Leinwand werden bei Systemstart einmalig vorgegeben. Das ist die **statische** Konfiguration, das heißt, die Leinwandabmsssungen ändern sich im Laufe des weiteren Betriebes nicht. Bei diesem Ansatz gibt es wiederum zwei Möglichkeiten, die im folgenden Abschnitt 5.5.1 beschrieben werden.

Möglichkeit 2: Eleganter wäre jedoch eine **dynamische** Anpassung der Abmessungen sobald sich das Begrenzungsrechteck B verändert, was beim Einpassen gemacht wird. Die Überlegungen dazu sind in Abschnitt 5.5.2 festgehalten worden. Bisher haben diese aber noch nicht zum Erfolg geführt, sollen aber der Vollständigkeit halber dokumentiert werden.

5.5.1 Statische Konfiguration

Die hier vorgestellten zwei Lösungen basieren auf statischen Abmessungen der Leinwand. Das heißt, der Leinwand werden einmalig Breite und Höhe zugewiesen. Im Laufe des Betriebes werden die Abmessungen **nicht** geändert.

5.5.1.1 Die empirische Lösung

Diese Methode ist am einfachsten realisierbar. Es gibt **keine Abhängigkeit** vom Begrenzungsrechteck B oder anderen Systemvariablen. Die Abmessungen der Leinwand sind möglichst großzügig zu bemessen, um auch bei sehr großen Monitoren das gesamte Rollfenster mit Leinwand auszufüllen. Wenn der Bediener das Hauptfenster vergrößert (z.B. durch Maximieren), vergrößert sich indirekt auch das Rollfenster. Es wird also mehr Leinwand freigelegt. Abbildung 5.12 zeigt, was passiert, wenn die Leinwand zu knapp bemessen wurde. Unten und rechts ist der entstehende leere Bereich[9] sichtbar.

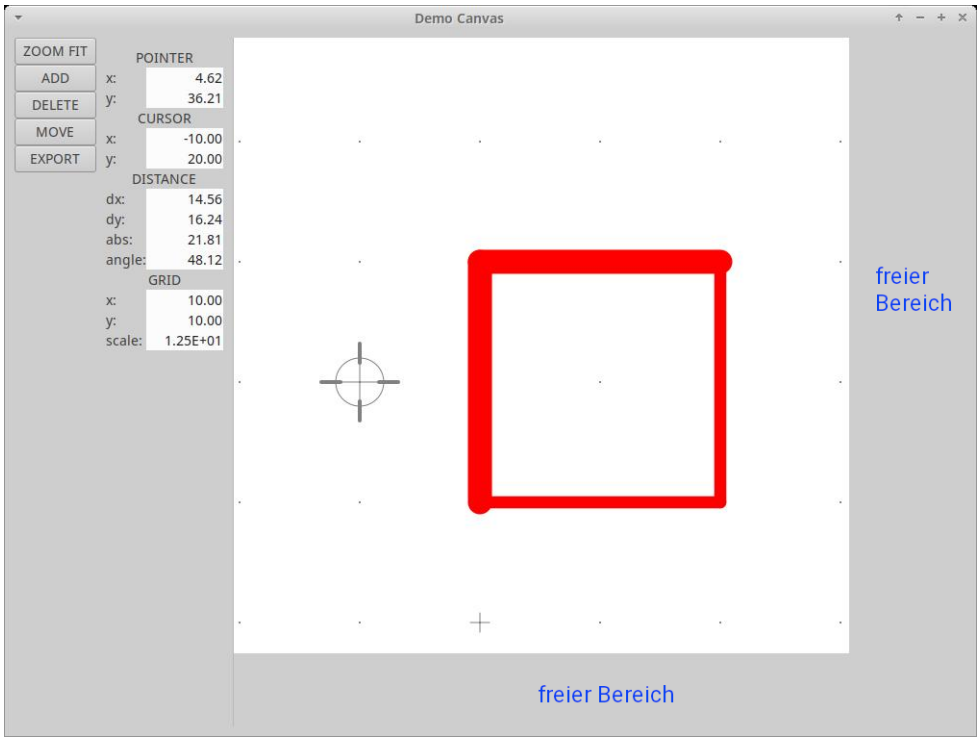

Abbildung 5.12: Zu wenig Leinwandfläche

Die einfachste Methode ist, einfach ausreichend große Festwerte für Breite und Höhe zu verwenden, um diesen Effekt zu vermeiden. Im Demoprogramm wurden diese Minimalabmessungen angewendet:

$$C_w = 100.000 \tag{5.33}$$

$$C_h = 100.000 \tag{5.34}$$

Die verwendeten Werte sind empirisch ermittelt worden. Bei sehr großen Bildschirmen müssen diese eventuell erhöht werden.

Wenn diese Lösung umgesetzt werden soll, muß die globale Variable `canvas_size` mit den oben festgelegten Werten initialisiert werden. Die Prozedur `set_up_canvas` initialisiert die Leinwand und weist diese Abmessungen der Leinwand zu. Siehe Quellcode in Abschnitt 9.5 und 9.8.

[9] engl. empty area

5.5.1.2 Konfiguration basierend auf größtmöglichem Begrenzungsrechteck

Dieser Ansatz wurde im Demoprogramm getestet und **erfolgreich** umgesetzt.

Der Grundgedanke ist, von dem größtmöglichen Begrenzungsrechteck B und maximalen Vergrößerungsfaktor ausgehend, die Abmessungen der Leinwand einmalig bei Systemstart zu berechnen. Die maximale Breite B_{w_max} und Höhe B_{h_max} des Modells sind Systemgrenzen, welche von der konkreten Anwendung abhängen (Siehe Abschitt 3.6.2). Betreffend der Festlegung des maximalen Vergrößerungsfaktor S_{max} sei auf Abschnitt 3.2.1 verwiesen. Die folgenden Schritte wurden in der Prozedur `compute_canvas_size` programmtechnisch umgesetzt. Siehe Quellcode in Abschnitt 9.5. Diese Prozedur wird im Demoprogramm einmalig bei Systemstart aufgerufen.

Eine wichtige Zwischengröße ist der maximale Basis-Offset F_{max}. Dieser muß zuerst ermittelt werden. Er wird nach gleichem Schema wie in Abschnitt 5.2 errechnet. Jedoch wird für die Breite des Begrenzungsrechtecks B_{w_max} und für die Höhe B_{h_max} eingesetzt. Wir handeln nun so, als hätten wir es mit einem Modell mit diese extremen Abmessungen zu tun.

Für den x-Anteil von F_{max} gilt:

$$F_{max\_x} = B_{w\_max}(S_{max} - 1) \tag{5.35}$$

Für die y-Komponente von F_{max} gilt:

$$F_{max\_y} = -B_{h\_max} \cdot S_{max} \tag{5.36}$$

Nun werden die Breite C_w und Höhe C_h der Leinwand berechnet.

Breite der Leinwand

Zur Berechnung der Breite betrachten wir Abbildung 5.13. Gesucht ist der Wert für C_w.

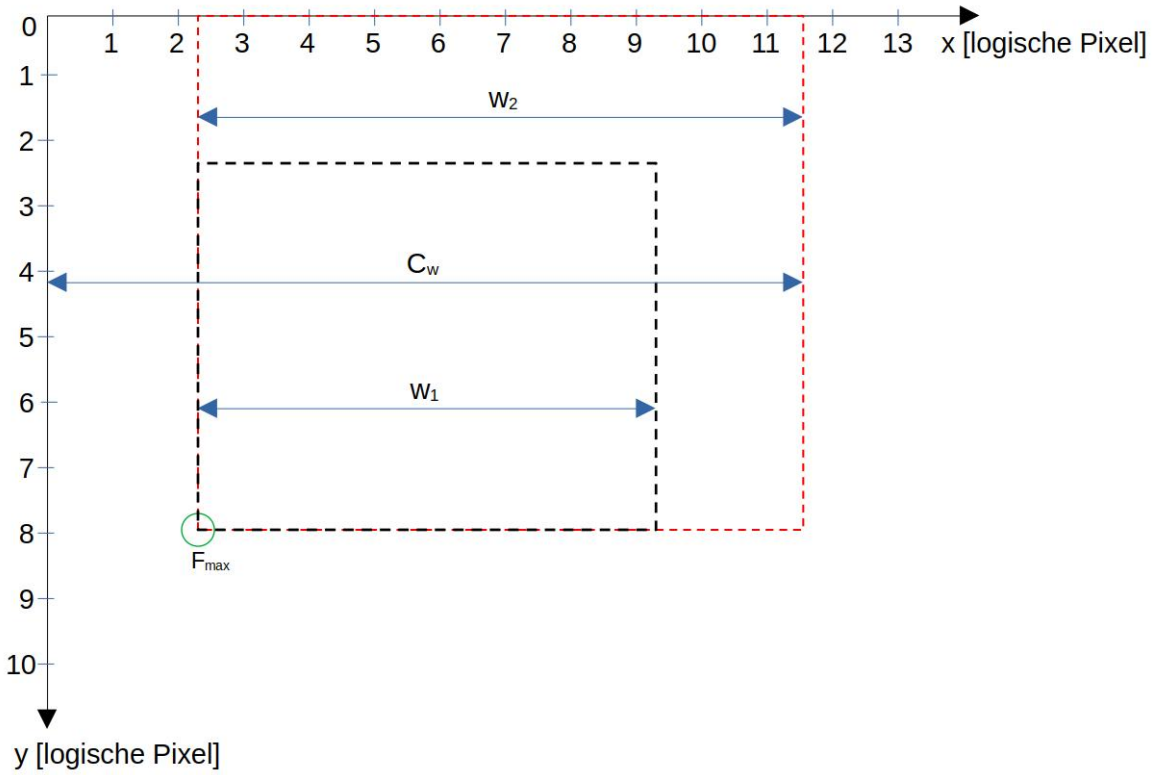

Abbildung 5.13: Breite der Leinwand

Ausgehend vom Punkt F_{max} dehnt sich das Bild bei maximaler Vergrößerung bis auf w_2 nach rechts aus. Die Breite der Leinwand ist also:

$$C_w = F_{max\_x} + w_2 \tag{5.37}$$

Die Spanne w_1 entspricht der maximalen Breite des Begrenzungsrechtecks B bei Vergrößerungsfaktor 1,0:

$$w_1 = B_{max\_w} \cdot 1,0 \frac{lp}{mm} \tag{5.38}$$

Bei maximaler Vergrößerung ergibt sich die Spanne w_2:

$$w_2 = w_1 \cdot S_{max} \tag{5.39}$$

Es ergibt sich also für die gesuchte Breite der Leinwand:

$$C_w = F_{max\_x} + B_{max\_w} \cdot S_{max} \tag{5.40}$$

Höhe der Leinwand

Für die Berechnung der Höhe sehen wir uns nun Abbildung 5.14 an. Gesucht ist der Wert für C_h.

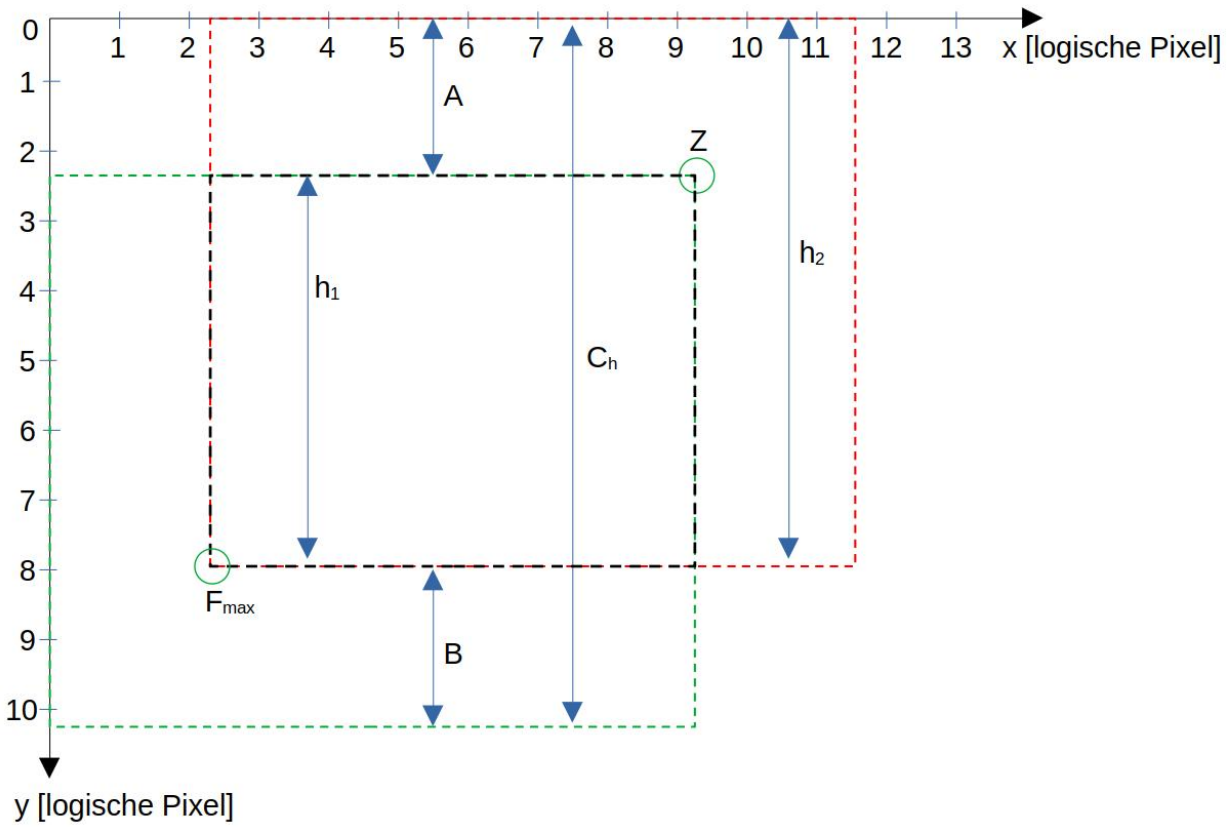

Abbildung 5.14: Höhe der Leinwand

Für die Höhe der Leinwand gilt:

$$C_h = -F_{max\_y} + B \tag{5.41}$$

Weil der y-Anteil des Basis-Offsets negativ ist, muß dieser mit -1 multipliziert werden, um eine positive Höhe zu erhalten.

Ausgehend vom Punkt F_{max} dehnt sich das Bild bei maximaler Vergrößerung um A nach oben aus. A ist die Differenz aus Höhe ohne Vergrößerung h_1 und Höhe bei maximaler Vergrößerung h_2:

$$A = h_2 - h_1 \tag{5.42}$$

Wenn im Punkt Z vergrößert wird, dehnt sich das Bild um den gleichen Betrag B nach unten aus. Also gilt:

$$B = h_2 - h_1 \tag{5.43}$$

Die Spanne h_1 entspricht der maximalen Höhe des Begrenzungsrechtecks ohne Vergrößerung, also bei Vergrößerungsfaktor 1,0:

$$h_1 = B_{max\_h} \cdot 1,0 \frac{lp}{mm} \tag{5.44}$$

Die Spanne h_2 wird vom maximalen Vergrößerungsfaktor bestimmt:

$$h_2 = h_1 \cdot S_{max} \tag{5.45}$$

Nun setzen wir Gleichung 5.43 in 5.41 ein:

$$C_h = -F_{max\_y} + (h_2 - h_1) \tag{5.46}$$

h_2 ersetzen wir durch Gleichung 5.45:

$$C_h = -F_{max\_y} + (h_1 \cdot S_{max} - h_1) \tag{5.47}$$

Ausklammern von h_1 liefert:

$$C_h = -F_{max\_y} + h_1(S_{max} - 1) \tag{5.48}$$

h_1 ersetzen wir durch Gleichung 5.44 und erhalten schließlich für die Höhe der Leinwand diese Formel:

$$C_h = -F_{max\_y} + B_{max\_h} \cdot 1,0\frac{lp}{mm}(S_{max} - 1) \tag{5.49}$$

5.5.2 Dynamische Konfiguration

Diese Methode wartet noch auf eine Umsetzung in die Praxis. Bisherige Experimente waren erfolglos. Die folgenden Überlegungen zeigen also **keine** funktionierende Lösung, sollen aber für die Zukunft hier festgehalten werden.

Der Grundgedanke ist, von den Anschlägen L und U der Rollbalken ausgehend, die Breite und Höhe der Leinwand festzulegen. Die Anschläge der Rollbalken werden immer beim Einpassen neu berechnet. Es liegt also nahe, immer unmittelbar danach die Abmessungen der Leinwand anzupassen. Die Umsetzung scheitert daran, daß eine **nachträgliche** Änderung der Abmessungen nach der Initialisierung der Leinwand nicht mehr möglich ist[10]. Der zugehörige experimentelle, auskommentierte Programmcode ist in der Prozedur `set_initial_scrollbar_settings` zu finden. Siehe Quellcode in Abschnitt 9.10.

5.5.2.1 Höhe

Wir sehen uns Abbildung 5.15 an. Der Punkt Z ist der Ort, an dem vergrößert oder verkleinert wird.

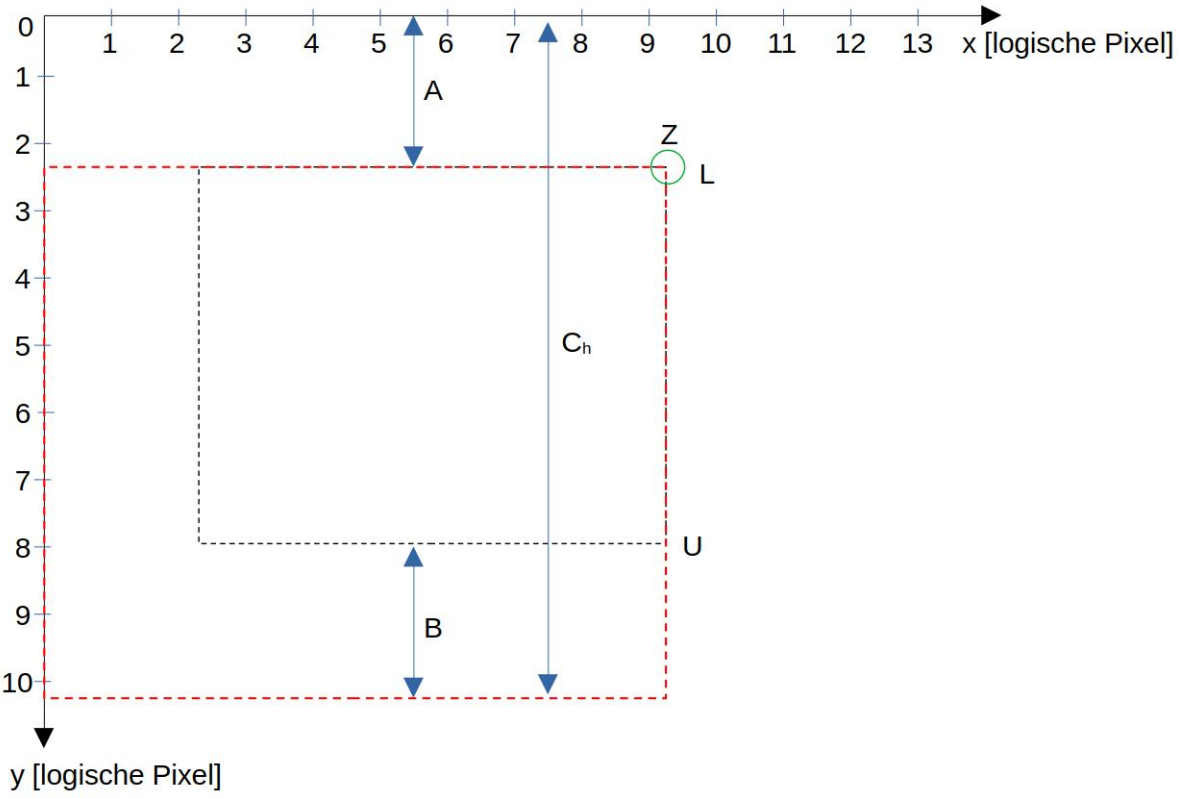

Abbildung 5.15: Berechnung Höhe Leinwand

Es werden die Anschläge des vertikalen Rollbalkens verwendet (Abschnitt 5.3.1). Der untere Anschlag L (lower limit) ist bekannt. Von dort nach oben, Richtung Y-Wert Null, besteht also ein Aussteuerbereich A.

Der obere Anschlag U (upper limit) ist ebenfalls bekannt. Von dort aus nach unten, Richtung steigender Y-Wert, muß der gleiche Aussteuerbereich der Höhe B vorhanden sein.

[10] Jedenfalls konnte ich dafür keine Lösung finden. Vielleicht kann ein erfahrener *GTK*-Anwender das Problem eines Tages lösen.

Es gilt also:

$$B = A \tag{5.50}$$

Die Höhe der Leinwand ist dann:

$$C_h = U + L \tag{5.51}$$

5.5.2.2 Breite

Wir betrachten Abbildung 5.16. Wieder ist der Punkt Z ist der Ort, an dem vergrößert oder verkleinert wird.

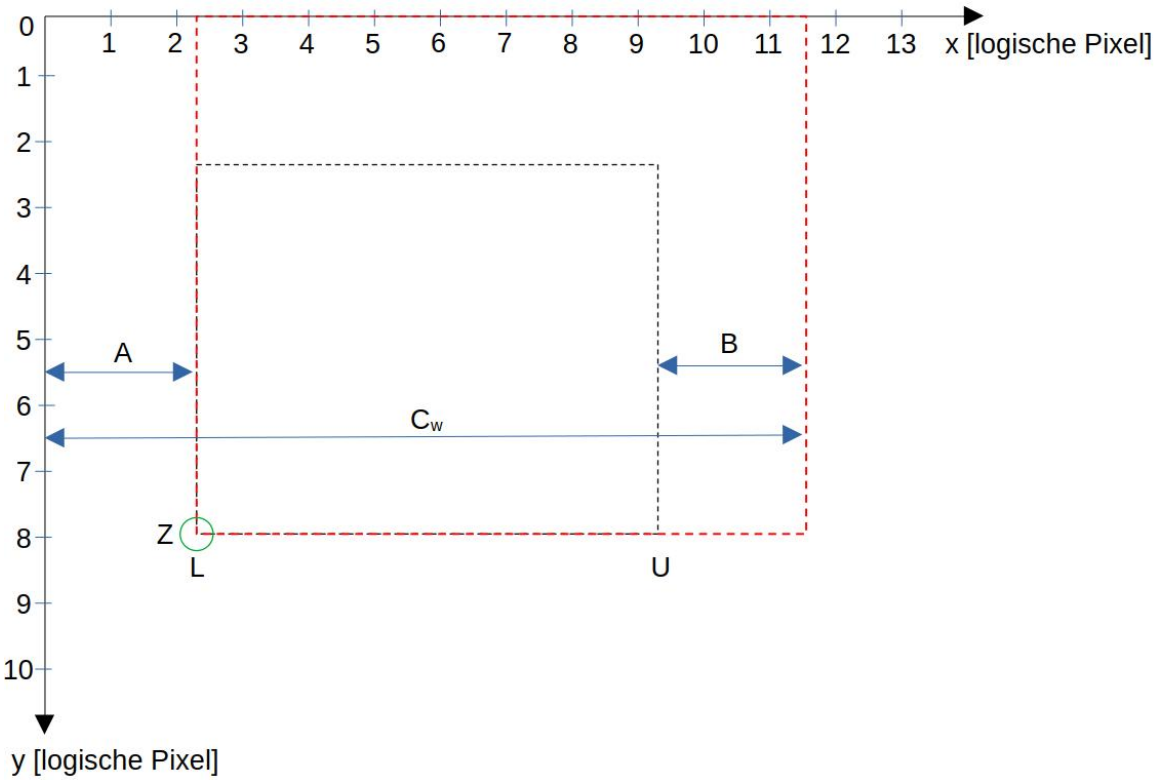

Abbildung 5.16: Berechnung Breite Leinwand

Es werden die Anschläge des horizontalen Rollbalkens verwendet. Siehe Abschnitt 5.3.2. Der untere Anschlag L (lower limit) ist bekannt (linkes Ende). Von dort weiter nach links, Richtung X-Wert Null, besteht also ein Aussteuerbereich A.

Der obere Anschlag U (upper limit) ist ebenfalls bekannt (rechtes Ende). Von dort aus weiter nach rechts, Richtung steigender X-Wert, muß der gleiche Aussteuerbereich der Breite B vorhanden sein. Es gilt also:

$$B = A \tag{5.52}$$

Die Breite der Leinwand ist dann:

$$C_w = U + L \tag{5.53}$$

Kapitel 6

Operativer Modus

Dieses Kapitel behandelt grundlegende Mechanismen, die im normalen Betrieb, also nach der Initialisierung ablaufen. Das Verständis für die Vorgänge beim Systemstart - behandelt in Kapitel 5 - ist nun eine Voraussetzung um die Abläufe in diesem Kapitel zu verstehen.

6.1 Objekte einpassen

Im Laufe des Betriebes ist es von Zeit zu Zeit nötig, alle Objekte einzupassen[1] um einen Gesamtüberblick zu erhalten. Dafür ist in der graphischen Bedienoberfläche der Knopf `Zoom Fit` o.ä. vorgesehen (siehe Abbildung 1.2). Alternativ kann auch eine Funktionstaste mit dieser Aufgabe verknüpft sein. Siehe Abschnitt 8.1.3 für die Bedienung des Demoprogrammes.

Die Verfahrensweise ist ähnlich der in Abschnitt 5.4 beschriebenen:

Ablauf

1. Translation-Offset T zurücksetzen auf (0;0).
2. Begrenzungsrechteck B berechnen. Siehe Abschitt 5.1.
3. Basis-Offset F berechnen.
4. Weil wir es nun mit einem neuen Begrenzungsrechteck B zu tun haben, müssen die Rollbalken neu eingestellt werden. Das wird mit der Prozedur `set_initial_scrollbar_settings` gemacht.
5. Weil der Bediener die Abmessungen des Hauptfensters jederzeit ändern kann, kann nun auch das Rollfenster völlig andere Dimensionen haben, als beim Systemstart. Entsprechend dieser Abmessungen des Rollfensters muß nun ein Vergrößerungsfaktor gefunden werden, bei dem alle Objekte (innerhalb des Begrenzungsrechtecks) durch das Rollfenster sichtbar sind. Wir haben es wieder mit zwei rechteckigen Flächen zu tun: Die Fläche des Rollfensters und die des Begrenzungsrechtecks B. Mittels der Funktion `get_ratio` wird das Größenverhältnis beider Flächen ermittelt und der Vergrößerungsfaktor S entsprechend gesetzt. (Siehe Abschnitt 5.4.1).
6. Nun muß der Translation-Offset T so gesetzt werden, daß der Inhalt des Begrenzungsrechtecks in der Mitte des Rollfenster erscheint.
   Das wird mit der Prozedur `center_to_visible_area` bewerkstelligt (Siehe Abschnitt 5.4.2).

[1] engl. Zoom-to-Fit

In der Prozedur `cb_zoom_to_fit` wurde dieser Ablauf umgesetzt. Siehe Quellcode in Abschnitt 9.8.

6.2 Bereich vergrößern

Um einen bestimmten Bereich der Leinwand zu vergrößern[2], ist in der graphischen Bedienoberfläche der Knopf `Zoom Area` vorgesehen (siehe Abbildung 1.2). Die Kurzfassung zur Bedienung ist in Abschnitt 8.1.4 zu finden. Die Bedienung ist so unspektakulär und selbstverständlich, daß es dieser Anleitung kaum bedarf.

Die Abbildungen 6.1 und 6.2 sollen zum Verständis der folgenden Unterabschnitte dienen.

Der ausgewählte Bereich ist in Abbildung 6.1 zu sehen. Er ist durch ein mit Strichellinie gezeichnetes Rechteck gekennzeichnet. Für die Spezifikation eines rechteckigen Bereiches bedarf es nur zweier Punkte, nämlich `k1` und `k2`. Diese Punkte sind Modellkoordinaten. Wichtig ist, daß sie sich diagonal gegenüberstehen. So kann `k1` auch unten links und `k2` oben rechts sein.

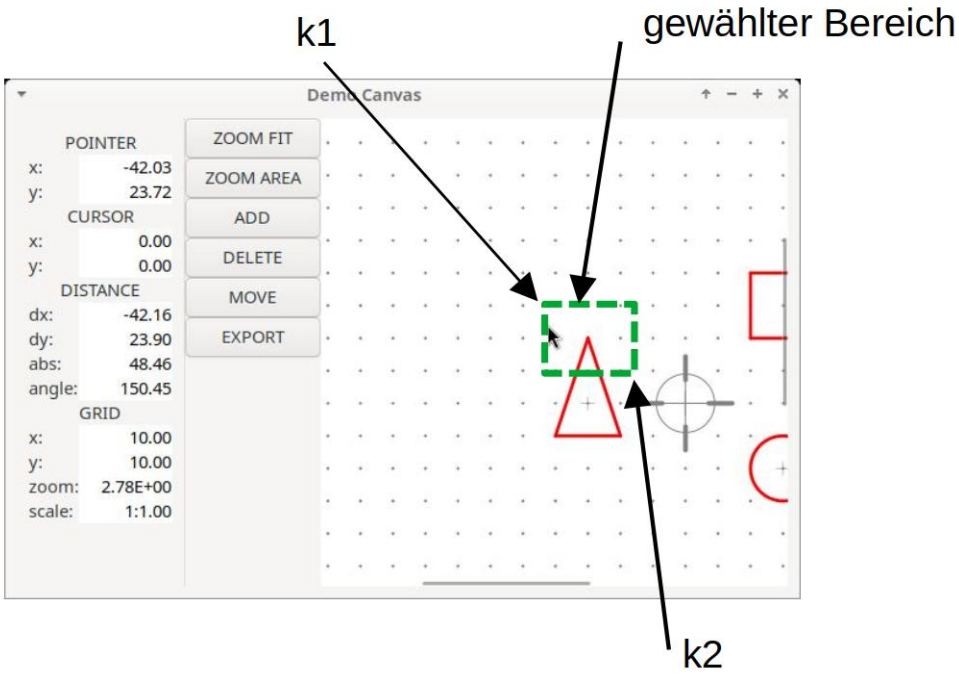

Abbildung 6.1: Bereich wählen

Sobald der Bediener den Bereich bestimmt hat, wird dieser vergrößert im Rollfenster angezeigt. Ziel ist es, den gesamten Bereich im Rollfenster zu zentrieren und mit einem angemessenen Vergrößerungsfaktor S zu skalieren. Im Beispiel wurde ein rechteckiger Bereich um die Spitze des Dreiecks herum ausgewählt. Die Spitze des Dreieckes soll also vergrößert werden.

[2] engl. Zoom-to-Area. Manche CAD-Systeme nennen diesen Vorgang `Zoom Selection`.

In Abbildung 6.2 ist der zuvor gewählte Bereich nun vergrößert zu sehen.

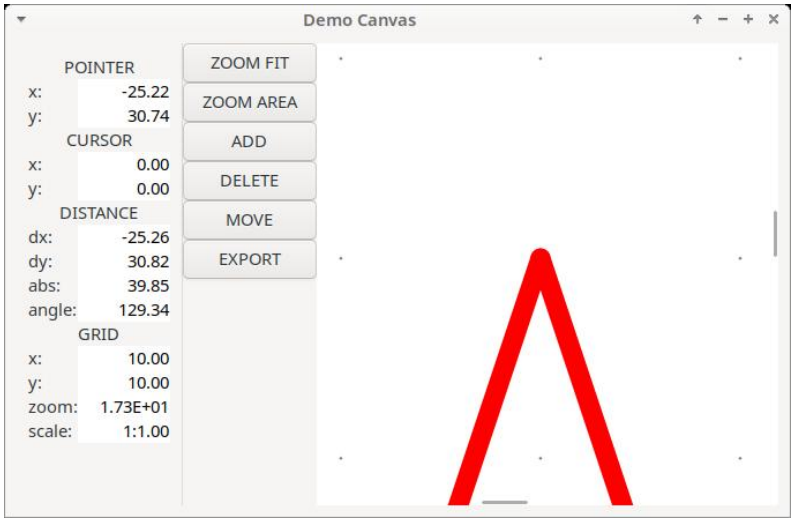

Abbildung 6.2: Bereich vergrößert

Der zu vergrößernde Bereich wird primär in der Modelldomäne (CS1) spezifiziert, aber auf der Leinwand (CS2) in Form eines Rechteckes visualisiert. Wir haben es mit zwei voneinander getrennten Mechanismen zu tun: Der primären Bereichswahl und der sekundären Visualisierung des Bereiches.

Zu den nachfolgenden Unterabschnitten sei ergänzend auf Details im Quellcode in Abschnitt 9.12 und 9.8 verwiesen. Insbesondere die exakte zeitliche Abfolge und damit verbundene Feinheiten sind dem Quellcode zu entnehmen.

6.2.1 Primäre Bereichswahl im Modell

Die Bereichswahl erfolgt primär in der Modelldomäne, also in CS1. Es ist folgendes Verfahren anzuwenden:

1. Durch Betätigen des Knopfes `Zoom Area` wird die Prozedur `cb_zoom_area` aufgerufen. Dies setzt das Flag `zoom_area.active`. Somit ist der Vorgang ausgelöst.

2. Sobald der Bediener die rechte Maustaste auf der Leinwand drückt,
   wird die Prozedur `cb_canvas_button_pressed` aufgerufen. Diese registriert den zugehörigen Modellpunkt k1. Der Bediener hält die rechte Maustaste gedrückt und bewegt den Mauszeiger an dem Ort diagonal gegenüberliegend zu k1.

3. Beim Loslassen der rechten Maustaste wird die Prozedur `cb_canvas_button_released` aufgerufen, der Modellpunkt `k2` registriert und das Flag `zoom_area.active` wieder zurückgesetzt.

4. Die Ecken des Begrenzungsrechteckes B mit Prozedur `get_bounding_box_corners` (Siehe Quellcode in Abschnitt 9.15) ermitteln und in Variable `C1` zu speichern. Die Ecken des Begrenzungsrechtecks B sind hier Leinwandkoordinaten - also logische Pixel - in Abhängigkeit vom gegenwärtigen Vergrößerungsfaktor S und Translation-Offset T.

5. Aus den Punkten `k1` und `k2` wird die eigentliche zu vergrößernde Fläche bestimmt. Falls `k1` und `k2` gleich groß sind, wird der Vorgang abgebrochen,
   und das Flag `zoom_area.active` bleibt gesetzt. Das passiert, wenn der Bediener an ein und der selben Stelle die Maustaste drückt und wieder losläßt.

6. Nun wird der Translation-Offset T zurückgesetzt auf (0,0;0,0) und die gewählte Fläche an die Prozedur `zoom_to_fit` übergeben. Das Ergebnis ist ein neuer Translation-Offset T und ein neuer Vergrößerungsfaktor S.

7. Die Ecken des Begrenzungsrechteckes B mit Prozedur `get_bounding_box_corners` ermitteln und in Variable `C2` speichern. `C2` enthält nun die Ecken entsprechend des neuen Vergrößerungsfaktors S und des neuen Translation-Offsets T.

8. Die Anschläge der Rollbalken mit Prozedur `update_scrollbar_limits` und ihren Argumenten `C1` (alte Ecken) und `C2` (neue Ecken) aktualisieren.

9. Abschließend die Konfiguration (Anschläge und Einstellungen) der Rollbalken mit Prozedur `backup_scrollbar_settings` sichern.

6.2.2 Visualisierung auf der Leinwand

Hier geht es nur um die Visualisierung des zu vergrößernden Bereiches auf der Leinwand, also in CS2. Es wird ein Rechteck gezeichnet, welches vom Zeitpunkt des Drückens bis zum Loslassen der rechten Maustaste zu sehen ist. Siehe Abbildung 6.2. Es handelt sich um eine reine Hilfsfunktionen, die mit der primären Bereichswahl in Abschnitt 6.2.1 wenig zu tun hat.

In der Leinwanddomäne ist folgendes Verfahren anzuwenden:

1. Sobald der Bediener die rechte Maustaste auf der Leinwand drückt,
   wird die Prozedur `cb_canvas_button_pressed` aufgerufen. Diese setzt das Flag `zoom_area.started` und registriert den zugehörigen Leinwandpunkt als Startpunkt `l1`.

2. Während die Maus bewegt wird, aktualisiert die Prozedur `cb_mouse_moved` den Endpunkt `l2` fortlaufend und löst das Neuzeichnen der Leinwand mittels der Prozedur `refresh` aus (Quellcode in Abschnitt 9.5).

3. Solange das Flag `zoom_area.started` gesetzt ist, malt die Prozedur `draw_zoom_area` das sich aus `l1` und `l2` ergebende Rechteck. Dieses Rechteck wird direkt in Leinwandkoordinaten berechnet und gezeichnet.

4. Mit dem Loslassen der rechten Maustaste wird das Flag `zoom_area.started` zurückgesetzt. Die Prozedur `draw_zoom_area` zeichnet daraufhin nichts mehr, und das Rechteck verschwindet.

6.3 Berechnung des Translation-Offsets beim Skalieren

Die in diesem Abschnitt beschriebenen Mechanismen sind grundlegend für alles, was mit dem Vergrößern oder Verkleinern (zoom-in, zoom-out) an einem bestimmten Punkt zu tun hat. Dieser Punkt ist das Zentrum einer Vergrößerung oder Verkleinerung. Wir verwenden das allgemeinere Wort "Skalierung" in diesem Zusammenhang. Das Zentrum kann ein realer Punkt im Modell (CS1) oder ein Punkt auf der Leinwand sein (CS2).

Auch wenn die hier vorgestellten Mechanismen sehr einfach wirken, war die Ausarbeitung der theoretischen Grundlagen und die programmtechnische Umsetzung eine der größten Herausforderungen der vorliegenden Arbeit. Zum gedanklichen Nachvollziehen der Abläufe sollen hier die einzelnen Schritte zur Lösung dokumentiert werden.

Der Translation-Offset T ist beim **jedem** Skalieren, also Vergößern oder Verkleinern, neu zu berechnen, weil das Bild im Zentrum der Skalierung stehen bleiben muß. Das Bild wird gewissermaßen um den Betrag T "zurückgezerrt". Der Bediener hat dann den Eindruck, als würde das Bild sich um das Zentrum herum ausdehnen oder oder schrumpfen. Um die Ermittlung des dafür nötigen Translation-Offsets T soll es in diesem Abschnitt gehen.

Die Berechnung basiert auf drei gegebenen Werten:

1. gegenwärtiger alter Vergrößerungsfaktor S_1 VOR der Skalierung
2. gewünschter neuer Vergrößerungsfaktor S_2 NACH der Skalierung
3. Zentrum der Skalierung. Dieses kann als entweder als Leinwandpunkt Z_1 (im CS2) oder als realer Modellpunkt M (im CS1) gegeben sein.

Die Handlungen für diese beiden möglichen Fälle wurden in den überladenen[3] Prozeduren `set_translation_for_zoom` umgesetzt. Siehe Quellcode in Abschnitt 9.12.

Die Vorgehensweise in diesen beiden Situationen soll nun in den nachfolgenden Abschnitten näher betrachtet werden.

[3] "Überladen" bedeutet, daß sich gleichnamige Prozeduren in ihren Argumenten unterscheiden.

6.3.1 Leinwandpunkt als Zentrum

Wenn z.B. an der Position des Mauszeigers skaliert wird, tritt dieser Fall ein. Es handelt sich um das häufigste Szenario im Zusammenhang mit Skalieroperationen. Das Zentrum der Skalierung ist als Leinwandpunkt Z_1 gegeben. Abbildung 6.3 zeigt die nötigen Arbeitsschritte um zum gesuchten Translation-Offset T_2 zu gelangen.

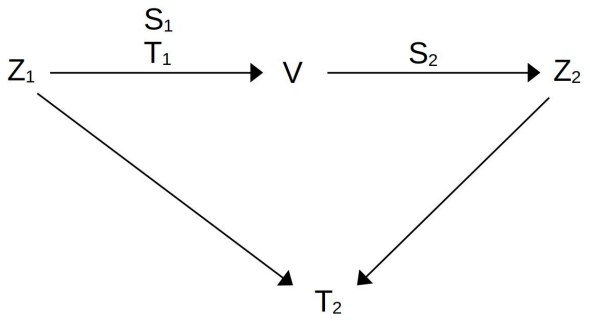

Abbildung 6.3: von Leinwandpunkt nach Translation-Offset

Im Detail ist nun so vorzugehen:

1. Z_1 ist der Leinwandpunkt VOR der Skalierung. Mittels der Funktion `canvas_to_virtual` wird der Leinwandpunkt Z_1 in den zugehörigen virtuellen Modellpunkt V umgerechnet. Der Umrechnung liegt der gegenwärtige Vergrößerungsfaktor S_1 zugrunde. Weil zu diesem Zeitpunkt durchaus schon eine Verschiebung des Bildes bestehen könnte, ist auch der gegenwärtige Translation-Offset T_1 in die Rechnung einzubeziehen. Die Funktion `canvas_to_virtual` macht das grundsätzlich immer. Siehe Abschnitt 4.4 für Details.

2. Nun wird vom virtuellen Punkt V ausgehend mittels der Funktion `virtual_to_canvas` ein zweiter Leinwandpunkt Z_2 berechnent. Dieser Umrechnung liegt der gewünschte neue Vergrößerungsfaktor S_2 zugrunde. Der gegenwärtige Translation-Offset T wird **nicht** berücksichtigt[4]. Z_2 ist nun der zu V gehörige Leinwandpunkt NACH der Skalierung. Z_2 ist als als eine Art "Vorschau" zu verstehen. Siehe Abschnitt 4.3 für Details.

3. Dann wird die Entfernung zwischen den beiden Leinwandpunkten ermittelt. Die Entfernung von Z_1 nach Z_2 ist

$$\vec{D} = \vec{Z}_2 - \vec{Z}_1 \tag{6.1}$$

Um ein scheinbar stehendes Bild zu erhalten, muß es um D "zurückgezerrt" werden. Darum wird D nun mit -1 multipliziert was den neuen Translation-Offset T_2 ergibt:

$$\vec{T}_2 = -\vec{D} \tag{6.2}$$

Durch Zusammenfassen obiger Gleichungen erhalten wir schließlich diese einfache Formel für den neuen Translation-Offset T_2:

$$\vec{T}_2 = \vec{Z}_1 - \vec{Z}_2 \tag{6.3}$$

[4]Durch das Argument `translate` wird der Funktion `virtual_to_canvas` mitgeteilt, daß der Translation-Offset nicht zu berücksichtigen ist.

6.3.2 Realer Modellpunkt als Zentrum

Das Zentrum ist als realer Modellpunkt M gegeben. Wenn z.B. an der Position des Cursors vergrößert oder verkleinert werden soll, tritt dieser Fall ein. In Abbildung 6.4 sind die nötigen Arbeitsschritte um zum gesuchten Translation-Offset T_2 vereinfacht dargestellt.

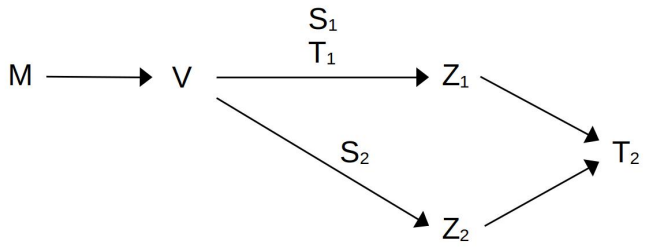

Abbildung 6.4: von Modellpunkt nach Translation-Offset

Die detaillierte Vorgehensweise ist wie folgt:

1. Der gegebene reale Punkt M wird in einen virtuellen Modellpunkt V umgerechet. Dafür ist die Funktion `to_virtual` zu verwenden.

2. Mittels der Funktion `virtual_to_canvas` wird von V ausgehend der zugehörige Leinwandpunkt Z_1 bestimmt. Z_1 ist der Leinwandpunkt VOR der Skalierung. Der Umrechnung liegt der gegenwärtige Vergrößerungsfaktor S_1 zugrunde. Weil zu diesem Zeitpunkt durchaus schon eine Verschiebung bestehen könnte, ist der gegenwärtige Translation-Offset T_1 in der Rechnung zu berücksichtigen. Siehe Abschnitt 4.4 für Details.

3. Nun wird von V ausgehend mittels der Funktion `virtual_to_canvas` ein zweiter Leinwand-Punkt Z_2 berechnet. Dieser Umrechnung liegt der gewünschte neue Vergrößerungsfaktor S_2 zugrunde. Der gegenwärtige Translation-Offset T_1 wird **nicht** berücksichtigt. Durch das Argument `translate` wird der Funktion `virtual_to_canvas` mitgeteilt, daß der Translation-Offset nicht zu berücksichtigen ist. Z_2 ist nun der zu V gehörige Leinwandpunkt NACH der Skalierung. Z_2 ist als als eine Art "Vorschau" zu verstehen.

4. Der Abstand zwischen den beiden Leinwandpunkten bestimmt die nötige Verschiebung des Bildes:
$$\vec{D} = \vec{Z}_2 - \vec{Z}_1 \tag{6.4}$$
Um ein scheinbar stehendes Bild zu erhalten, muß es um D "zurückgezerrt" werden. Darum wird D nun mit -1 multipliziert was den neuen Translation-Offset T_2 ergibt:
$$\vec{T}_2 = -\vec{D} \tag{6.5}$$
Die Vereinfachung obiger Gleichungen liefert schließlich diese einfache Formel für den neuen Translation-Offset T_2:
$$\vec{T}_2 = \vec{Z}_1 - \vec{Z}_2 \tag{6.6}$$

6.4 Veränderung der Rollbalken beim Skalieren

Bei Vergrößerungsfaktoren größer 1,0 wird mehr Leinwand in Anspruch genommen. Das bedeutet für die Anschläge der Rollbalken, daß diese sich der Ausdehnung anpassen müssen. So wird erreicht, daß der Bediener durch Verschieben der Rollbalken jeden Teil der Zeichnung erreichen kann. Umgekehrt - beim Verkleinern - wird weniger Leinwand gebraucht. Auch dann müssen die Anschläge sich anpassen.

Der Weg zur hier beschriebenen Lösung war steinig und geprägt von Versuch und Irrtum. Deshalb sollen die Grundgedanken und Einzelschritte im Folgenden detailliert niedergeschrieben werden. Abbildung 6.5 soll zum Einstieg in die Problematik und dem Verständnis der Abläufe helfen.

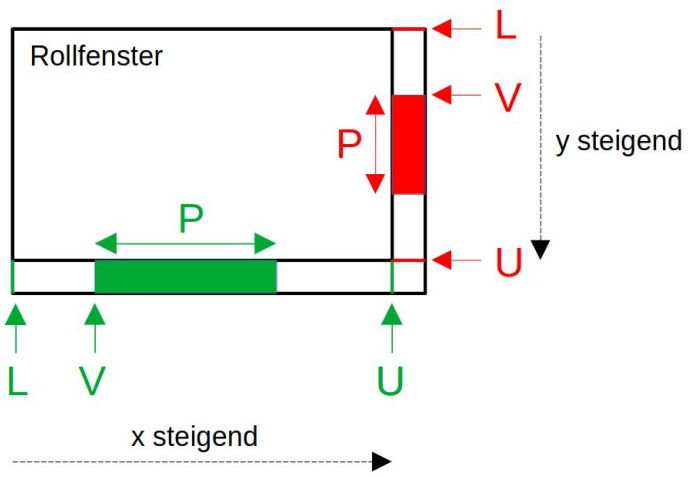

Abbildung 6.5: Anschläge der Rollbalken

Grundsätzlich gilt für die nachfolgend beschriebenen Mechanismen:

1. Der Einstellwert V - oder die Position - bleibt **unverändert**. V wird einzig durch den **Bediener** beim Verschieben des Balkens beeinflußt.
2. Auch die Seitengröße P bleibt **unverändert**. P wurde bei der Initialisierung einmalig bestimmt (Siehe Abschnitt 5.3.1.3 und 5.3.2.3).
3. Es wird **nur** der untere (L) und/oder der obere (U) Anschlag geändert.

Das Begrenzungsrechteck B hat vier Ecken. Es handelt sich um reale Modellkoordinaten[5]. Die Ecken werden wie folgt bezeichnet:

1. oben links (engl. top left), abgekürzt mit `TL`
2. oben rechts (engl. top right), abgekürzt mit `TR`
3. unten links (engl. bottom left), abgekürzt mit `BL`
4. unten rechts (engl. bottom right), abgekürzt mit `BR`

Im Beispiel in Abbildung 6.6 sind die Ecken eingtragen. Dort ist TL auf Position (-5/5), TR auf (6/5), BL auf (-5/-4) und BR auf (6/-4).

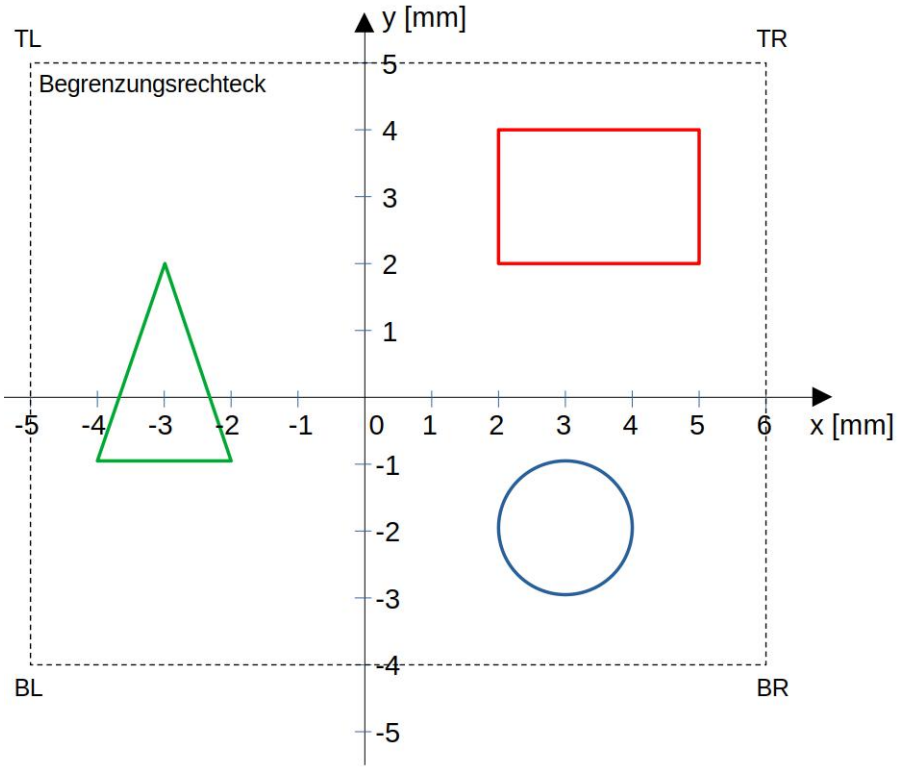

Abbildung 6.6: Ecken des Begrenzungsrechtecks

Ausgangspunkt für die Berechnung der Anschläge der Rollbalken sind die Ecken des Begrenzungsrechtecks - umgerechnet in Leinwandkoordinaten - **vor** und **nach** der Skalierung. Mit C1 bezeichnen wir zusammenfassend die Ecken vor der Skalierung. C2 sind die Ecken nach der Skalierung.

Wärend die Modellkoordinaten der Ecken immer gleich sind, ergeben sich nach der Umrechnung in Leinwandkoordinaten Punkte, die vom gegenwärtigen Vergrößerungsfaktor S und Translation-Offset T abhängen. Für diese Umrechnung ist die Funktion `get_bounding_box_corners` vorgesehen. Siehe Quellcode in Abschnitt 9.15. Mit C1 und C2 geht es dann weiter zur Ermittlung der Anschläge L und U der Rollbalken.

[5]Das Begrenzungsrechteck B kann für diesen Moment als etwas Konstantes betrachtet werden. Es ändert sich erst wieder nach dem Einpassen.

Es gilt nun diese Abfolge, welche in der Funktion `cb_mouse_wheel_rolled`, Unterprozedur `zoom` umgesetzt wurde. Siehe auch Quellcode in Abschnitt 9.8:

1. C1 ermitteln mit Funktion `get_bounding_box_corners`
2. Vergrößerungsfaktor S erhöhen oder verringern
3. Translation-Offset T mit Prozedur `set_translation_for_zoom` berechnen. Siehe 6.3.
4. C2 ermitteln mit Funktion `get_bounding_box_corners`.
5. Anschläge der Rollbalken mit Prozedur `update_scrollbar_limits` aktualisieren. Die nachfolgenden Unterabschnitte liefern die Details. Siehe Quellcode in Abschnitt 9.10.

In den folgenden Unterabschnitten wird eine etwas eigentümliche Schreibweise verwendet. Darum hier eine kleine Erläuterung:

Die Ecken des Begrenzungsrechteckes, also C1 und C2, sind Sammelbegriffe. Darin enthalten sind jeweils 4 Ecken, welche wiederum eine x- und eine y-Position haben. Um diese Hierarchie kenntlich zu machen, wird ein Punkt als Trenner verwendet. So bedeutet z.B. C2.BL.x :

- C2 - Ecken nach der Skalierung
- BL - unten links
- x - x-Position

6.4.1 Horizontaler Rollbalken

6.4.1.1 Unterer Anschlag

Die Veränderung ist:

$$dL = C2.BL.x - C1.BL.x \tag{6.7}$$

Der alte Wert für den unteren Anschlag ist L_1. Zu diesem wird die Änderung dL addiert, was den neuen unteren Anschlag L_2 ergibt:

$$L_2 = L_1 + dL \tag{6.8}$$

Es gelten folgende Einschränkungen für den neuen unteren Anschlag L_2:

- L_2 muß positiv sein weil es grundsätzlich keine negativen Werte für die Konfiguration der Rollbalken gibt. Wenn Gleichung 6.8 einen negativen Wert liefert, ist L_2 auf 0 zu setzen.
- L_2 muß kleiner als der Einstellwert V sein, weil der untere Anschlag L immer **links** von V sein muß. Aus Abbildung 6.5 sollte dies hervorgehen. Liefert Gleichung 6.8 einen Wert größer als V, dann ist für L_2 der gegenwärtige Einstellwert V einzusetzen.

6.4.1.2 Oberer Anschlag

Die Veränderung ist:

$$dU = C2.BR.x - C1.BR.x \tag{6.9}$$

Der minimal erlaubte Wert des oberen Anschlages U ist die Summe aus gegenwärtigem Einstellwert V und der Seitenbreite P, was in Abbildung 6.5 erkennbar sein sollte:

$$U_{min} = V + P \tag{6.10}$$

Der maximal erlaubte Wert des oberen Anschlages U entspricht dem rechten Ende der Leinwand:

$$U_{max} = C_{width} \tag{6.11}$$

Der alte Wert für den oberen Anschlag ist U_1. Zu diesem wird nun die Änderung dU addiert, was den neuen oberen Anschlag U_2 ergibt:

$$U_2 = U_1 + dU \tag{6.12}$$

Die Grenzen U_{min} und U_{max} müssen unbedingt eingehalten werden. Es gelten also diese Einschränkungen:

- U_2 muß größer oder gleich U_{min} sein. Anderenfalls ist U_2 auf U_{min} zu setzen.
- U_2 muß kleiner oder gleich der Breite der Leinwand sein. Anderenfalls ist U_2 auf U_{max} zu setzen.

6.4.2 Vertikaler Rollbalken

6.4.2.1 Unterer Anschlag

Die Veränderung ist:

$$dL = C2.TL.y - C1.TL.y \tag{6.13}$$

Der alte Wert für den unteren Anschlag ist L_1. Zu diesem wird die Änderung dL addiert, was den neuen unteren Anschlag L_2 ergibt:

$$L_2 = L_1 + dL \tag{6.14}$$

Es gelten folgende Einschränkungen für den neuen unteren Anschlag L_2:

- L_2 muß positiv sein weil es grundsätzlich keine negativen Werte für die Konfiguration der Rollbalken gibt. Wenn Gleichung 6.8 einen negativen Wert liefert, ist L_2 auf 0 zu setzen.
- L_2 muß kleiner als der Einstellwert V sein, weil der untere Anschlag L immer **oberhalb** von V sein muß. Aus Abbildung 6.5 sollte dies hervorgehen. Zu beachten ist, daß der untere Anschlag des vertikalen Rollbalkens in der Abbildung oben ist! Liefert Gleichung 6.14 einen Wert größer als V, dann ist für L_2 der gegenwärtige Einstellwert V einzusetzen.

6.4.2.2 Oberer Anschlag

Es ist zu beachten, daß sich der obere Anschlag des vertikalen Rollbalkens in der Abbildung 6.5 **unten** befindet! Die Veränderung ist:

$$dU = C2.BL.y - C1.BL.y \tag{6.15}$$

Der minimal erlaubte Wert des oberen Anschlages ist die Summe aus gegenwärtigem Einstellwert V und der Seitenbreite P:

$$U_{min} = V + P \tag{6.16}$$

Der maximale erlaubte Wert des oberen Anschlages entspricht dem unteren Ende der Leinwand:

$$U_{max} = C_{height} \tag{6.17}$$

Der alte Wert für den oberen Anschlag ist U_1. Zu diesem wird die Änderung dU addiert, was den neuen oberen Anschlag U_2 ergibt:

$$U_2 = U_1 + dU \tag{6.18}$$

Die Grenzen U_{min} und U_{max} müssen unbedingt eingehalten werden, was diese Einschränkungen zur Folge hat:

- U_2 muß größer oder gleich U_{min} sein. Anderenfalls ist U_2 auf U_{min} zu setzen.
- U_2 muß kleiner oder gleich der Höhe der Leinwand sein. Anderenfalls ist U_2 auf U_{max} zu setzen.

6.5 Größe des Rollfensters, Leinwand und Rollbalken

In diesem Abschnitt soll der Einfluß der Größe des Rollfensters auf Leinwand und Rollbalken untersucht und eine praktikable Lösung aufgezeigt werden. Das Finden einer vernünftigen und effektiven Lösung erwies sich als das schwierigste Problem in dem vorliegenden Werk.

Das Rollfenster hat eine Initialgröße, wie in Abschnitt 3.8 ausgeführt wurde. Mit dieser Größe ist es unmittelbar nach Systemstart zu sehen. Der Bediener hat jedoch im operativen Modus - also während der ganz normalen Arbeit mit dem Programm - jederzeit die Möglichkeit, die Größe des Hauptfensters zu ändern[6], indem er z.B. mit der Maus die Ränder des Hauptfensters verschiebt oder es maximiert.

Weil das Rollfenster samt aller anderen Bedienelemente im Hauptfenster eingebettet ist, ändert sich bei diesem Vorgang auch die Größe des Rollfensters. In Abbildung 6.7 sollte dieser Zusammenhang erkennbar sein. Es handelt sich um ein allgemeines Prinzip, das wir von vielen anderen graphischen Benutzeroberflächen kennen. In unserem Demoprogramm ist die Größe der Koordinatenanzeige (für Mauszeiger, Cursor, Entfernungen, Raster, Vergrößerung und Maßstab) mit konstanter Größe programmiert. Zwangsläufig müssen also die Abmessungen des Rollfensters (auf der rechten Seite) proportional zu denen des Hauptfensters angepaßt werden. Das passiert durch *GTK*-Internas automatisch, sodaß wir dazu keine weiteren Worte verlieren müssen.

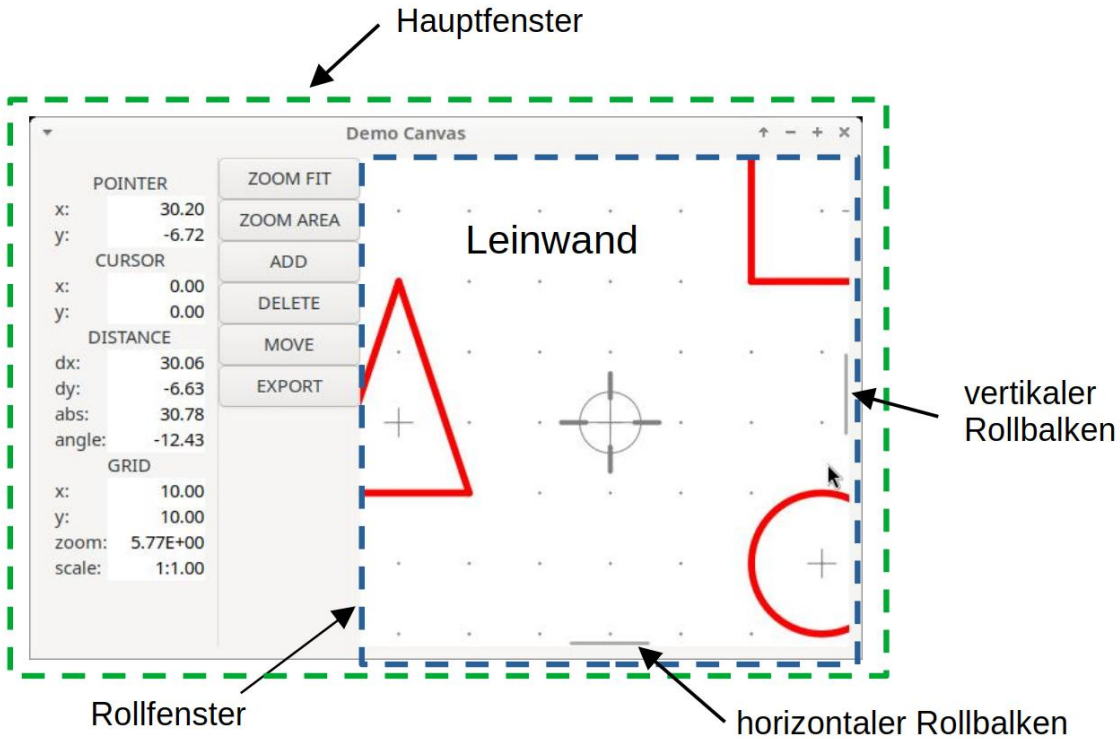

Abbildung 6.7: Größenveränderung Rollfenster

Soviel zum zu lösenden Problem. Es stellen sich nun folgende Fragen im Zusammenhang mit der Größenveränderung des Rollfensters:

- Welche Auswirkungen hat das auf die Leinwand und die Rollbalken ?
- Wie ist das sichtbare Bild im Rollfenster zu zentrieren ?

[6] engl. resizing, resize

- Ist der Vergrößerungsfaktor ebenfalls zu ändern ?

Bei diesem Vorgang muß das sichtbare Bild - relativ zur unteren Kante des Rollfensters - fixiert werden. Das läßt sich zum Beispiel durch Änderung des y-Anteils des Basis-Offsets F bewerkstelligen. Die y-Komponente von F ist also so zu ändern, daß der untere Rand des Bildes sich im gleichen Maße verschiebt wie der untere Rand des Rollfensters. Außerdem kann das Bild optional im Rollfenster zentriert werden.

Hinsichtlich des Vergrößerungsfaktors S bestehen zwei Möglichkeiten:

1. Man kann die Vergrößerung proportional mit der Größe des Rollfensters ändern.

2. Die Vergrößerung bleibt unverändert.

Die hierfür nötigen Mechanismen sollen in den folgenden Unterabschnitten detailliert erklärt werden.

Die nachfolgenden Überlegungen wurden in der Prozedur `cb_swin_size_allocate` in die Praxis umgesetzt. Der Quellcode in Abschnitt 9.8 ist zum Verständis unbedingt nötig. Ich hoffe, ausreichend Kommentare im Quellcode hinterlassen zu haben. Der Leser möge es mir nachsehen, daß diese nur in englischer Sprache geschrieben sind.

6.5.1 Erkennung der Größenänderung

Zunächst muß eine Größenveränderung überhaupt erkannt werden. Dann muß das Maß der Veränderung ermittelt und in Breiten- und Höhenveränderung unterschieden werden.

Die Prozedur `cb_swin_size_allocate` wird durch das Signal `size_allocate` ausgelöst. Das ist der Fall, wenn sich die Breite, Höhe oder Position des Rollfensters ändert. Wir wollen aber nur die Veränderung von Breite und Höhe erkennen. Das gelingt, wenn man die gegenwärtige Größe mit der vorherigen Größe vergleicht. Die gegenwärtige Größe bekommt die Prozedur `cb_swin_size_allocate` mit ihrem Aufruf automatisch übergeben. Die vorherige Größe ist über die globale Variable `swin_size` bekannt. Am Ende der Prozedur wird `swin_size` mit der gegenwärtigen Größe aktualisiert um für die nächste Größenänderung vorbereitet zu sein.

Es müssen also beide Größen einfach nur miteinander verglichen werden. Weichen sie voneinander ab, so hat eine Größenveränderung stattgefunden. Fand keine Größenveränderung statt, ist die Prozedur `cb_swin_size_allocate` zu Ende, ansonsten sind die nachfolgenden Schritte abzuarbeiten.

6.5.2 Wiederherstellung der Konfiguration der Rollbalken

Es gibt im Zusammenhang mit der Größenveränderung des Rollfensterns eine unschöne Eigenheit der Rollbalken. Die Einstellungen, also untere und obere Anschläge und Seitengröße passen sich mit jeder noch so kleinen Größenveränderung der neuen Größe des Rollfensters an. Es ist darum also erforderlich, die Einstellungen schnellstens wieder herzustellen indem die Prozedur `restore_scrollbar_settings` aufgerufen wird.

6.5.3 Synchronisation Bild-Unterkante mit Rollfenster

Ein grundsätzliches Problem ist, daß das Rollfenster an der oberen linken Ecke seinen Ursprung hat. Es dehnt sich nur nach rechts-unten aus oder schrumpft nach links-oben. Das im Rollfenster sichtbare Bild soll aber relativ zur unteren Kante des Rollfensters still stehen.

Das Problem der nach unten steigenden Y-Werte macht sich auch hier wieder bemerkbar. Es ist egal, in welcher Weise sich die Maße des Rollfensters ändern, ob sich die obere, untere, linke oder rechte Kante verschiebt. Ändert sich die Höhe des Rollfensters, muß das Bild in gleichem Maße in Y-Richtung verschoben werden. Der Bediener hat dann den Eindruck, als stünde das Bild im Rollfenster still.

Die Leinwand ist durch ihre Allokation[7] an die obere linke Ecke des Rollfensters bei (0;0) gebunden. Daran werden wir auch **nichts** ändern. Wir müssen das Bild auf eine andere Weise verschieben.

Abbildung 6.8 zeigt die Problematik und die von der Größenveränderung des Rollfensters betroffenen Maße.

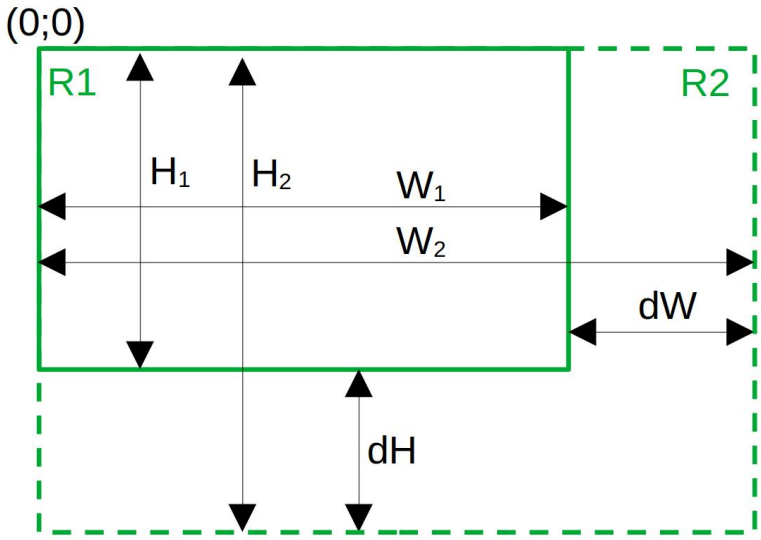

Abbildung 6.8: Änderung der Größe des Rollfensters

Das Rollfenster R1 vor der Änderung ist mit durchgezogener Linie dargestellt. R2 mit gestrichelter Linie ist das Rollfenster nach der Änderung. W_1 und H_1 sind die Breite und Höhe des Fensters VOR der Änderung. W_2 und H_2 sind die Breite und Höhe des Fensters NACH der Änderung. Es kann sich die Breite bei unveränderter Höhe verändern wie auch die Höhe bei unveränderter Breite. Es kann sich aber auch Breite und Höhe gleichzeitig verändern.

Die Veränderung der Breite ist:

$$dW = W_2 - W_1 \tag{6.19}$$

Für die Veränderung der Höhe gilt:

$$dH = H_2 - H_1 \tag{6.20}$$

dW und dH müssen grundsätzlich nach **jeder** Änderung der Fenstergröße berechnet werden. Nun stellt sich die Frage, wie und wann dW und dH weiterverwendet werden.

[7] Allokation bedeutet in diesem Zusammenhang: Zuweisung von Position, Breite und Höhe

Die Verschiebung des Bildes in y-Richtung, also nach oben oder unten, kann einfach bewerkstelligt werden, indem man mit dH den Basis-Offset F modifizert[8]. In Abbildung 6.9 ist das Rollfenster R1 vor der Änderung wieder mit durchgezogener Linie zu sehen. R2 ist das Rollfenster nach der Veränderung mit gestricheltem Linienzug. D1 soll mit durchgezogener Linie das Bild vor der Änderung andeuten. D2 ist das Bild nach der Veränderung mit gestrichelter Linie und zwar um den Betrag dH gegenüber D1 nach unten verschoben.

Ist dH positiv (Höhe hat zugenommen), wird das Bild um dH nach unten verschoben. Ist dH negativ (Höhe wurde verringert), rückt das Bild um dH nach oben.

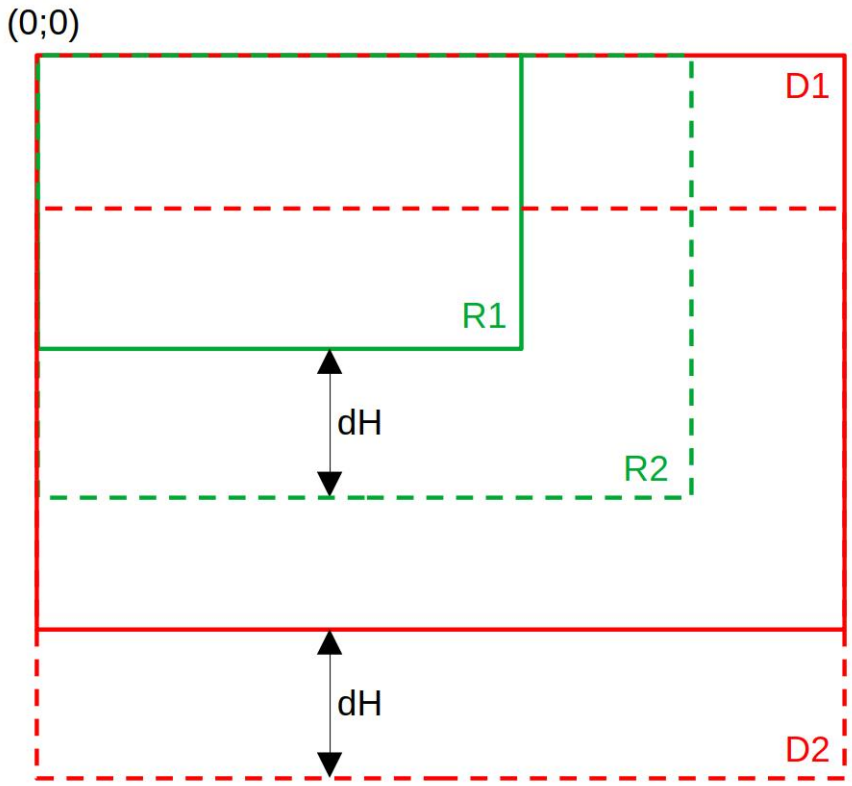

Abbildung 6.9: Vertikale Verschiebung des Bildes

Zusammenfassend - in der Sprache der Mathematik - ist festzuhalten: F_1 ist der Basis-Offset vor der Änderung. F_2 ist der Basis-Offset nach der Änderung. Die y-Komponente wird modifizert:

$$F_{2.y} = F_{1.y} - dH \tag{6.21}$$

Diese Lösung wurde in der Prozedur `move_canvas_bottom` umgesetzt. Der Aufruf dieser Prozedur ist - wie im Quellcode zu sehen - die erste Handlung nach Ermittlung von dH und dW.

[8] Alternativ könnte man auch mittels dH den Translation-Offset T ändern. Dieser Gedanke wurde aber nicht weiter verfolgt.

6.5.4 Betriebsarten der Ansicht

Im Zusammenhang mit Größenveränderungen des Rollfensters kann sich das Bild (oder die Ansicht) auf verschiedene Weisen verhalten. Es stehen diese Modi zur Diskussion:

1. Modus 1: `MODE_1_EXPOSE_CANVAS`. Vergrößerungsfaktor konstant. Keinerlei Verschiebung des Bildes. Es wird lediglich mehr oder weniger Leinwand sichtbar gemacht.

2. Modus 2: `MODE_2_KEEP_CENTER`. Vergrößerungsfaktor konstant. Die Mitte M des Bildes bleibt in der Mitte des Rollfensters (Zentrierung). Um M herum wird mehr oder weniger Leinwand sichtbar gemacht. Diesen Modus benutzen z.B. die quelloffenen[9] Werkzeuge `GIMP` und `KiCad`.

3. Modus 3: `MODE_3_ZOOM_FIT`. Der gegenwärtig sichtbare Bereich der Leinwand wird fortlaufend in das sich ändernde Rollfenster eingefaßt. Das impliziert eine fortlaufende Skalierung. Dieser Modus wird z.B. von dem nicht-quelloffenen Werkzeug `EAGLE`® des Herstellers `Autodesk`® verwendet.

Mit der obigen Reihenfolge dieser Modi nimmt die Komplexität der zugehörigen Mechanismen zu. In den folgenden Unterabschnitten sollen diese genauer erklärt werden.

Die Betriebsarten sind im Quellcode in Abschnitt 9.10 definiert. Ebenso ist dort die Standard-Betriebsart durch die globale Varible `zoom_mode` eingestellt. An dieser Stelle kann der Anwender des Demoprogrammes die gewünschte Betriebsart festlegen.

[9] engl. open sourced

6.5.4.1 Modus 1 (MODE_1_EXPOSE_CANVAS)

Dieser Modus ist am einfachsten umzusetzen. Die Anschläge (L und U) und Seitengrößen (P) der Rollbalken werden **nicht** geändert, weil es sich hier nicht um eine Vergrößerung oder Verkleinerung handelt. Der Vergrößerungsfaktor ändert sich also **nicht**. Die Positionen (V) der Rollbalken bleiben ebenfalls unverändert. Wie sich das Bild in diesem Modus verhält, soll mit den beiden folgenden Abbildungen anschaulich gemacht werden. In Bild 6.10 ist das Objekt - ein Dreieck - zu sehen.

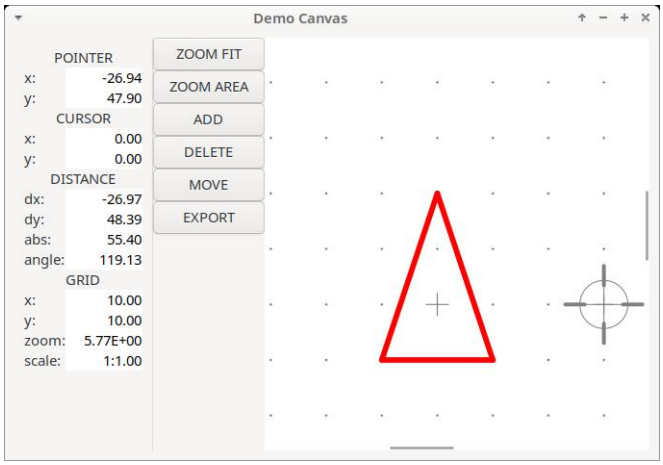

Abbildung 6.10: Vorher

Nun vergrößert der Bediener das Hauptfenster durch Verschieben der Ränder oder durch Maximieren. Das Ergebnis ist in Abbildung 6.11 zu sehen. Es wird von der Leinwand mehr Fläche nach rechts-oben freigelegt was nun auch andere Objekte, wie ein Rechteck und einen Kreis erkennen läßt. Der Vorgang kehrt sich um, wenn das Hauptfenster verkleinert wird. Der Vergrößerungsfaktor S (5,77) ändert sich nicht.

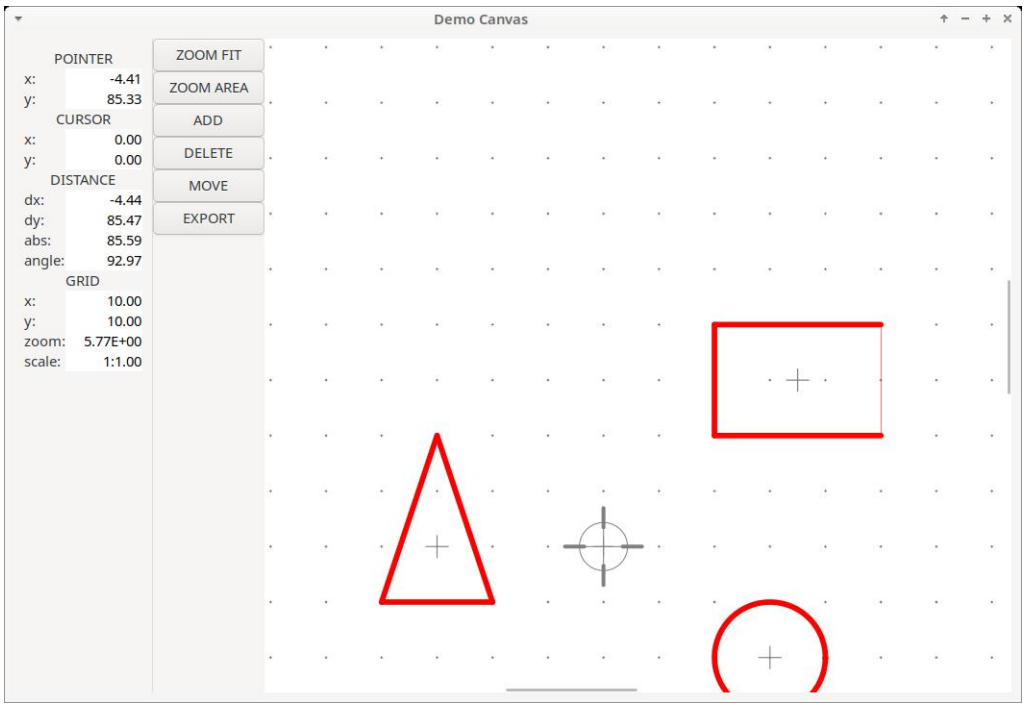

Abbildung 6.11: Nachher

6.5.4.2 Modus 2 (MODE_2_KEEP_CENTER)

Das Verhalten der Ansicht in dieser Betriebsart ist aus den beiden folgenden Abbildungen ersichtlich. Bild 6.12 zeigt als darzustellendes Objekt - ein Dreieck - welches sich in der Mitte des Bildes befindet.

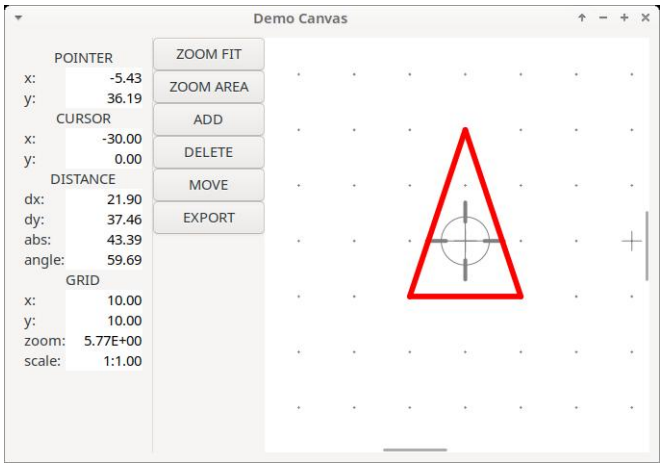

Abbildung 6.12: Vorher

Nachdem der Bediener die Dimensionen des Hauptfensters verändert hat, befindet sich das Dreieck weiterhin in der Mitte des sichtbaren Bereiches. Zusätzlich ist mehr Leinwandfläche freigelegt worden, was andere Objekte, wie das Rechteck und dem Kreis erkennen läßt.

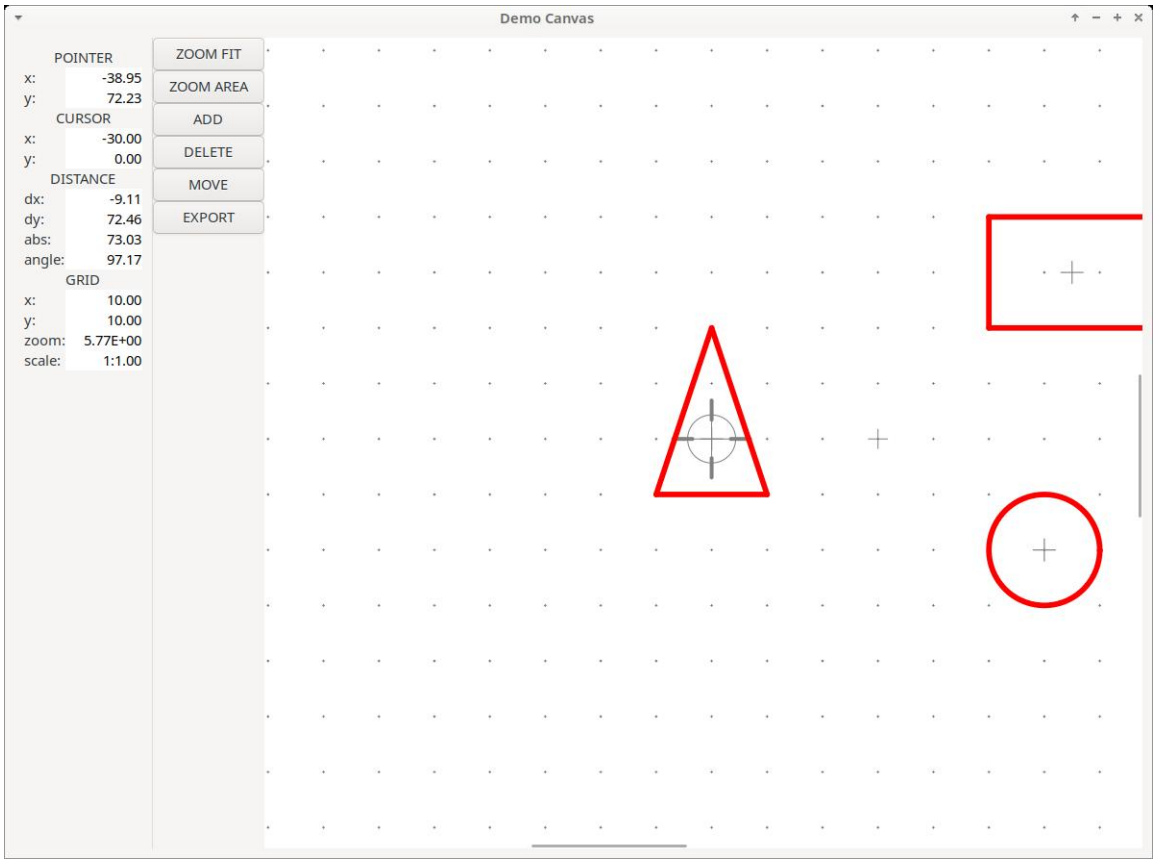

Abbildung 6.13: Nachher

Weil auch in diesem Modus der Vergrößerungsfaktor (5,77) sich nicht ändert, bleiben die Anschläge (L und U) und Seitengrößen (P) der Rollbalken unverändert. Die Position (V) wird ebenfalls nicht geändert.

Abbildung 6.14 soll den im Hintergrund ablaufenden Mechanismus verständlich machen. Das Rollfenster R1 vor der Änderung ist in durchgezogener Linie dargestellt. R2 mit gestrichelter Linie ist das Rollfenster nach der Änderung. Die Breitenveränderung dW und Höhenveränderung dH sind bereits bekannt. In durchgezogener Linie ist das Bild D1 vor der Änderung gezeichnet. Das Bild nach der Änderung, also D2, ist mit gestrichelter Linie zu sehen.

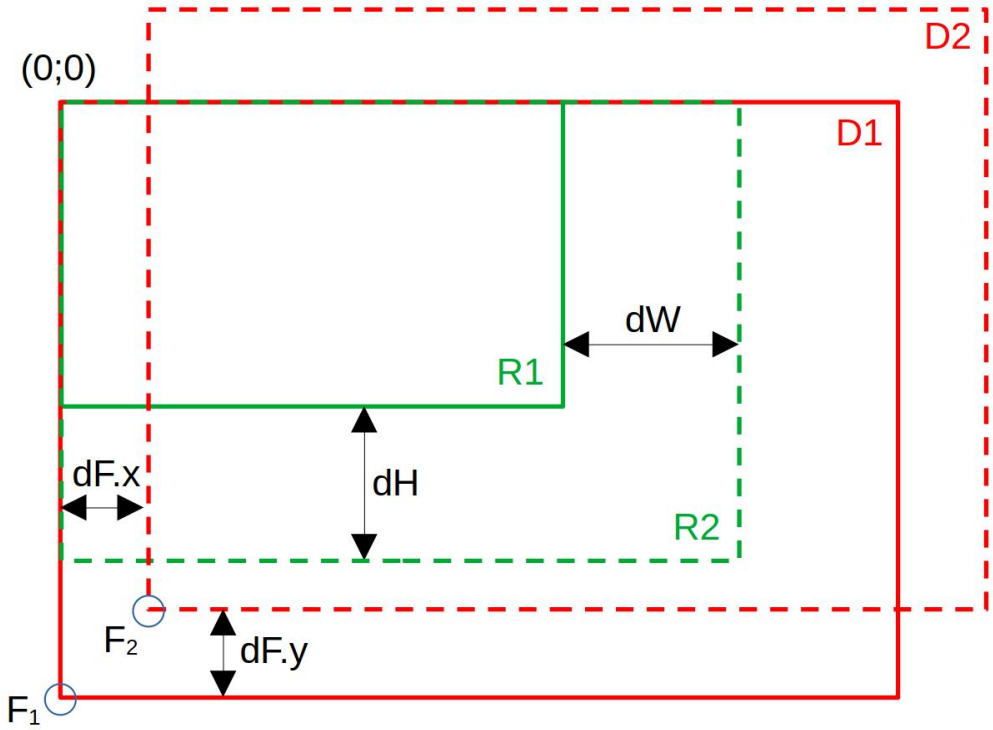

Abbildung 6.14: Verschiebung des Bildes in Modus 2

In diesem Modus wird der Basis-Offset F modifizert (Siehe Abschnitt 3.7). Der Index 1 kennzeichnet F VOR der Änderung, Index 2 markiert den neuen Basis-Offset NACH der Änderung:

1. Zur x-Komponente von F wird die Hälfte der Breitenveränderung dW addiert:

$$F_{2.x} = F_{1.x} + 0,5 \cdot dW \tag{6.22}$$

   Ist dW positiv (Breite hat zugenommen), wird das Bild um 1/2 dW nach rechts verschoben. Ist dW negativ (Breite wurde verringert), rückt das Bild um 1/2 dW nach links. Der Faktor 0,5 bewirkt die horizontale Zentrierung.

2. Zur y-Komponente von F wird die Hälfte der Höhenveränderung dH addiert:

$$F_{2.y} = F_{1.y} + 0,5 \cdot dH \tag{6.23}$$

   Ist dH positiv (Höhe hat zugenommen), wird das Bild um 1/2 dH nach oben verschoben. Ist dH negativ (Höhe wurde verringert), rückt das Bild um 1/2 dH nach unten. Der Faktor 0,5 bewirkt die vertikale Zentrierung.

   Beachte: Der y-Anteil von F ist negativ. Siehe Abschnitt 5.2.1.

Die praktische Umsetzung dieses Verfahren ist in Unterprozedur `move_center` zu finden.

6.5.4.3 Modus 3 (MODE_3_ZOOM_FIT)

Die Realisierung dieses Modus hat mir die meisten gedanklichen Klimmzüge bereitet. In diesem Modus soll das im Rollfenster sichtbare Bild mit Vergrößerung oder Verkleinerung des Hauptfensters ebenfalls skalieren. Die Abbildung 6.15 zeigt wieder das Dreieck in der Bildmitte. Der Vergrößerungsfaktor beträgt 4,81.

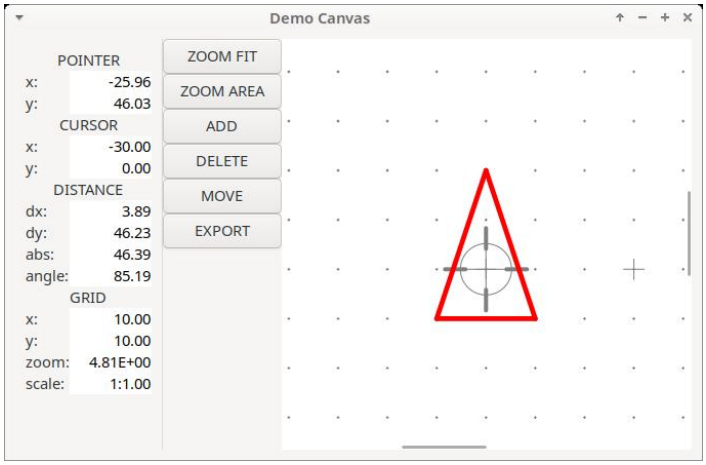

Abbildung 6.15: Vorher

Nun wird das Hauptfenster vergrößert, wie in Abbildung 6.16 zu sehen ist. Der gesamte bisher sichtbare Bereich bleibt sichtbar, wird aber **vergrößert** dargestellt. Der Vergrößerungsfaktor hat von 4,8 nach 9,7 zugenommen. Somit müssen auch die Anschläge (L und U) der Rollbalken angepaßt werden.

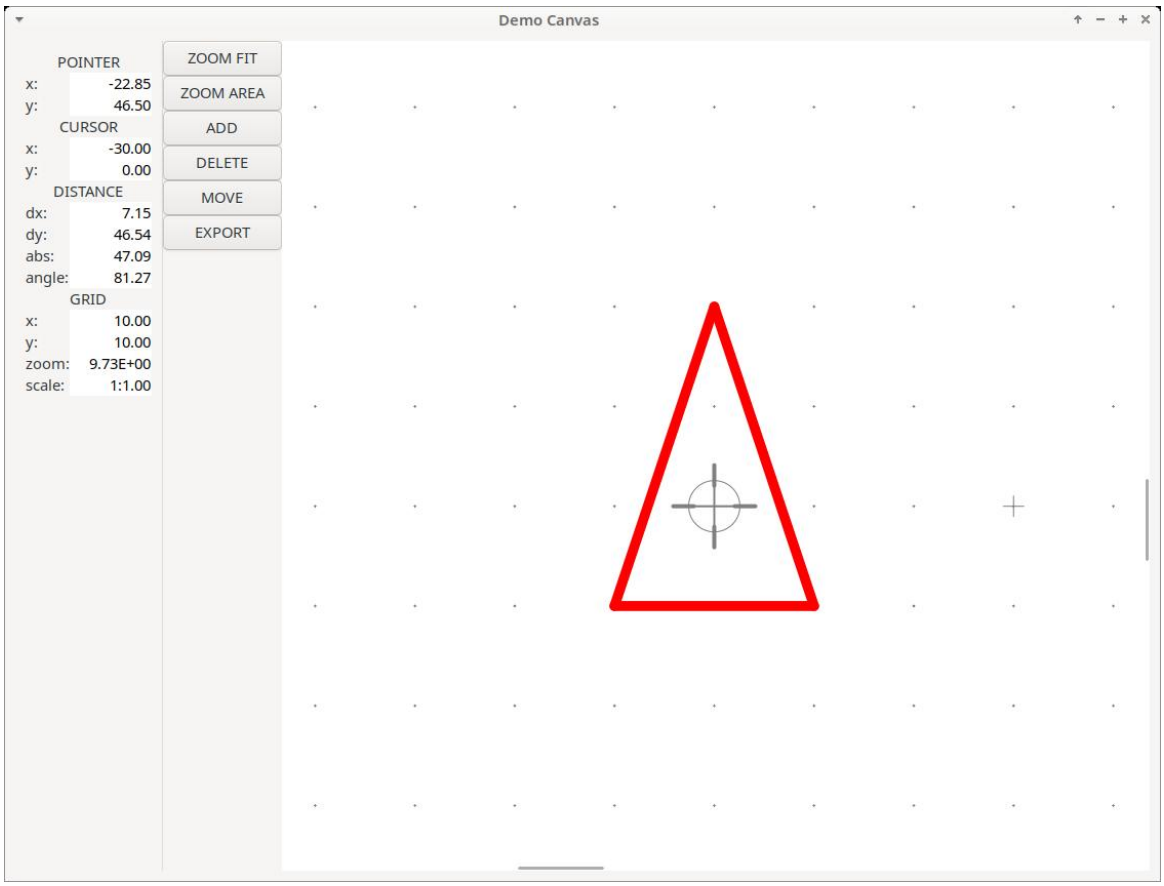

Abbildung 6.16: Nachher

Es gibt zwei Methoden, diesen eleganten Modus zu realisieren:

1. Methode 1:

   Achtung: Der an einer funktionierenen Lösung interessierte Leser kann die Ausführungen zu Methode 1 überspringen.

   Die Idee ist, mit der Bildmitte als Zentrum, das Bild zu skalieren. Der dafür nötige Vergrößerungsfaktor wird aus der Veränderung der Abmessungen des Rollfensters ermittelt. Diese Methode wurde zuerst versucht, erwies sich aber letztlich als zu komplex und vermutlich falsch. Außerdem ist es nicht gelungen, die Skalierung sauber mit den Abmessungen des Rollfensters zu synchronisieren. Der Vollständigkeit halber sollen die Überlegungen aber dennoch dokumentiert werden. Es handelt sich hier also um einen **unvollständigen** Mechanismus ! Der Quellcode für diese Methode wurde später mit dem Commit `028ae8a228cb7145182dab5069a2fe9ca72edefd` gelöscht, kann also bei Bedarf wiederhergestellt werden [8] [9].

2. Methode 2: Der Grundgedanke ist ein fortlaufendes Einpassen eines bestimmten sichtbaren Bereiches in das sich ändernde Rollfenster. Diese Methode erwies sich als die einfachste, eleganteste und zuverlässigste Lösung. Obwohl diese Methode die anspruchsvollste ist, sieht die Realisierung letztendlich überhaupt nicht kompliziert aus.

Beschreibung der Methode 1

Das Maß der Änderung der Abmessungen des Rollfensters entscheidet, ob das Bild vergrößert oder verkleinert wird. Es stellt sich dabei die zentrale Frage, wie die Breiten- und Höhenveränderung in die Berechnung des Vergrößerungsfaktors eingeht.

Der Ablauf ist im Wesentlichen folgender:

1. Prozedur `move_center` aufrufen um das Bild mittig im Rollfenster zu zentrieren. Siehe Abschnitt 6.5.4.2.
2. Das Bild entsprechend der Veränderung der Abmessungen des Rollfensters skalieren. Die folgenden Schritte sollen dieser Prozedur genauer beschreiben.
3. S_1 ist der Vergrößerungsfaktor vor der Skalierung.
4. Das Zentrum M der Skalierung ist die geometrische Mitte des sichtbaren Bereiches.
5. C1 und C2 sind die Ecken des Begrenzungsrechteckes vor und nach der Skalierung. Es wird zunächst C1 mittels der Funktion `get_bounding_box_corners` ermittelt.
6. Nun wird das Verhältnis S_{2W} aus neuer Breite W_2 des Rollfensters zur initialen Breite W_1 des Rollfensters berechnet:

$$S_{2W} = \frac{W_2}{W_1} \tag{6.24}$$

   Die initiale Breite W_1 des Rollfensters ist hier keine Konstante. Sie wurde erstmalig beim Systemstart gesetzt, wie in Abschnitt 3.8 ausgeführt wurde. Sie dient in diesem Fall als Bezugswert um das Verhältnis von neuer Breite zu alter Breite zu ermitteln. Bevor das Rollfenster seine Größe ändert, muß also W_1 mit der gegenwärtigen Breite des Rollfensters geladen sein. Das kann z.B. durch Auslesen der Variable `swin_size` gemacht werden.
7. Nach gleichem Schema wird das Verhältnis aus neuer Höhe H_2 des Rollfensters zu initialer Höhe H_1 berechnet:

$$S_{2H} = \frac{H_2}{H_1} \tag{6.25}$$

8. Nun wird von S_{2W} und S_{2H} der kleinere von beiden Werten ausgewählt und mit dem initialen Vergrößerungsfaktor S_i multipliziert. Das Ergebnis ist der neue Vergrößerungsfaktor S_2. S_i ist auch hier ein Bezugswert, der vor der Änderung des Rollfensters mit dem gegenwärtigen Vergrößerungsfaktor geladen sein muß. Das kann sehr einfach durch Auslesen der globalen Variable `S` geschehen.

$$S_2 = S_i \cdot min(S_{2W}, S_{2H}) \tag{6.26}$$

9. S_1, S_2 und M werden der Prozedur `set_translation_for_zoom` übergeben und somit der neue Translation-Offset T berechnet.
10. Der globale Vergrößerungsfaktor S ist dann mit S_2 zu laden.
11. Die Ecken C2 des Begrenzungsrechtecks mit Funktion `get_bounding_box_corners` ermitteln.
12. Die Anschläge der Rollbalken mit Prozedur `update_scrollbar_limits` und den Argumenten C1 und C2 setzen.
13. Konfiguration der Rollbalken sichern mit Prozedur `backup_scrollbar_settings`.

Beschreibung der Methode 2

Der Grundgedanke ist, eine bestimmte sichtbare Fläche A_v fortlaufend in das Rollfenster einzufassen. Die Prozedur `zoom_to_fit` bietet sich an, um eine konstante vorgegebene Fläche in ein sich änderndes Rollfenster einzufassen. Das Ergebnis dieser Überlegungen ist in der Unterprozedur `zoom_visible_area` zu finden.

Der Ablauf ist folgender:

1. C1 und C2 sind die Ecken des Begrenzungsrechteckes vor und nach dem Einpassen. Es wird zunächst C1 mittels der Funktion `get_bounding_box_corners` ermittelt.
2. Translation-Offset T zurücksetzen auf (0;0).
3. Den letzten bekannten sichtbaren Bereich A_v an die Prozedur `zoom_to_fit` übergeben. A_v muß zu diesem Zeitpunkt aktuell sein ! Dazu folgen später Details.
4. Die Ecken C2 des Begrenzungsrechtecks mittels der Funktion `get_bounding_box_corners` ermitteln.
5. Die Anschläge der Rollbalken mit Prozedur `update_scrollbar_limits` und den Argumenten C1 und C2 aktualisieren.
6. Konfiguration der Rollbalken sichern mit Prozedur `backup_scrollbar_settings`.

Details sind der Unterprozedur `zoom_visible_area` zu entnehmen.

Anmerkungen zur Aktualisierung des sichtbaren Bereiches

Der sichtbare Bereich A_v muß immer aktualisiert werden,

- nachdem ein Rollbalken verschoben und losgelassen wurde,
- nachdem mittels Mausrad das Bild verschoben wurde,
- nachdem der Cursor bewegt wurde,
- nachdem an der Cursor- oder Mausposition vergrößert oder verkleinert wurde,
- nach dem Einpassen aller Objekte des Modelles,
- nach dem Einpassen eines ausgewählten Bereiches.

Es handelt sich also um alle Operationen, die in Abschnitt 8.1, beschrieben sind. Dafür ist die Prozedur `backup_visible_area` vorgesehen, welche eine Fläche entgegennimmt und in globalen Variable `last_visible_area` speichert. Diese Variable entspricht dem oben verwendeten Formelzeichen A_v.

6.6 Zeichnen auf der Leinwand

Das Zeichnen auf der Leinwand bedeutet, daß alle Objekte, der Zeichnungsrahmen, das Raster und der Cursor auf die Leinwand gezeichnet werden. Mittels des sichtbaren Bereiches und des gegenwärtigen Vergrößerungsfaktors kann dieser Vorgang optimiert werden, in dem nur das gezeichnet wird, was sichtbar und hinreichend groß ist. Doch dazu kommen wir später. Für jemanden, der sich das erste mal mit der Thematik *Leinwand* beschäftigt, ist folgende Erkenntnis elementar: Das (Neu)Zeichnen muß immer stattfinden nachdem

1. ein Rollbalken verschoben wurde.
2. die Größe des Rollfenster sich geändert hat. Das passiert indirekt wenn der Bediener die Größe des Hauptfensters ändert.
3. skaliert wurde, d.h. eine Vergrößerung oder Verkleinerung stattfand.
4. Objekte vom Bediener verschoben, entfernt oder hinzugefügt wurden.
5. alle Objekte eingefaßt wurden (zoom-to-fit).

Diese Ereignisse erfordern den sogenannten *expliziten Aufruf* des Neuzeichnens. Dazu siehe Abschnitt 6.6.10.

Wir haben es also immer mit einem vollständigen Aufbau des Bildes zu tun. Je nachdem wie viele Dinge auf der Leinwand darzustellen sind und wie effizient programmiert wurde, gestaltet sich der Bildaufbau fließend oder stockend. Es geht hier nicht nur um das Aufrufen von primiven Zeichenoperationen, sondern auch um zahlreiche Rechenoperationen die möglichst zeitsparend sein sollten.

Im Demoprogramm ist für das Zeichen der Objekte die Prozedur `cb_draw` zuständig. Siehe Quellcode in Abschnitt 9.8. Wann und wie diese Prozedur aufgerufen wird, soll in Abschnitt 6.6.10 erklärt werden.

Der Prozess des (Neu)Zeichnens besteht im Wesentlichen aus diesen Schritten:

1. Berechnen des sichtbaren Bereiches. Siehe Abschnitt 3.9 und 6.6.1.
2. Zeichnen des Rasters - sofern ein Raster gewünscht ist - innerhalb des sichtbaren Bereiches. Siehe Quellcode in Abschnitt 9.4.
3. Zeichnen des Ursprungs auf der realen Modellposition (0;0). Weil es sich hier lediglich um zwei kurze und dünne Linien handelt, kann auf die Prüfung, ob diese sich im sichtbaren Bereich befinden, verzichtet werden. Siehe Quellcode in Abschnitt 9.17.
4. Zeichnen des Cursors auf seiner Modellposition. Weil sich der Cursor meistens im sichtbaren Bereich befindet, kann auf die Prüfung, ob er sich dort befindet ebenfalls verzichtet werden. Siehe Quellcode in Abschnitt 9.16.
5. Falls gerade ein Bereich zum Vergrößern ausgewählt wird, muß dessen Umrahmung mit Prozedur `draw_zoom_area` gezeichnet werden. Siehe Quellcode in Abschnitt 9.12.
6. Zeichnen des Zeichnungsrahmens (ZR) und Dokumentationsfeldes (DF). Siehe Quellcode in Abschnitt 9.2.
7. Datenbank des Modells durchforsten[10] und komplexe Objekte - oder Teile davon - zeichnen wenn diese sich innerhalb des sichtbaren Bereiches befinden. Bei sehr großen Datenbanken mit vielen Objekten wird das zu einer äußerst zeitraubenden Prozedur. Dazu

[10] engl. to parse

kommt, daß die einzelnen primitiven Objekte des komplexen Objektes geprüft werden müssen, ob sie sich im sichtbaren Bereich befinden. Siehe Quellcode in Abschnitt 9.3.

In den folgenden Unterabschnitten wollen wir diese Schritte genauer betrachten.

6.6.1 Berechnung des sichtbaren Bereiches

Die Bedeutung des sichtbaren Bereiches wurde bereits in Abschnitt 3.9 dargelegt. Seine Berechnung basiert auf

- der Allokation W des Rollfensters,
- dem Basis-Offest F,
- dem Translation-Offset T,
- und der Einstellung V der Rollbalken.

Es gelten die zum Zeitpunkt des Darstellens auf der Leinwand aktuellen Werte der genannten Größen. Im Demoprogramm ist dafür die Funktion `get_visible_area` vorgesehen. Siehe Quellcode in Abschnitt 9.13.

X-Achse

Der sichtbare Bereich auf der Leinwand entlang der X-Achse - also in der Horizontalen - beginnt am Einstellwert V des horizontalen Rollbalkens. Diesen Punkt nennen wir h_s [11]. Er wird in logischen Pixeln angegeben. Seine Einheit ist also `lp`.

$$h_s = V \tag{6.27}$$

Die Breite des sichtbaren Bereiches h_l [12] entspricht der Breite des Rollfensters. Diesen Wert entnehmen wir der Allokation W des Rollfensters. Die Allokation wird in Gerät-Bildpunkten ausgedrückt. Die Einheit ist also `dp`. Es gilt:

$$h_l = W_{width} \tag{6.28}$$

Die unspektakuläre Beziehung 6.28 impliziert eine Typumwandlung[13] von ganzzahligen Bildpunkten nach logischen Pixeln. Siehe Abschnitte 2.2 und 2.3.

Für das Ende h_e des sichtbaren Bereiches auf der Leinwand gilt:

$$h_e = h_s + h_l \tag{6.29}$$

Y-Achse

Der sichtbare Bereich in vertikaler Richtung entlang der Y-Achse beginnt auf der Leinwand am Einstellwert V des vertikalen Rollbalkens. Diesen Punkt nennen wir v_s. Es handelt sich um logische Pixel. Seine Einheit ist also `lp`.

$$v_s = V \tag{6.30}$$

Die Breite des sichtbaren Bereiches v_l entspricht der Höhe des Rollfensters. Diesen Wert entnehmen wir der Allokation W des Rollfensters. Es gilt:

$$v_l = W_{height} \tag{6.31}$$

Auch die Gleichung 6.31 impliziert eine Typumwandlung (siehe Bemerkungen zur X-Achse oben).

[11] Der Index s steht für Start.
[12] Der Index l steht für Länge.
[13] engl. type casting

Für das Ende v_e des sichtbaren Bereiches auf der Leinwand gilt:

$$v_e = v_s + v_l \tag{6.32}$$

Wir kennen jetzt also die Position und Abmessungen des sichtbaren Bereiches auf der Leinwand. Aus diesen Werten werden nun die Ecken des sichtbaren Bereiches - ausgedrückt in Modellkoordinaten - errechnet.

Ecken des sichtbaren Bereiches

Mittels h_s, h_e, v_s und v_e sind nun die Ecken des sichtbaren Bereiches A auf der Leinwand bestimmbar. Der tiefgestellte Buchstabe c im Folgenden bedeutet, daß es sich um Leinwandkoordinaten handelt. Mit dem tiefgestellten Buchstaben m werden Modellkoordinaten bezeichnet:

- oben links TL_c bei (h_s;v_s)
- oben rechts TR_c bei (h_e;v_s)
- unten links BL_c bei (h_s;v_e)
- unten rechts BR_c bei (h_e;v_e)

TL_c, TR_c, BL_c und BR_c müssen nun in reale Modellkoordinaten TL_m, TR_m, BL_m und BR_m bei gegenwärtigem Vergrößerungsfaktor S konvertiert werden. Dazu sei auf Abschnitt 4.4 verwiesen.

Die untere linke Ecke BL_m ist gleichzeitig auch die Position des sichtbaren Bereiches A:

$$A_{pos} = BL_m \tag{6.33}$$

Breite und Höhe des sichtbaren Bereiches

Die Breite ergibt sich aus der Differenz der x-Werte. Es gilt:

$$A_{width} = TR_{m.x} - TL_{m.x} \tag{6.34}$$

Die Höhe ergibt sich aus der Differenz der y-Werte. Es gilt:

$$A_{height} = TL_{m.y} - BL_{m.y} \tag{6.35}$$

Am Ende diese Prozedur ist der sichtbare Bereich A vollständig bekannt. A_{pos} liefert x und y für die Position, A_{width} die Breite und A_{height} die Höhe.

Als Speicherort für den sichtbaren Bereich A ist eine globale Variable zu verwenden, weil auf A noch andere Prozesse zugreifen. Im Demoprogramm ist das die globale Variable `visible_area`. Siehe Quellcode in Abschitt 9.13.

6.6.2 Berechnen des Rasters

Das Raster ist für ein CAD-System eine Option. Es erleichtert das Platzieren und Ausrichten von Objekten und kann in Form von Punkten oder Linien dargestellt werden. Dafür gibt es üblicherweise innerhalb des CAD-Systems eine Einstellmöglichkeit für den Bediener. Unser Demoprogramm ist jedoch auf ein Minimum reduziert, sodaß die Konfiguration des Rasters im Quellcode vorgenommen werden muß. Dazu sei auf Abschnitt 9.4 verwiesen. Der Programmcode das Raster betreffend, ist in der Prozedur `draw_grid` zu finden. Siehe Quellcode dazu ebenfalls in Abschnitt 9.4.

Um Rechenzeit zu sparen, wird das Raster nur für den sichtbaren Bereich (siehe 6.6.1) berechnet und auch nur dort gezeichnet. Abbildung 6.17 zeigt den sichtbaren Bereich (Rechteck) und relevante Rasterpunkte innerhalb dessen mit Kreuzen markiert. Theoretische, aber überflüssige Rasterpunkte außerhalb des Bereiches werden nicht auf die Leinwand gezeichnet.

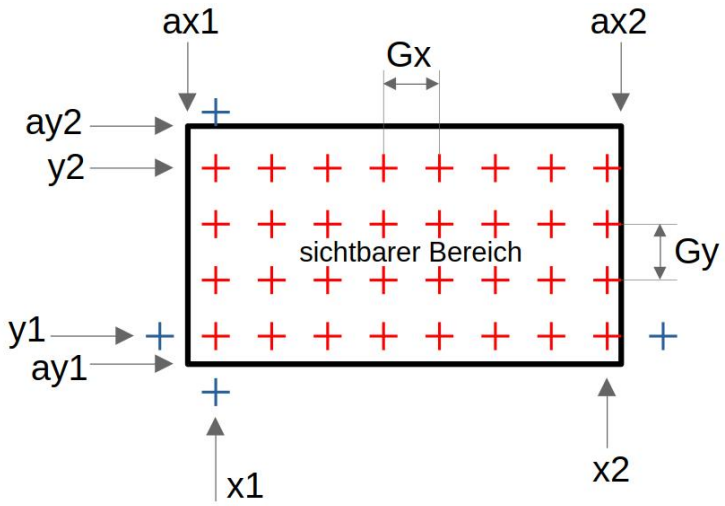

Abbildung 6.17: Berechnung des Rasters

Wie oben schon gesagt, legt der Bediener üblicherweise die Größe G des Rasters fest. Dies kann separat für die x- wie auch y-Achse gemacht werden. Meist gibt es aber nur einen Wert, der für x und y gleichermaßen gilt. Weil sich das Raster im Koordinatensystem des Modells (CS1) befindet, müssen auch hier Festkommazahlen verwendet werden. Dazu siehe Abschnitt 2.1.2.

Wir haben nun mit den Ausgangsgrößen G_x und G_y zu tun. Diese Werte entsprechen dem Abstand der Spalten[14] und Reihen[15]. Die Entfernung zwischen Spalten oder zwischen Reihen ist also immer ein ganzzahiges Vielfaches von G_x oder G_y.

Die Typdeklaration und die Standardwerte für das Raster sind im Quellcode in Abschnitt 9.4 zu finden. Die Weite des Rasters kann hier mit der Konstanten `grid_spacing_default` eingestellt werden.

[14] engl. columns
[15] engl. rows

6.6.2.1 Dichte des Rasters

Bevor die eigentliche Berechnung und das Zeichen des Rasters begonnen wird, ist zu prüfen, ob die Dichte[16] des Rasters klein genug ist. Die Dichte ist vom Vergrößerungsfaktor S abhängig. Beim Verkleinern ergibt sich irgendwann eine so hohe Dichte, daß im gesamten Bild nurnoch Rasterpunkte oder Rasterlinien erscheinen, was keinen Sinn mehr macht. Wir brauchen also den Abstand zwischen Reihen oder Spalten in Form von logischen Pixeln, um zu entscheiden, ob das Raster überhaupt berechnet und gezeichnet werden soll. Die einfachste Lösung ist, den gegenwärtigen Rasterabstand G mit dem gegenwärtigen Vergrößerungsfaktor S zu multiplizieren. Das Ergebnis ist der zu erwartende Abstand zwischen den Spalten und Reihen in logischen Pixeln:

Spaltenabstand:

$$d_c = G_x \cdot S \tag{6.36}$$

Reihenabstand:

$$d_r = G_y \cdot S \tag{6.37}$$

Ist d_c **kleiner** als der systeminterne Minimalwert $d_{c.min}$ oder ist d_r **kleiner** als $d_{r.min}$, dann wird das Raster **nicht** berechnet und auch **nicht** gezeichnet.

Mit der Funktion `get_grid_spacing` wird der Raster-Abstand erfragt. Die Funktion betrachtet Spalten- und Reihenabstand und liefert den von beiden kleineren Wert zurück. Sofern für die Größe des Rasters nur ein Wert für x und y verwendet wird, vereinfacht sich diese Prüfung natürlich.

Im Demoprogramm gibt es für Reihen- und Spaltenabstand nur einen systeminternen Minimalwert, welcher in der globalen Variablen `grid_spacing_min` gespeichert ist.

6.6.2.2 Berechnung der sichtbaren Spalten und Reihen

Das Ziel der folgenden Rechnung ist es, nur die Spalten und Reihen zu ermitteln, die sich innerhalb des sichtbaren Bereiches A befinden. Zum besseren Verständis der folgenden Überlegungen soll Abbildung 6.18 helfen.

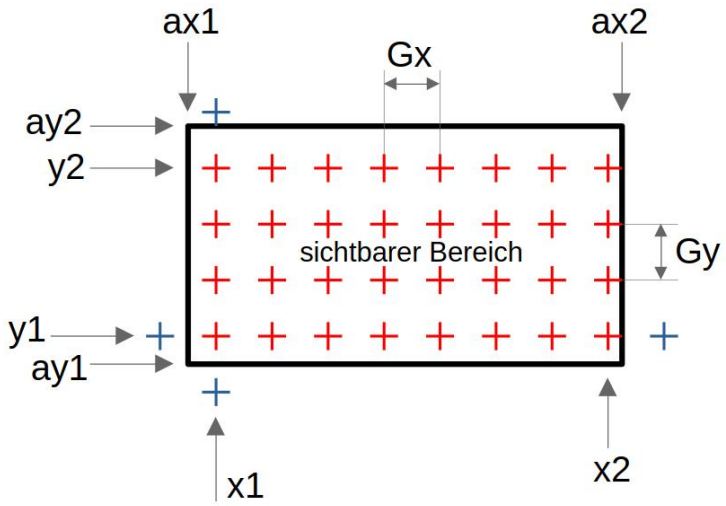

Abbildung 6.18: sichtbare Spalten und Reihen

[16] engl. density

Spalten entlang der X-Achse

Wir müssen die erste und letzte Spalte im sichtbaren Bereich bestimmen.

1. Aus dem sichtbaren Bereich A läßt sich sein Beginn und Ende ableiten. Der Beginn bei a_{x1} entspricht:
$$a_{x1} = A_{pos.x} \tag{6.38}$$
Das Ende bei a_{x2} ist die Summe aus Begin und Länge:
$$A_{x2} = A_{x1} + A_{length} \tag{6.39}$$

2. Nun soll die erste Spalte bei x_1 **nach** Beginn des sichtbaren Bereiches bestimmt werden. Die Größe C enthält die Anzahl der vollständigen Spalten zwischen dem x-Wert Null und dem Anfang des sichtbaren Bereiches bei a_{x1}:
$$C = floor\frac{a_{x1}}{G_x} \tag{6.40}$$
Das englische Wort `floor` bedeutet wörtlich übersetzt `Boden`, also das Abrunden auf die nächst kleinere ganze Zahl. Auch wenn a_{x1} negativ ist, funktioniert diese Methode. C wird dann aber negativ, was einer "negativen" Anzahl von Spalten entspricht.
Nun wird C mit G_x multipliziert um die letzte Spalte **vor** Beginn des sichtbaren Bereiches zu erhalten:
$$L = C \cdot G_x \tag{6.41}$$
Weil wir aber an der ersten Spalte **nach** dem Beginn des sichtbaren Bereiches interessiert sind, muß abschließend noch einmal G_x addiert werden:
$$x_1 = L + G_x \tag{6.42}$$
Die obigen Gleichungen lassen sich zusammenfassen:
$$x_1 = (G_x \cdot floor\frac{a_{x1}}{G_x}) + G_x \tag{6.43}$$
Nach Ausklammern von G_x erhalten wir abschließend:
$$x_1 = (1 + floor\frac{a_{x1}}{G_x}) \cdot G_x \tag{6.44}$$
Entlang der X-Achse befindet sich also bei x_1 die erste Spalte.

3. Nun wird die letzte Spalte bei x_2 **vor** dem Ende des sichtbaren Bereiches bestimmt. Die Größe C enthält die Anzahl der vollständigen Spalten zwischen x-Wert Null und dem Ende des sichtbaren Bereiches bei a_{x2}:
$$C = floor\frac{a_{x2}}{G_x} \tag{6.45}$$
Nun wird C mit G_x multipliziert um die letzte Spalte **vor** Ende des sichtbaren Bereiches zu erhalten:
$$L = C \cdot G_x \tag{6.46}$$
Die obigen Gleichungen lassen sich zusammenfassen:
$$x_2 = (floor\frac{a_{x2}}{G_x}) \cdot G_x \tag{6.47}$$
Entlang der X-Achse befindet sich also bei x_2 die letzte Spalte.

Die programmtechnische Umsetzung ist in der Prozedur `compute_first_and_last_column` zu finden. Siehe Quellcode in Abschnitt 9.4.

Reihen entlang der Y-Achse

Hier soll die erste und letzte Reihe (horizontal) im sichtbaren Bereich bestimmt werden.

1. Aus dem sichtbaren Bereich A läßt sich sein Beginn und Ende ableiten. Der Beginn bei a_{y1} entspricht schlicht der Position von A:

$$a_{y1} = A_{pos.y} \tag{6.48}$$

   Das Ende bei a_{y2} ist die Summe aus Beginn und Höhe:

$$a_{y2} = a_{y1} + A_{height} \tag{6.49}$$

2. Die weitere Berechnung der ersten und letzten Reihe entlang der Y-Achse erfolgt analog dem oben beschriebenen Verfahren für die X-Achse. Lediglich der Index x ist gegen y zu tauschen.

Die programmtechnische Umsetzung ist in der Prozedur `compute_first_and_last_row` im Quellcode in Abschitt 9.4 zu finden.

6.6.3 Zeichnen des Rasters

Dem Zeichnen des Rasters auf der Leinwand geht die Berechnung der Reihen und Spalten laut dem vorherigen Abschnitt voraus. Das Raster kann daraufhin dargestellt werden in Form von

- kleinen Punkten
- dünnen horizontalen und vertikalen Linien

6.6.3.1 Punkte

In Abbildung 6.19 ist die Leinwand mit Rasterpunkten zu sehen.

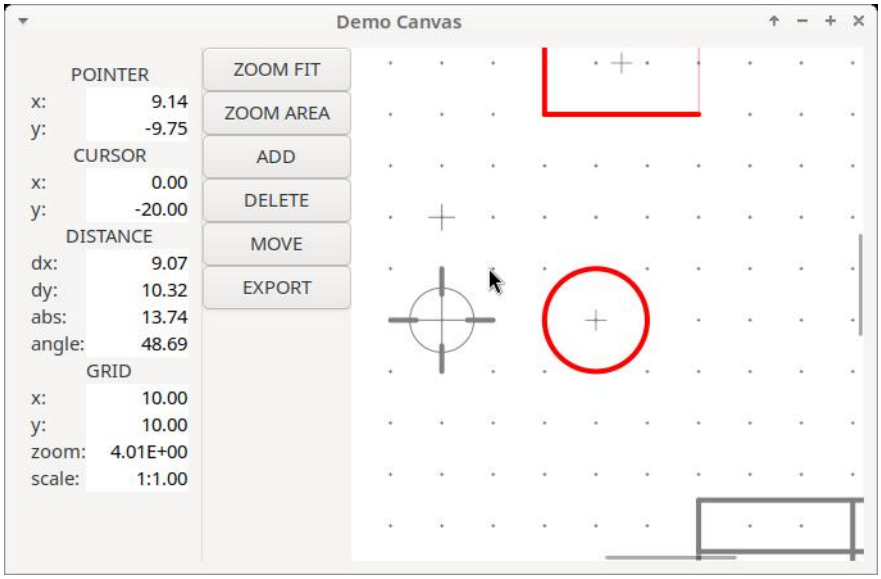

Abbildung 6.19: Raster in Form von Punkten

Wenn das Raster in Form von kleinen Punkten dargestellt werden soll, kann man sich eine Matrix mit N Spalten und M Reihen wie in Abbildung 6.20 gezeigt vorstellen.

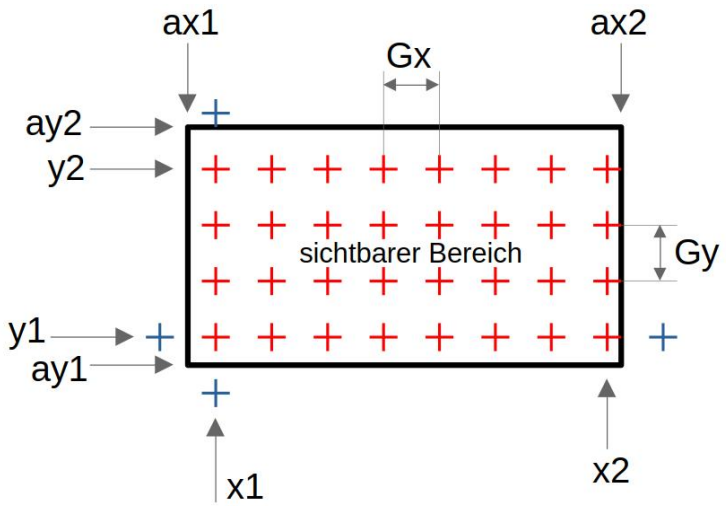

Abbildung 6.20: Punkte im sichtbaren Bereich

Die erste Spalte (links) befindet sich bei x_1 und die letzte Spalte (rechts) bei x_2. Die erste Reihe (unten) befindet sich bei y_1 und die letzte Reihe (oben) bei y_2. Dort wo sich eine Spalte

und eine Reihe kreuzt, ist also ein Punkt zu zeichnen. Die Position eines Rasterpunktes wird folglich in Modellkoordinaten (CS1) berechnet und angegeben.

Der Punkt selbst besteht aus zwei sehr kurzen dünnen Linien, die sich kreuzen und zusammen als ein Punkt erscheinen. Das finale Zeichnen der beiden Linien geschieht in Leinwandkoordinaten (CS2). Die Länge der beiden Linien und deren Linienbreite sind also als logische Pixel (`lp`) spezifiziert. Dies bewirkt den Effekt, daß unabhägig vom Vergrößerungsfaktor S der Punkt immmer mit gleicher Größe erscheint. Im Quellcode in Abschitt 9.4 ist die globale Konstante `grid_width_lines` für die Breite dieser Linien zu finden. Die Länge der Linien wird mit der Konstanten `grid_cross_arm_length` festgelegt.

Der Algorithmus zum Zeichnen der Punkte ist denkbar einfach: Eine äußere Schleife durchläuft die Reihen und eine innere Schleife die Spalten[17]. Pro Punkt werden die zwei kurzen dünnen Linien gezeichnet.

In der Prozedur `draw_dots` ist dieser Algorithmus realisert worden.

Anmerkungen zu GTK3/Cairo

Die beiden Linien des Rasterpunktes werden mit den *GTK3*-Prozeduren `move_to` und `line_to` gezeichnet. Alle Linien haben gleiche Breite und gleiche Farbe. Um die Darstellung der Rasterpunkte möglichst zügig zu vollziehen, braucht der Befehl `stroke` nur einmalig am Schluß der Routine ausgeführt werden (Siehe auch Abschnitt 6.6.11).

[17] Von Spalte zu Spalte wird G_x addiert. Von Reihe zu Reihe wird G_y addiert. Weil wir hier mit Festkommazahlen arbeiten, besteht - im Gegensatz zu Fließkommazahlen - nicht die Gefahr von sich aufaddierenden Rundungsfehlern. Siehe auch Abschnitt 2.1.2.

6.6.3.2 Linien

In Abbildung 6.21 ist die Leinwand mit Rasterlinien zu sehen.

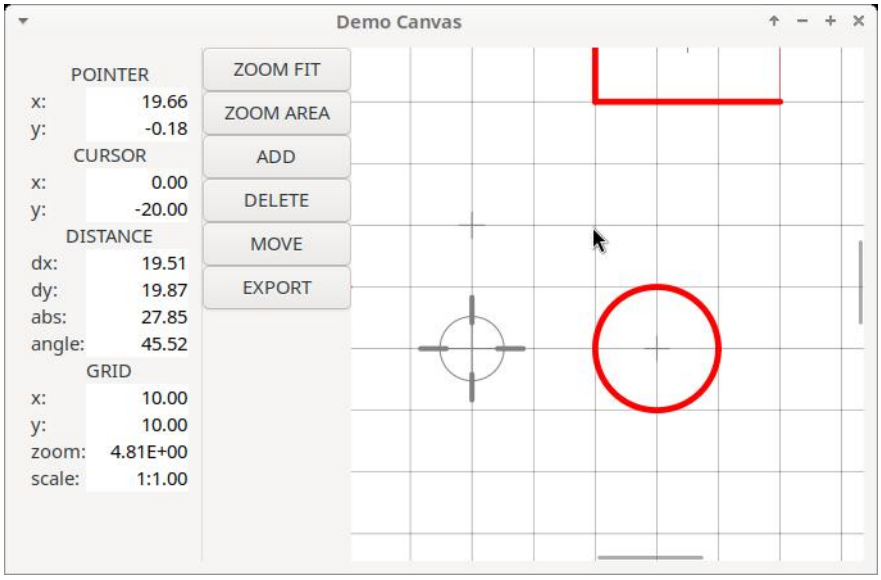

Abbildung 6.21: Raster in Form von Linien

Betrachten wir Abbildung 6.22.

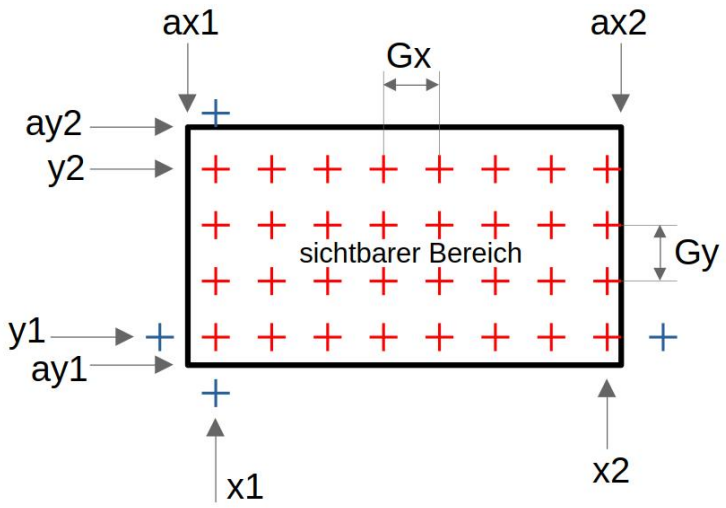

Abbildung 6.22: Linien im sichtbaren Bereich

Soll das Raster durch dünne Linien angezeigt werden, dann gilt für die **vertikalen** Linien: Die erste Linie befindet sich in x-Richtung bei x_1 und die letzte bei x_2. Alle Linien beginnen in y-Richtung bei ay_1 und enden bei ay_2, gehen also über den gesamten sichtbaren Bereich.

Für die **horizontalen** Linien gilt: Die erste Linie befindet sich in y-Richtung bei bei y_1 und die letzte bei y_2. Alle Linien beginnen in x-Richtung bei ax_1 und enden bei ax_2.

Wie beim Zeichnen der Rasterpunkte (siehe 6.6.3.1) ist auch hier das finale Zeichnen der Linien eine primitive Zeichenoperation in der Domäne der Leinwandkoordinaten (CS2). Die Linienbreite wird also in logischen Pixeln (`1p`) spezifiziert. Das bewirkt eine konstante Linienbreite, unabhägig vom Vergrößerungsfaktor S.

Die programmtechnische Umsetzung gestaltet sich einfach: Mittels zwei nacheinander zu durchlaufender Schleifen werden zuerst die vertikalen Linien (Spalten) und danach die hori-

zontalen Linien (Reihen) gezeichnet[18].

Im Quellcode in Abschitt 9.4 ist die globale Konstante `grid_width_lines` für die Breite der Linien zu finden. In der Prozedur `draw_lines` ist der Algorithmus realisert worden.

Anmerkungen zu GTK3/Cairo

Die Linien werden mit den *GTK3*-Prozeduren `move_to` und `line_to` gezeichnet. Alle Linien haben gleiche Breite und gleiche Farbe. Um die Darstellung der Linien möglichst zügig zu vollziehen, braucht der Befehl `stroke` nur einmalig am Schluß der Routine ausgeführt werden (Siehe auch Abschnitt 6.6.11).

[18]Von Spalte zu Spalte wird G_x addiert. Von Reihe zu Reihe wird G_y addiert. Weil wir hier mit Festkommazahlen arbeiten, besteht - im Gegensatz zu Fließkommazahlen - nicht die Gefahr von sich aufaddierenden Rundungsfehlern. Siehe auch Abschnitt 2.1.2.

6.6.4 Zeichnen des Ursprunges

Der Zeichnungsursprung ist dort, wo sich die Modellkoordinate (0;0) befindet. Um diesen Punkt leicht zu finden, wird er durch ein kleines Kreuz markiert wie in Abbildung 6.23 zu sehen ist. Im Bild zeigt ein Pfeil auf den Ursprung. Links davon, auf Position (-10;0), steht der Cursor, welcher als dem Fadenkreuz zu erkennen ist.

Der Ursprung muß als Kreuz - ähnlich wie das Raster - unabhägig vom Vergrößerungsfaktor immer mit gleicher Größe und Linienbreite auf die Leinwand gemalt werden. Darum ist er in der Domäne der Leinwandkoordinaten (CS2) in logischen Pixeln (`lp`) zu zeichnen. Im Quellcode des Demoprogrammes gibt es die Konstante `origin_size` für die Länge der Speichen des Kreuzes und die Konstante `origin_linewidth` für die Breite der Lininen. Dazu siehe Abschnitt 9.17.

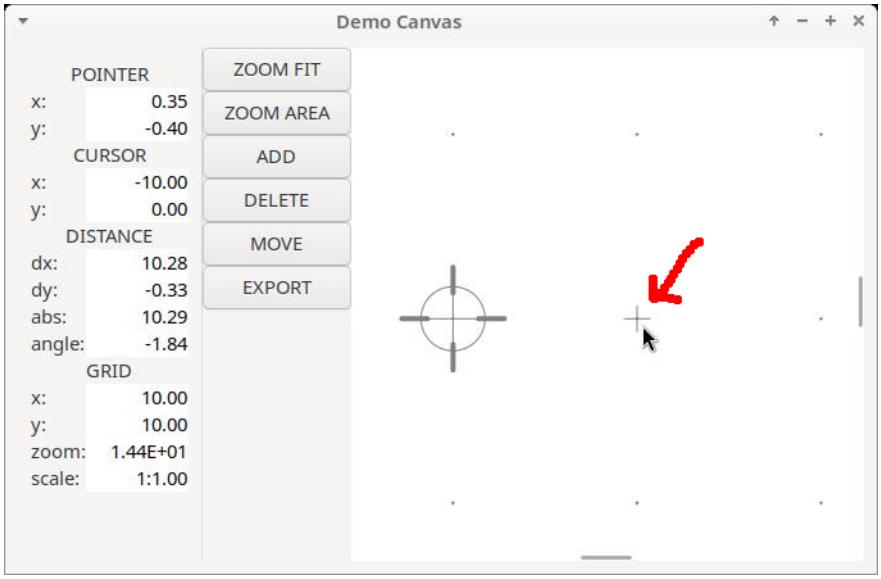

Abbildung 6.23: Zeichnungsursprung

Eine Bereichsprüfung ist in der gegenwärtigen Umsetzung in der Prozedur `draw_origin` nicht vorgesehen. Zur Bereichsprüfung siehe Abschnitt 6.6.6.

6.6.5 Zeichnen des Cursors

Der Cursor kann durch ein schlichtes Kreuz - ähnlich dem Ursprung in Abschnitt 6.6.4 - dargestellt werden. Die Unterscheidung vom Ursprung ist dann aber etwas schwierig. Darum sollte der Cursor die Gestalt eines Fadenkreuzes wie in Abbildung 6.23 haben. Das Zeichnen des Cursors geschieht - wie auch beim Ursprung - direkt auf der Zeichenfläche, also im System CS2. Auf diese Weise bleiben Linienbreiten, Längen, Kreise und Kreisbögen unabhägig vom Vergrößerungsfaktor.

Da der Cursor so gut wie immer im sichtbaren Bereich ist, wird auf eine Bereichsprüfung verzichtet. Details sind in der Prozedur `draw_cursor` zu finden. Siehe Abschnitt 9.16 für den Quellcode.

6.6.6 Bereichsprüfung

Sinn der Bereichsprüfung ist, festzustellen, ob ein primitives Objekt sich teilweise oder vollständig im sichtbaren Bereich befindet. Ist das Objekt im sichtbaren Bereich, wird es auf die Leinwand gezeichnet, ansonsten nicht. Der Grund für die Bereichsprüfung ist, daß die *GTK/Cairo*-Routinen (`line_to, move_to, arc, ...`) zum Zeichnen von Linien, Kreisen und Kreisbögen relativ viel Zeit beanspruchen. Bei tausenden primitiven Objekten macht sich das störend bemerkbar[19]. Der Bildaufbau wird langsamer je mehr Dinge auf die Leinwand gemalt werden. Es geht also darum Rechenzeit zu sparen.

Im Demoprogramm ist in Prozedur `make_database_2` ein Codeabschnitt vorgesehen, um tausende Objekte zu erzeugen. Bei sehr vielen primitiven Objekten ist die Wirkung der Bereichsprüfung gut nachweisbar. Je weniger Objekte sichtbar sind, umso flüssiger geht das Verschieben des Bildes und das Vergrößern vonstatten.

Der im Folgenden beschriebene Aufwand ist also gerechtfertigt und zahlt sich aus. Die Prüfung vereinfacht sich, wenn für ein primitives Objekt vorübergehend ein Begrenzungsrechteck gebildet wird. Dieses Begrenzungsrechteck wird dann darauf geprüft, ob es den rechteckigen sichtbaren Bereich überlappt oder sich darin befindet. Es handelt sich hier also nur im eine überschlägige Prüfung, die den Zweck hinreichend erfüllt. Sie gestaltet sich äußerst einfach, weil die primitiven Objekte nur Linien, Kreise oder Kreisbögen mit einer bestimmten Linienbreite sind. Die Struktur des übergeordneten komplexen Objektes spielt keine Rolle, weil wir nur die primitiven Objekte untersuchen.

Grundlagen zum Begrenzungsrechteck wurden bereits in Abschnitt 3.5 behandelt. Wir kümmern uns hier jedoch nur um ein einziges primitives Objekt. Dessen Begrenzungsrechteck ergibt sich aus diesen Ausgangswerten,

1. der weitesten Ausdehnung nach links, also dem kleinsten verwendeten x-Wert x_1
2. der weitesten Ausdehnung nach rechts, also dem größten verwendeten x-Wert x_2
3. der weitesten Ausdehnung nach unten, also dem kleinsten verwendeten y-Wert y_1
4. der weitesten Ausdehnung nach oben, also dem größten verwendeten y-Wert y_2

Kommen wir nun zur eigentlichen Bereichsprüfung. Diese hat als Ausgangspunkt zwei rechteckige Flächen wie in Abbildung 6.24 dargestellt. Fläche A sei der sichtbare Bereich und Fläche B das Begrenzungsrechteck unseres primitiven Objektes. Jede Fläche hat eine mit einem Kreis gekennzeichnete Position P.

[19] Zu möglichen Lösungen siehe Abschnitt 6.6.11.

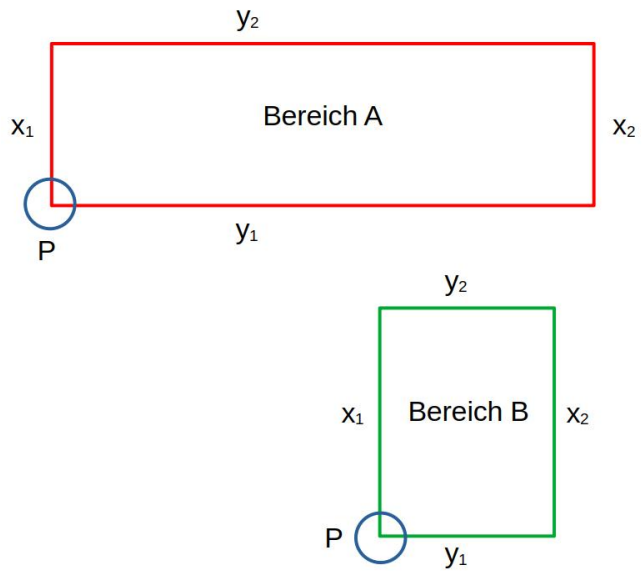

Abbildung 6.24: Bereichsprüfung

Die oben genannten Grenzwerte stellen wir uns zunächst geordnet und systematisch für Fläche A zusammen.

Der kleinste x-Wert ist direkt aus der Position entnehmbar:

$$A.x_1 = A.P_x \tag{6.50}$$

Der größte x-Wert ist um die Breite der Fläche weiter rechts zu finden:

$$A.x_2 = A.x_1 + A_{width} \tag{6.51}$$

Der kleinste y-Wert ist direkt aus der Position entnehmbar:

$$A.y_1 = A.P_y \tag{6.52}$$

Der größte y-Wert ist um die Höhe der Fläche weiter oben zu finden:

$$A.y_2 = A.y_1 + A_{height} \tag{6.53}$$

Nun wiederholen wir diese Schritte für Fläche B und sparen uns die Erläuterungen:

$$B.x_1 = B.P_x \tag{6.54}$$

$$B.x_2 = B.x_1 + B_{width} \tag{6.55}$$

$$B.y_1 = B.P_y \tag{6.56}$$

$$B.y_2 = B.y_1 + B_{height} \tag{6.57}$$

Es folgen nun vier Vergleichsoperationen:

1. Ist B.x_1 kleiner als A.x_2 ?
2. Ist B.x_2 größer als A.x_1 ?
3. Ist B.y_1 kleiner als A.y_2 ?
4. Ist B.y_2 größer als A.y_1 ?

Nur wenn **jeder** dieser Vergleiche positiv ausfällt, dann überlappen sich beide Flächen in irgendeiner Weise.

Im Demoprogramm wurde die Bereichsprüfung in der Funktion `areas_overlap` realisert. Details zur Funktion sind im Quellcode in Abschnitt 9.18 zu finden.

6.6.7 Größenprüfung

Beim Verkleinern der Ansicht schrumpfen die primitiven Objekte irgendwann soweit, daß für dem Betrachter keine Details mehr erkennbar sind. Es macht keinen Sinn, extrem kleine Objekte auf die Leinwand zu malen. Den Rechenaufwand kann man sich also sparen.

Der Anlaß für die Größenprüfung ist derselbe wie bei der Bereichsprüfung (Abschnitt 6.6.6). Es geht wieder um das Reduzieren von Rechenzeit. Die GTK/Cairo-Routinen (`line_to`, `move_to`, `arc`, `...`) zum Zeichnen von Linien, Kreisen und Kreisbögen sind zeitfressend, was bei tausenden primitiven Objekten die Darstellung auf der Leinwand ausbremst[20].

Im Demoprogramm ist in Prozedur `make_database_2` ein Codeabschnitt vorgesehen, um tausende Objekte zu erzeugen. Bei sehr vielen primitiven Objekten ist die Wirkung der Größenprüfung leicht erkennbar. Mit zunehmender Verkleinerung werden immer weniger Objekte dargestellt, und der Bildaufbau geschieht immer zügiger.

Der Aufwand für die Größenprüfung macht also Sinn. Es geht wie auch bei der Bereichsprüfung um die primitiven Objekte. Das Ergebnis der Größenprüfung entscheidet, ob ein primitives Objekt auf die Leinwand gemalt werden soll oder nicht. Eine sehr einfache überschlägige Prüfung der Größe eines primitiven Objektes kann wieder mit Hilfe seines Begrenzungsrechteckes vorgenommen werden. In Abschnitt 3.5 haben wir gelernt, wie dieses gebildet wird. Abbildung 6.25 zeigt das Begrenzungsrechteck eines primitiven Objektes.

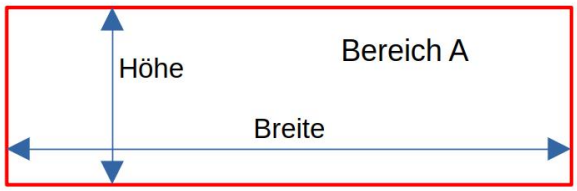

Abbildung 6.25: Größenprüfung

[20] Zu möglichen Lösungen siehe Abschnitt 6.6.11.

Der Ablauf der Prüfung gestaltet sich einfach. Vom Begrenzungsrechteck des Objektes ist nur die Breite und Höhe relevant:

1. Die Breite und Höhe des Begrenzungsrechteckes werden umgerechet in Längen auf der Leinwand (Einheit `lp`). Siehe dazu Abschitte 3.2 und 4.1. Das Ergebnis sind die Werte w und h.

2. Dann wird von w und h der größere beider Werte in die Variable l übernommen.

3. Ist l größer als die Sichtbarkeitsschwelle, dann ist das Objekt groß genug um gezeichnet zu werden. Die Sichtbarkeitsschwelle ist eine systemweite Konstante. Sie hat die Einheit `lp`. Im Demoprogramm hat sie die Bezeichnung `visibility_threshold`.

Im Demoprogramm ist die Größenprüfung durch die Funktion `above_visibility_threshold` realisert worden. Details zur Funktion sind im Quellcode in Abschnitt 9.23 zu finden.

6.6.8 Zeichnungsrahmen und Dokumentationsfeld

Die grundlegenden Dinge zum Zeichnungsrahmen (ZR) und Dokumentationsfeld (DF) wurden bereits in Abschnitt 3.4 behandelt. Beim Darstellen von ZR und DF auf der Leinwand sind diese zu behandeln wie komplexe Objekte. Das heißt: Die Linien, aus denen sie zusammengesetzt sind, werden wie primitive Objekte behandelt und mit der Routine `draw_line` gezeichnet. Die Bereichs- und Größenprüfung wird also auch hier angewendet. Weil ZR und DF eine eigene Position haben (deren untere linke Ecke), werden die Linien durch `draw_line` auch an der finalen Stelle platziert.

Im Demoprogramm ist die Prozedur `draw_drawing_frame` für das Darstellen von ZR und DF zuständig. Details sind im Quellcode in Abschnitt 9.2 zu finden.

6.6.8.1 Anmerkungen zu GTK3/Cairo

Die primitiven Elemente des ZR und DF im Demoprogramm sind Linien. Die Linien sind alle mit der selben Farbe zu zeichnen, jedoch nicht unbedingt mit der selben Linienbreite.

Sofern immer die gleiche Linienbreite verwendet wird, kann man zunächst alles mit den *Cairo*-Routinen `move_to` und `line_to` zeichnen und abschließend nur einmal mit dem Befehl `stroke` den Vorgang abschließen. Das brächte die höchste Zeitersparnis mit sich (Siehe auch Abschnitt 6.6.11).

Im Demoprogramm setzt sich der ZR aus Linien zusammen, die jeweils eine eigene Breite haben dürfen. Die oben beschriebene Optimierung ist also nicht möglich. Es muß nach jeder einzelnen Linie der Befehl `stroke` ausgeführt werden. Allerdings hält sich die Menge primitver Objekte von ZR und DF mit max. 50 Linien in Grenzen, sodaß die fehlende Optimierung nicht ins Gewicht fällt. Zum Vergleich: Die Menge der primitiven Objekte welche durch das Modell zusammenkommt, kann durchaus im vierstelligen Bereich liegen.

6.6.9 Komplexe Objekte zeichnen

Nun ist zu diskutieren, wie komplexe Objekte auf die Leinwand gezeichnet werden sollen. Die Definition für ein komplexes Objekt wurde in Abschnitt 3.3.2 vorgenommen. Wir haben es also mit einer Vielzahl von primitiven Objekten zu tun, die eine relative Lage zur absoluten Position des übergeordneten komplexen Objektes haben. Der Drehwinkel des komplexen Objektes ist zu berücksichtigen. Eventuell ist das komplexe Objekt auch noch gespiegelt. Dazu kommt, daß besonders große komplexe Objekte entsprechend viel Platz auf der Leinwand beanspruchen. Von diesen ist aber oft nur ein Teil im sichtbaren Bereich und hinreichend groß.

Wie ist nun vorzugehen ? Das Zeichnen der komplexen Objekte des Modells beinhaltet folgende Aktionen:

1. Datenbank durchforsten und jedes komplexe Objekt mit einem sogenannten *Iterator* behandeln. Dieser Vorgang ist in der Prozedur `draw_objects` zu finden.

2. Ursprung des komplexen Objektes zeichnen. Der Ursprung markiert die absolute Position des Objektes.

3. Primitive Objekte (Linien, Kreise, Kreisbögen) eines jeden komplexen Objektes mittels Iteratoren durchforsten.

4. Es wird für jedes primitive Objekt - unter Berücksichtigung von Position, Drehung und Spiegelung - ein Begrenzungsrechteck gebildet und dieses der **Bereichsprüfung** unterzogen[21]. Der damit verbundene Rechenaufwand ist unvermeidbar.

5. Jedes primitives Objekt mittels der **Größenprüfung** testen, ob es groß genug ist, um gezeichnet zu werden. Auch um diesen Rechenaufwand kommen wir nicht herum.

6. Die vorangegangenen beiden Punkte wurden im Demoprogramm in den Prozeduren `draw_line` und `draw_circle` untergebracht. Dort erfolgt dann auch der letztendliche Aufruf der primitiven *Cairo*-Zeichenroutinen `move_to`, `line_to` und `arc`.

6.6.9.1 Ursprung

In Abschnitt 3.3.2 wurde die Notwendigkeit eines Ursprunges für ein komplexes Objekt erklärt. Im einfachsten Fall wird der Ursprung als ein kleines Kreuz gezeichnet, wie es in Abbildung 3.3 zu sehen ist.

Betreffend des Vergrößerungsfaktors S kann der Ursprung auf zwei Wegen dargestellt werden:

1. Die Größe des Kreuzes ist konstant, also unabhägig vom Vergrößerungsfaktor S.

2. Das Kreuz wird mit vergrößert oder verkleinert, ist also vom Vergrößerungsfaktor S abhängig.

Im Demoprogramm sind beide Varianten verfügbar. Mittels des Schalters `origin_fixed_size` kann die gewünschte Darstellung eingestellt werden. Siehe Quellcode in Abschnitt 9.3.

Im Modus 1, also konstante Größe, wird direkt auf die Leinwand gezeichnet (CS2). Im Demoprogramm ist eine Bereichs- und Größenprüfung ist für diesen Modus noch nicht eingebaut.

Im Modus 2, variable Größe, werden die beiden Linien des Ursprunges in Modellkoordinaten (CS1) spezifiziert und wie gewöhnliche primitive Objekte gezeichnet. Somit erfolgt auch eine Bereichs- und Größenprüfung. Dieser Modus ist im Demoprogramm standardmäßig eingeschaltet.

[21] Es mußten einige Experimente angestellt werden um zu dieser Erkenntnis zu gelangen.

6.6.9.2 Primitive Zeichenoperationen

Ähnlich dem in Abschnitt 5.1.1.4 beschriebenen Mechanismus zum Berechnen des Begrenzungsrechteckes B wird auch hier vorgegangen. Es ist also wieder zu beachten, daß in Bibliotheken die primitiven Objekte in ihrer **Urform** modelliert sind (Siehe Abschnitt 3.3.2.1). Es gibt folglich Standardwerte für:

- x/y-Position
- Drehwinkel
- Spiegelung (optional)

Wenn ein komplexes Objekt zur Zeichnung hinzugefügt wird, gelten diese Standardwerte nicht mehr, denn der Bediener hat an diesem Punkt das letzte Wort zu Position, Drehwinkel und evtl. einer Spiegelung. Diese Daten müssen beim Zeichnen der primitiven Objekte an ihren finalen Zielkoordinaten einbezogen werden.

Ein Iterator greift ein Objekt nach dem anderen aus der Datenbank (globale Variable `objects_database`) heraus und übergibt es an eine Prozedur, welche das einzelne Objekt behandelt. Diese Prozedur ist entsprechend benannt als `query_object` und kümmert sich nun um die primitiven Objekte. Ein Iterator durchforstet alle Linien, ein zweiter alle Kreise und eventuell ein dritter um alle Kreisbögen. Alle Iteratoren übergeben das jeweilige primitive Objekt an ein spezielles Unterprogramm zur Weiterbehandlung. Im Demoprogramm sind das die Prozeduren `query_line` und `query_circle`. Von dort wird die primitive Zeichenroutine `draw_line` oder `draw_circle` aufgrufen. Diese Prozeduren berechnen die absolute Position der primitiven Objekte, führen die Bereichs- und Größenprüfung durch und geben das Objekt für das letztendliche Zeichnen auf der Leinwand frei. Details zu den primitiven Zeichenoperationen sind im Quellcode in Abschnitt 9.20 zu finden.

Die Zeichenroutinen im Demoprogramm konnten auf ein **Minimum** reduziert werden, weil wir hier nicht mit Drehwinkeln und Spiegelungen arbeiten. Es wird nach dieser Abfolge vorgegangen:

1. Eine Kopie C des primitiven Objektes erzeugen
2. C um die Position des komplexen Objektes verschieben.
3. Bereichs- und Größenprüfung mit dem primitiven Objekt durchführen.
4. Bei Linien: Start- und Endpunkt in Leinwandkoordinaten (CS2) umrechnen.
5. Bei Kreisen: Mittelpunkt in Leinwandkoordinaten (CS2) umrechnen. Radius in logische Pixel (`lp`) umrechnen.
6. GTK/Cairo-Routinen zum Zeichnen von Linie oder Kreis aufrufen. (`line_to, move_to, arc, ...`)

Die **vollständige** Behandlung des primitiven Objektes muß aber nach diesem Schema erfolgen:

1. Eine Kopie C des Objektes erzeugen.
2. Falls mit Spiegelung komplexer Objekte gearbeitet wird, kann C hier gespiegelt werden. Alternativ kann auch später gespiegelt werden (siehe unten).
3. C um den Drehwinkel des komplexen Objektes drehen.
4. Alternativ kann C jetzt gespiegelt werden (siehe oben).

5. C um die Position des komplexen Objektes verschieben.
6. Bereichs- und Größenprüfung durchführen.
7. Bei Linien: Start- und Endpunkt in Leinwandkoordinaten umrechnen.
8. Bei Kreisen: Mittelpunkt in Leinwandkoordinaten umrechnen. Radius in logische Pixel (`lp`) umrechnen.
9. GTK/Cairo-Routinen zum Zeichnen von Linie oder Kreis aufrufen. (`line_to, move_to, arc, ...`)

Betreffend des Darstellens von Linien, Kreisbögen und Kreisen sei zur Wiederholung auf die Abschnitte 3.3.1.1 und 3.3.1.2 verwiesen.

Ist eine Linie, ein Kreis oder ein Kreisbogen Teil einer Kontur, muß die Breite direkt in logischen Pixeln (`lp`) angegeben werden. Somit unterliegt die Breite auf der Leinwand nicht dem Vergrößerungsfaktor und erscheint dem Bediener konstant[22].

Anderenfalls, wenn die Breite in der Realität eine Rolle spielt, muß die Breite in Millimetern angegeben werden. Die Linienbreite wird dann wie jedes andere Maß des Modelles abhängig vom Vergrößerungsfaktor S dargestellt.

[22] Beim Darstellen des Cursors haben wir es mit einer ähnlichen Problematik zu tun. Siehe Abschnitt 6.6.5

6.6.10 Auslösung des Neuzeichnens

Einleitend wurde bereits gesagt, daß das Zeichnen aller Objekte, des Rasters und des Cursors mittels der Prozedur `cb_draw` vorgenommen wird. Nun stellt sich die Frage, wie `cb_draw` aufgerufen werden soll.

Eine grundlegende Sache ist das Signal `on_draw` und seine Verbindung zur Prozedur `cb_draw`. Die Verbindung zwischen dem Signal `on_draw` und der Prozedur `cb_draw` wird bei der Initialisierung des Demoprogrammes in Prozedur `set_up_canvas` hergestellt.

Wir unterscheiden zweierlei Aufrufe der Prozedur `cb_draw`:

1. **Implizierter** Aufruf durch das Signal `on_draw`. Dieses Signal wird durch *GTK* emittiert wenn z.B. der Bediener zwischen dem Hauptfenster des Demoprogrammes und einer anderen Anwendung wechselt. Es wird noch in einigen anderen Situationen gesendet, auf die der Anwender von *GTK* jedoch keinen Einfluß hat[23].

2. **Expliziter** Aufruf durch die Prozedur `refresh`. Diese Prozedur ruft das Unterprogramm `queue_draw` auf. Es handelt sich dabei um eine Prozedur, die die Zeichenfläche vom Typ `gtk_widget` geerbt hat[24]. Das Unterprogramm `queue_draw` sendet das Signal `on_draw`, was die Prozedur `cb_draw` auslöst. **Wichtig:** `queue_draw` plant[25] lediglich das Aussenden des Signals `on_draw` zur nächsten Gelegenheit. Der Name der Prozedur[26] sagt es ja schon, daß es sich um eine Warteschleife handelt.

   Die Situationen und Ereignisse, die das explizite Neuzeichnen erfordern, wurden bereits einleitend in Abschnitt 6.6 aufgezählt. So ist der Aufruf der Prozedur `refresh` in zahlreichen Routinen wie z.B. denen für die Behandlung der Verschiebung von Rollbalken wie `cb_horizontal_moved` zu finden.

Ungeklärt ist im Moment ein seltsamer Effekt, der im Zusammenhang mit Rollbalken auftritt. Möglicherweise sind auch andere Situationen davon betroffen. In der Rückrufroutine `cb_horizontal_moved` kann der Effekt reproduziert werden, indem der explizite Aufruf von `refresh` auskommentiert wird. Beim Bewegen des horizontalen Rollbalkens wird das Signal `on_draw` implizit gesendet, was die Prozedur `cb_draw` auslöst. Jedoch wird das Signal nicht häufig genug gesendet, was zu unbemalten Lücken in der Ansicht führt. Darüber hinaus funktioniert die Prozedur `cb_draw` dann nicht mehr zuverlässig. Aus diesem Grund ist der explizite Aufruf mittels Prozedur `refresh` nötig.

Für Details sei auf die Dokumentation zum *GTK3*-Paket `Gtk.Widget`,
Dateiname `gtk-widget.ads` verwiesen.

[23] Jedenfalls konnte ich das bisher nicht herausfinden.

[24] Unsere Zeichenfläche ist über die globale Variable `canvas` erreichbar. Der Datentyp ist `gtk_drawing_area`, welcher wiederum vom Typ `gtk_widget` abgeleitet ist.

[25] engl. to schedule

[26] Das englische Wort “queue” bedeutet “Reihe, Warteschleife.”

6.6.11 Anmerkungen zu GTK/Cairo-Routinen

Abschließend sollen hier noch einige Gedanken zu den GTK/Cairo-Routinen `line_to, move_to, arc, ...` festgehalten werden. Diese Routinen sind relativ zeitintensiv. Im Quellcode sind die entsprechenden Stellen in den Prozeduren `draw_line` und `draw_circle` gekennzeichnet (Abschnitt 9.20).

Die Cairo-Routine `stroke` ist ebenfalls ein Zeitfresser. Es ist üblich, verschiedene Farben für Objekte zu verwenden. Im Verlauf der vielen Iterationen durch hunderte primitive Objekte (siehe oben) und insbesondere im Zusammenspiel mit Zeichnungsebenen[27] kommt es zu sehr vielen Farbwechseln. Ebenso wird, je nach Anwendungsfall, die Linienbreite mehr oder weniger oft geändert. Die Routine `stroke` "streicht" alle bisher mit `move_to`, `line_to` und `arc` gemalten primitiven Objekte mit der der durch den Cairo-Befehl `set_source_rgb` spezifizierten Farbe und der durch `set_line_width` vorgegebenen Linienbreite. Die Wechsel von Farbe und Linienbreite sollten also auf ein Minimum reduziert werden.

Eine teilweise Entschärfung dieses Problems bringt die Bereichsprüfung und Größenprüfung wie in den Abschnitten 6.6.6 und 6.6.7 ausgeführt ist.

Eine weitere Reduzierung der Rechenzeit ist eventuell mit `OpenGL` oder dem Nachfolger `Vulcan`[10] zu erreichen.

[27] engl. layers

Kapitel 7

Maßstäbliche Darstellung

In Abschitt 3.1 wurde der Begriff "Maßstab" zunächst nur erklärt. Es wurde bisher angenommen, daß alle Objekte im Maßstab M von 1:1 dargestellt werden. Bei allen Umrechnungen und Zeichenoperation spielte der Maßstab M also bisher keine Rolle.

Wie die sogenannte *maßstäbliche Darstellung* in einem CAD-System zu realisieren ist, soll in diesem Kapitel dargelegt werden. Der Grund für die maßstäbliche Darstellung ist ganz einfach der, daß Zeichnungen auf Papier einer bestimmte, genormten Größe untergebracht werden müssen. Ob dann tatsächlich auf Papier oder in ein PDF gedruckt wird, ist zweitranging.

Mit manchen bereits bestehenden CAD-Systemen, die keine maßstäbliche Darstellung unterstützen, helfen sich die Anwender, indem sie einfach den Zeichnungsrahmen weglassen. Ein anderes Provisorium sind Zeichnungsrahmen, die je nach Anwendung sehr klein oder extrem groß (A2, A1, A0) angelegt werden. Das kann und darf jedoch keine Lösung sein (Siehe Ausführungen in Abschnitt 3.4) !.

Die Schreibweise 1:0,1 oder 1:100 wird im Demoprogramm intern auf die Zahl hinter dem Doppelpunkt reduziert. M ist dann z.B. 0,1 oder 100. Der Maßstab M ist im Demoprogramm eine globale Variable. In einem echten CAD-System muß diese durch den Zeichner spezifiziert werden. Siehe Quellcode in Abschnitt 9.21.

Anhand der beiden folgenden Beispiele soll die Anwendung der *maßstäblichen Darstellung* erklärt werden.

Beispiel 1: Große Objekte

Wenn große Bauwerke, wie eine Brücke, konstruiert werden, aber auf einer A4-Seite dargestellt werden sollen, ist z.B. ein Maßstab von etwa 1:50 nötig. Abbildung 7.1 zeigt das Demoprogramm wie es eine 10 Meter lange Brücke (stark vereinfacht) im Maßstab 1:50 - also verkleinert - darstellt. Die Koordinatenanzeige zeigt übrigens im unteren Feld den Maßstab 1:50 an. Der Cursor steht auf dem Fuß des linken Sockels wo hingegen der Mauszeiger sich auf dem Fuß es rechten Sockels befindet. Die Entfernungsanzeige zeigt knapp 10m an. Die Ausgabe der Maße erfolgt also in Originalabmessungen obwohl die verwendete Papiergröße A4 ist.

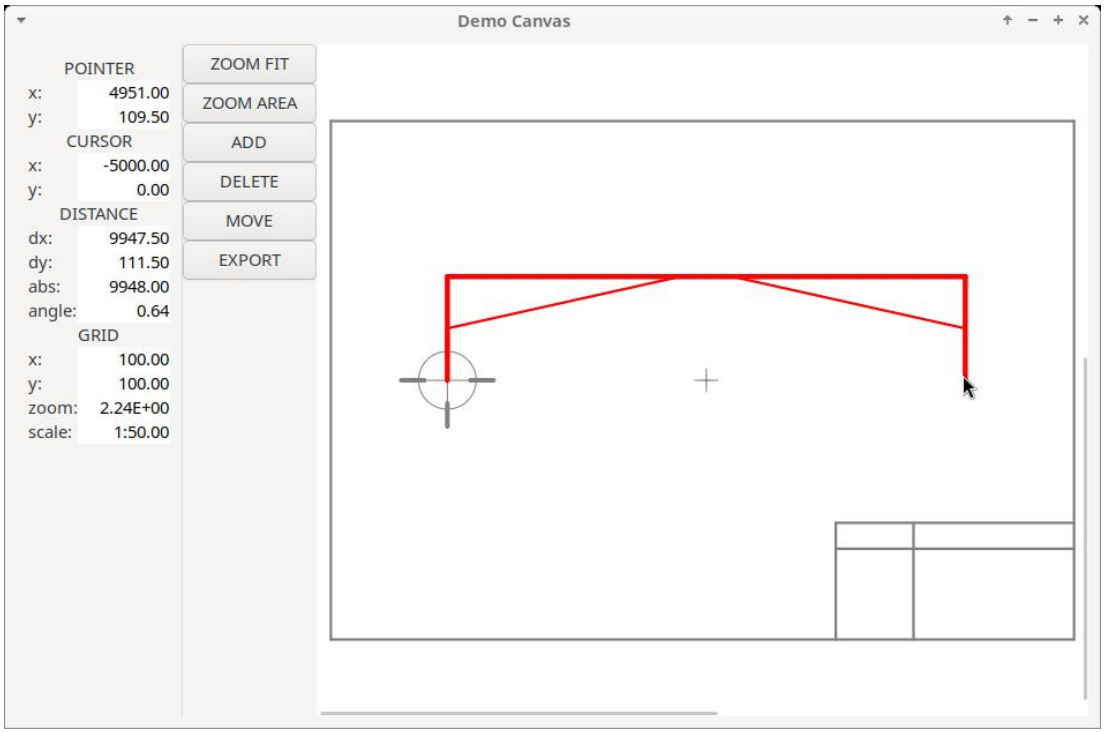

Abbildung 7.1: Darstellung einer Brücke im Maßstab 1:50

Beispiel 2: Kleine Objekte

Ein Beispiel für Vergrößerung im Maßstab 10:1 zeigt Abbildung 7.2. Die dargestellte Schraube ist in der Realität 7mm lang. Der Cursor steht auf der unteren Kante des Schraubenkopfes. Der Mauszeiger am rechten oberen Ende des Gewindes. Die Entfernung von 7,7mm zwischen beiden Punkten kann aus der Koordinatenanzeige direkt in Originalabmessungen abgelesen werden. Auch hier wird die Papiergröße A4 verwendet.

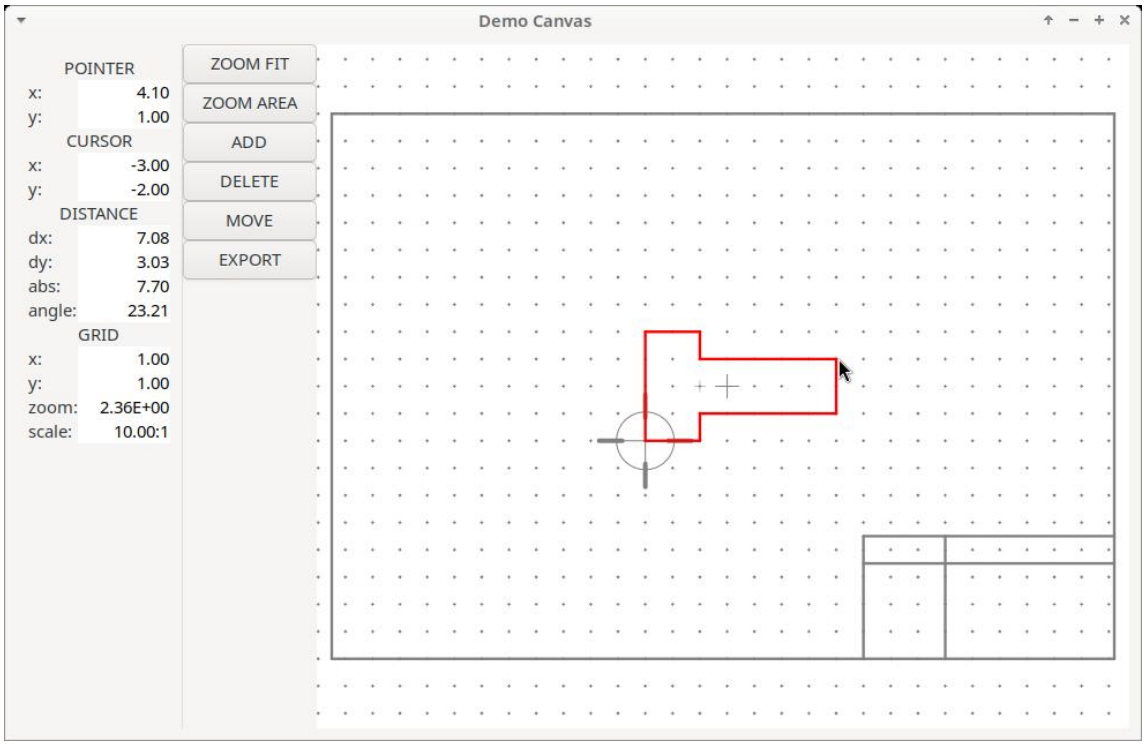

Abbildung 7.2: Darstellung einer Schraube im Maßstab 10:1

Wie ist nun also eine unkomplizierte Erweiterung aller bestehenen Mechanismen um die maßstäbliche Darstellung anzustellen? Die grundsätzliche Idee ist, die erforderlichen Umrechnungen zwischen Realität und Modell um die bestehenden Mechanismen gewissermaßen "herumzuwickeln".

Ausnahme: Alles, was den Zeichnungsrahmen (ZR) und das Dokumentationsfeld (DF) betrifft, ist davon ausgenommen. ZR und DF werden also im Maßstab 1:1 gezeichnet.

Die grundlegenden Routinen, die für die Umrechnung zwischen Modell und Original (Realität) nötig sind, sind im Quellcode in Abschnitt 9.21 zu finden.

Es gelten folgende Grundsätze:

1. Sämtliche Längen, Breiten und Positionsangaben aus der Realität werden bei **Eintritt** in das CAD-System durch M **geteilt**.
2. Beim **Austritt** aus dem System - in die Realität - werden sämtliche Längen, Breiten und Positionsangaben mit M **multipliziert**.
3. Winkel werden **nicht** geändert. Sie sind also vom Maßstab nicht betroffen.
4. Maße betreffend des Zeichnungsrahmens (ZR) und des Dokumentationsfeldes (DF) werden **nicht** geändert.

7.1 Eintrittsstellen

Im Informationsfluß von Realität zum Zeichner und CAD-System gibt es diese Eintrittspforten:

- Einlesen von y,x,z-Position, Längen, Radien und Durchmessern von Objekten aus Zeichnungsdateien[1] und Bauteilbibliotheken;
- y,x,z-Position, Längen, Radien und Durchmesser von Objekten wenn der Bediener diese im laufenden Betrieb ändert;
- die Festlegung des Rasterabstandes durch den Bediener;
- explizite Positionierung des Cursors (sofern ein Cursor verwendet wird) durch den Bediener;
- Importieren von Fertigungsdaten (CAM).

7.1.1 Einlesen von Zeichnungsdaten

Das Demoprogramm verwendet keine Zeichnungsdateien und auch keine Bibliotheken für komplexe Objekte. Statt dessen gibt es zwei Prozeduren `make_database` und `make_database_2` die eine Datenbank mit komplexen Objekten erzeugen. Siehe Quellcode in Abschnitt 9.3. Die Modellierung der primitiven und komplexen Objekte ist im Quellcode gut erkennbar. Alle Positionsangaben, Längen und Breiten entsprechen denen der Realität. Jede der oben genannten Prozeduren erzeugt zwei Datenbanken D_1 und D_2:

- D_1 enthält die realen Objekte. Im Demoprogramm ist das die Liste `objects_database_reality`.
- D_2 enthält die um M vergrößerten oder verkleinerten Objekte. Im Demoprogramm ist das die Liste `objects_database_model`.

Wenn kein Maßstab verwendet wird - also bei M 1:1 - ist D_2 einfach nur eine exakte Kopie von D_1.

Das Darstellen auf der Leinwand basiert einzig auf D_2. Die Ausführungen in den vorherigen Kapiteln 3, 4, 5 und 6 basieren also stillschwiegend nur auf der Datenbank D_2.

Im Programmcode des Demoprogrammes sind drei Beispiele untergebracht, um den Maßstab zu demonstrieren. In den Beispielen sind die Objekte der Realität entsprechend in der Datenbank `objects_database_reality` gespeichert. Sobald diese fertig erstellt ist, werden die um M skalierten Objekte in der Datenbank `objects_database_model` erzeugt. Siehe Prozedur `scale_objects` im Quellcode in Abschnitt 9.3.

[1] engl. design files

1. Prozedur `make_database_1` erzeugt drei Objekte: Ein Rechteck, ein Dreieck und einen Kreis. Diese Objekte sind im Maßstab 1:1 darzustellen und passen somit in den Zeichnungsrahmen auf einem A4 Blatt ohne Mühe hinen. Zum Testen muß der Maßstab auf 1.0 und die Rasterweite auf 10mm gesetzt werden.

2. Prozedur `make_database_3` erzeugt ein Objekt, nämlich eine Brücke von 10 Metern Spannweite und 2 Metern Höhe (siehe Abbildung 7.1) Um ein derart großes Objekt auf einem A4-Blatt darzustellen, bedarf es einer Verkleinerung. Bei einem A4-Blatt von 297mm Breite (Querformat), einem Rand m von 5mm um den Zeichnungsrahmen herum und einem Rand innerhalb des Zeichnungsrahmens von 10mm bleiben 267mm verfügbare Breite für die Brücke:

$$M = \frac{10.000mm}{267mm} \tag{7.1}$$

$$M = 37,5 \tag{7.2}$$

   Der Maßstab M ist also auf 1:40, besser noch 1:50 zu stellen. Für den Rasterabstand sind 100mm ausreichend.

3. Prozedur `make_database_4` erzeugt ebenfalls nur ein Objekt. Dieses ist eine Schraube mit 7mm Länge (Siehe Abbildung 7.2). Es ist also eine Vergrößerung nötig. Wenn die Schraube mit 70mm Länge gezeichnet werden soll, gilt:

$$M = \frac{7mm}{70mm} \tag{7.3}$$

$$M = 0,1 \tag{7.4}$$

   Der Maßstab ist auf 0,1 oder besser 0,04 zu setzen, was dem Verhältnis 10:1 oder 25:1 entspricht. Die Rasterweite kann für dieses Beispiel auf 1,0mm gestellt werden.

7.1.2 Umrechnen des Zeichnungsrasters

Die Weite des Rasters wird entsprechend den Dimensionen der realen Objekte eingestellt. Die Rasterweite ist also eine Eingangsgröße die vom Bediener in der Konstanten `grid_spacing_default` festgelegt wird.

Die Prozedur `set_grid_to_scale` rechnet die gewünschte Weite entsprechend dem Maßstab M um. Das Ergebnis wird in der globalen Variable `grid.spacing` festgehalten. Siehe auch Quellcode in Abschnitt 9.4.

7.2 Austrittsstellen

Im Informationsfluß von CAD-System zum Bediener und letztlich zur Realität gibt es diese Austrittspforten:

- Ausgabe von y,x,z-Positionsanzeigen, Längen, Radien, Durchmessern und Rasterabständen auf dem Bildschirm. Das offensichtlichste Beispiel dafür ist im Demoprogramm die Koordinatenanzeige links der Leinwand. Im Beispiel der 10 Meter langen Brücke (Abbildung 7.1) werden die Originalmaße, also die der Realität angezeigt obwohl wir uns auf einem Papier der Größe A4 befinden.
- Bemaßungen, Maßpfeile. Diese geben immer die Originalgrößen an.
- Erzeugung von Fertigungsdaten (CAM). Die Maße darin müssen dem Original, also der Realität, entsprechen ! Es seien hier nur einige Formate wie *Gerber*®, *Excellon*® und *Step*® als Beispiele genannt.

Kapitel 8

Bedienung des Demoprogrammes

Die Bedienung des Demoprogramms ist an gängige Konventionen angelehnt. Um die folgenden Unterabschnitte auch ohne eine kompilierte und lauffähige Binärdatei zu verstehen, ist Abbildung 8.1 vorgesehen.

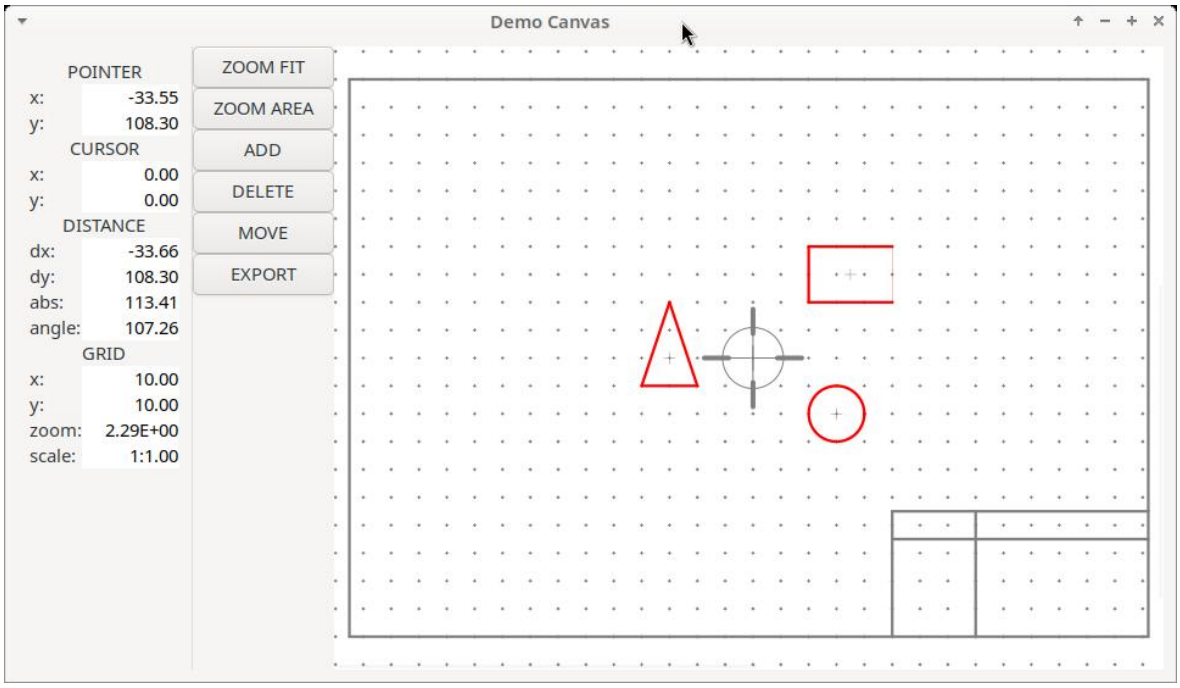

Abbildung 8.1: Demo-GUI

Alle nachfolgend beschriebenen Funktionen sind im Demoprogramm realisert worden und im Quellcode nachvollziehbar.

8.1 Navigation auf der Leinwand

8.1.1 Verschieben des Rollfensters

1. Direktes Verschieben der horizontalen und vertikalen Rollbalken.
2. Indirektes Verschieben der Rollbalken: Hoch-runter durch Drehen des Mausrades. Rechts-links durch Drehen des Mausrades bei gleichzeitig gedrückter SHIFT-Taste.

8.1.2 Vergrößerung und Verkleinerung

1. Vergrößerung oder Verkleinerung an der Position des Mauszeigers: STRG-Taste (engl. CTRL) gedrückt halten und Mausrad drehen.
2. Sofern ein Cursor verwendet wird: Vergrößerung oder Verkleinerung an der Position des Cursors: STRG-Taste (engl. CTRL) gedrückt halten und + oder - Taste drücken.

8.1.3 Einpassen aller Objekte

Dafür ist in der graphischen Bedienoberfläche der Knopf `Zoom Fit` o.ä. vorgesehen. Alternativ kann auch eine Funktionstaste mit dieser Aufgabe verknüpft sein. Im Demoprogramm ist das die Taste F5.

8.1.4 Vergrößern eines Bereiches

Die dafür nötigen Handlungen sind aus zahlreichen anderen graphischen Bedienoberflächen bekannt und sollen hier nur kurz wiedergegeben werden:

1. Durch Betätigen des Knopfes `Zoom Area` wird der Vorgang gestartet.
2. Mit dem Mauszeiger auf eine Ecke des zu vergrößernden Bereiches zeigen, linke Maustaste drücken und gedrückt halten. Die Cursorposition bleibt dabei unberührt.
3. Mauszeiger zum anderen Ende des Bereiches bewegen und linke Maustaste loslassen. Solange der Mauszeiger in Bewegung ist, wird der ausgewählte Bereich durch ein Rechteck visualisiert.

8.1.5 Cursor

Sobald der Bediener auf die Leinwand klickt, wird der Cursor auf den Rasterpunkt gesetzt, der der Zeigerposition am nahesten ist.

Der Cursor kann zudem mittels der Pfeiltasten (rechts, links, hoch, runter) bewegt werden. Die Schrittweite entspricht der Weite des Rasters. Erreicht der Cursor den Rand des sichtbaren Bereiches, verschiebt sich das Bild bis an den Rand des Begrenzungsrechtecks. Darüber hinaus kann der Cursor weiterbewegt werden, ist dann aber nicht mehr sichtbar.

Durch Drücken der Taste `Pos1` (auf englischen Tastaturen ist das die Taste `Home`) wird der Cursor in die Mitte des sichtbaren Bereiches gesetzt.

Der Cursor bewegt sich mit der Ansicht, wenn diese mittels der Rollbalken verschoben wird oder beim Vergrößern oder Verkleinern sich ausdehnt oder zusammenzieht. In diesem Fall wird der Cursor wie ein gewöhnliches Objekt des Modelles behandelt.

Abbildung 8.2 zeigt den Cursor. Links oben im Bild sind die Koordinaten des Mauszeigers zu sehen. Darunter wird die gegenwärtige Position des Cursors angezeigt. Links unten ist außerdem noch die Entfernung von Cursor zu Mauszeiger zu sehen.

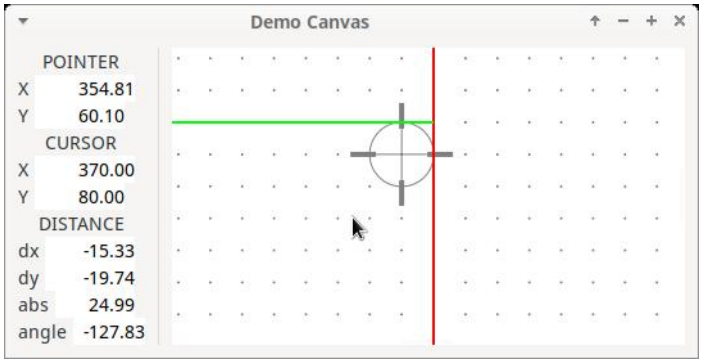

Abbildung 8.2: Cursor und Koordinatenanzeige

Kapitel 9

Quellcode

Der Quellcode wurde auf ein Minimum reduziert, kommt aber dennoch mit einer Komplexität daher, die einiger Erklärungen bedarf. Zunächst ist der Programmcode thematisch in sogenannte Pakete[1] aufgeteilt. Jedes Paket enthält eine Spezifikation mit der Dateiendung `*.ads` (Ada specification)[2] und meistens eine Datei, die die eigentliche Umsetzung von Funktionen und Prozeduren beinhaltet. Diese Dateien enden auf `*.adb` (Ada body). Man spricht hier von dem Körper des Paketes[3].

Die Namensgebung der Pakete ist etwas unüblich. Jeder Name beginnt grundsätzlich mit dem Prefix `demo_` und endet mit der eigentlichen Aufgabe. Der Grund ist, daß eindeutig zwischen Paketnamen, Variablen und reservierten Ada-Schlüsselwörtern getrennt werden muß. Anderenfalls beschwert sich der Kompiler gelegentlich über Namenskonflikte.

Im Folgenden sollen die Pakete kurz beschrieben und deren Inhalt zum Stand der Veröffentlichung dieses Buches abgedruckt werden. Die Kommentare in den Dateien sind in englischer Sprache geschrieben. Der Leser möge es mir nachsehen, daß ich an dieser Stelle keine Übersetzung vornehme. Der Quellcode in Papierform hat darüber hinaus den Vorteil, daß der Leser handschriftliche Notizen vornehmen kann.

Die Bezugsquelle des aktuellen Quellcodes ist in Abschnitt 10.2 zu finden.

Es ist im Rahmen dieses Buches nicht möglich, die Programmiersprache *Ada* umfassend zu lehren. Ich möchte an dieser Stelle lediglich auf die im Literaturverzeichnis gelisteten Quellen verweisen. Was zum Testen des Demoprogrammes nötig ist, werde ich dennoch erläutern.

Jede Datei, ob Spezifikation oder Körper, beginnt mit einem obligatorischen Block betreffend der GNU-Lizensierung und kann beim Lesen getrost übersprungen werden.

Am Anfang jeder Programmzeile steht die Zeilennummer.

Welche Paktete eingebunden[4] werden sollen, wird durch die Anweisungen beginnend mit `with` bestimmt. Zeilen, die mit "- -" beginnen, sind Kommentare oder auskommentierte Anweisungen oder Befehle.

Kommentare, die mit dem Kürzel "CS" beginnen, kennzeichnen Baustellen[5]. Dort sind Dinge noch zu tun oder für zukünftige Erweiterungen vorgesehen.

[1] engl. packages
[2] In der Programmiersprache C ist das die `*.h-Datei`.
[3] In der Programmiersprache C ist das die `*.c-Datei`.
[4] engl. to include
[5] engl. construction site

9.1 Hauptdatei (Main)

Die Hauptdatei ist kein echtes Paket, weil es dazu keine Spezifikation gibt. Es gibt nur die Datei `demo.adb`. Mit dem Kompilieren entsteht unter Linux die ausführbare Binärdatei `demo`. Von der Hauptdatei aus werden Unterprogramme aufgerufen, die in den Paketen zu finden sind.

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                               B o d y                                    --
6  --                                                                          --
7  -- Copyright (C) 2024                                                       --
8  -- Mario Blunk / Blunk electronic                                           --
9  -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
10 --                                                                          --
11 -- This program is free software: you can redistribute it and/or modify    --
12 -- it under the terms of the GNU General Public License as published by    --
13 -- the Free Software Foundation, either version 3 of the License, or        --
14 -- (at your option) any later version.                                      --
15 --                                                                          --
16 -- This program is distributed in the hope that it will be useful,          --
17 -- but WITHOUT ANY WARRANTY; without even the implied warranty of           --
18 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the            --
19 -- GNU General Public License for more details.                            --
20 --                                                                          --
21 -- You should have received a copy of the GNU General Public License       --
22 -- along with this program.  If not, see <http://www.gnu.org/licenses/>.   --
23 ------------------------------------------------------------------------------
24
25 --   For correct displaying set tab width in your editor to 4.
26
27 --   The two letters "CS" indicate a "construction site" where things are not
28 --   finished yet or intended for the future.
29
30 --   Please send your questions and comments to:
31 --
32 --   info@blunk-electronic.de
33 --   or visit <http://www.blunk-electronic.de> for more contact data
34 --
35 --   history of changes:
36 --
37
38 -- This program demonstrates a drawing area - a so called
39 -- canvas (german: Leinwand, Zeichenflaeche) inside a scrolled window.
40 -- In this example we distinguish between canvas-coordinates
41 -- and real-world-model-coordinates.
42
43 -- A simple zoom-in/out-at-mouse-pointer solution is provided here.
44
45 -- In this world the real-world coordinates have the y-axis going
46 -- upwards !
47
48 -- Signals emitted by the canvas are output on the
49 -- console so that the user can watch how callback
50 -- procedures and functions are called.
51
52 -- build with command "gprbuild"
53 -- clean up with command "gprclean"
54
55 with ada.text_io;               use ada.text_io;
56
57 with gtk.main;                  use gtk.main;
58
59 with demo_callbacks;
60 with demo_objects;
61 with demo_frame;
62 with demo_bounding_box;
63 with demo_canvas;
64 with demo_base_offset;
65 with demo_main_window;
66 with demo_scrolled_window;
67 with demo_visible_area;
```

```
68 with demo_coordinates_display;
69 with demo_zoom;
70 with demo_grid;
71
72 procedure demo is begin
73
74     -- Generate a simple drawing frame:
75     demo_frame.make_drawing_frame;
76
77     -- Generate a database with some useless dummy objects:
78     demo_objects.make_database_1; -- rectangle, triangle, circle
79     -- demo_objects.make_database_2; -- performance test
80     -- demo_objects.make_database_3; -- bridge example
81     -- demo_objects.make_database_4; -- screw
82
83     demo_grid.set_grid_to_scale;
84
85     -- Alternatively generate a database with many
86     -- useless dummy objects for perfomance tests:
87     -- demo_objects.make_database_2;
88     -- NOTE: Mind drawing_frame_position in package demo_frame
89     -- for the origin of the drawing.
90
91
92
93     -- Compute the maximal dimensions of the canvas:
94     demo_canvas.compute_canvas_size;
95
96     -- Compute the initial bounding-box:
97     demo_bounding_box.compute_bounding_box;
98
99     -- Compute the base-offset F:
100    demo_base_offset.set_base_offset;
101
102
103    init; -- inits the GTK-stuff
104
105
106    demo_callbacks.set_up_main_window; -- incl. box_h, box_v1, separator, box_v2
107
108    demo_coordinates_display.set_up_coordinates_display; -- table in box_v1
109
110    demo_callbacks.set_up_swin_and_scrollbars;
111    demo_callbacks.set_up_canvas;
112
113
114    put_line ("show␣all␣widgets");
115    demo_main_window.main_window.show_all;
116    -- assigns the allocation to the canvas
117
118
119    demo_scrolled_window.set_initial_scrollbar_settings;
120
121    -- show_adjustments_v;
122    -- show_adjustments_h;
123
124    demo_coordinates_display.update_zoom_display;
125    demo_coordinates_display.update_grid_display;
126    demo_coordinates_display.update_scale_display;
127
128    -- On startup the canvas has the focus. This enables the operator
129    -- to move the cursor with the cursor-keys from the beginning:
130    demo_canvas.canvas.grab_focus;
131
132
133    -- Zoom so that all objects are visible:
134    demo_zoom.zoom_to_fit (demo_bounding_box.bounding_box);
135
136
137    -- Backup the currently visible area.
138    -- This is relevant for canvas mode MODE_3_ZOOM_FIT only:
139    demo_visible_area.backup_visible_area (demo_bounding_box.bounding_box);
140
141    put_line ("start␣gtk␣main␣loop");
142
143    gtk.main.main;
```

```
144
145 end demo;
```

9.2 Das Paket demo_frame

Dieses Paket ist für Papiergrößen, Zeichnungsrahmen und Dokumentationsfeld zuständig.

9.2.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                             DRAWING FRAME                                --
6  --                                                                          --
7  --                                S p e c                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with demo_geometry;             use demo_geometry;
40 with demo_objects;              use demo_objects;
41
42
43 package demo_frame is
44
45     -- The default paper size is A4 landscape:
46     type type_paper is record
47         width   : type_distance_model_positive := 297.0;
48         height  : type_distance_model_positive := 210.0;
49     end record;
50
51     paper : type_paper;
52
53
54     -- The margin around the drawing frame.
55     -- This is the width of the area around the drawing frame:
56     margin : constant type_distance_model_positive := 5.0;
57
58
59
60     type type_drawing_frame is new type_object with record
61         lines   : pac_lines.list;
62         -- CS texts
63         -- CS separate entry for title block elements
64     end record;
65
66     drawing_frame : type_drawing_frame;
67
```

```
68
69      -- The place where the lower left corner of the
70      -- drawing frame relative to the origin is:
71      drawing_frame_position : type_vector_model := (-150.0, -105.0);
72      --drawing_frame_position : type_vector_model := (0.0, 0.0);
73
74
75      -- This procedure generates a very simple
76      -- dummy drawing frame. It applies the
77      -- drawing_frame_position (see above) to the
78      -- drawing_frame.position:
79      procedure make_drawing_frame;
80
81
82      -- Draws all primitive objects of the drawing frame
83      -- and draws them one by one:
84      procedure draw_drawing_frame;
85
86  end demo_frame;
```

9.2.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                             DRAWING FRAME                                --
6  --                                                                          --
7  --                                B o d y                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;                use ada.text_io;
40
41 with cairo;
42
43 with demo_canvas;
44 with demo_primitive_draw_ops;
45
46
47 package body demo_frame is
48
49
50     procedure make_drawing_frame is
51         use pac_lines;
52
53         -- This simple drawing frame consists of lines
54         -- which have a linewidth of 1mm:
55         line : type_line;
56         w : type_distance_model := 1.0;
57
58         c1, c2, c3, c4 : type_vector_model;
59     begin
60         put_line ("make_drawing_frame");
61
62         -- FRAME POSITION:
63
64         -- Set the position of the frame (lower left corner):
65         drawing_frame.position := drawing_frame_position;
66
67         put_line ("␣frame␣position:" & to_string (drawing_frame.position));
68
69         -- PRIMITIVE OBJECTS:
70
71         -- IMPORTANT: The primitve objects are defined as if
72         -- the frame were placed on position (0;0).
73         -- When the frame is drawn on the canvas or when
74         -- the bounding-box is computed, then the primitve
```

```
75          -- objects are moved by the frame position (see assignment above).
76
77          -- These are the four lines that make the
78          -- main rectangle (landscape format):
79
80          -- The dimensions adapt to the paper size and margin:
81          c1 := (margin, margin);
82          c2 := (paper.width - margin, margin);
83          c3 := (paper.width - margin, paper.height - margin);
84          c4 := (margin, paper.height - margin);
85
86          line := (s => c1, e => c2, w => w);
87          drawing_frame.lines.append (line);
88
89          line := (s => c2, e => c3, w => w);
90          drawing_frame.lines.append (line);
91
92          line := (s => c3, e => c4, w => w);
93          drawing_frame.lines.append (line);
94
95          line := (s => c4, e => c1, w => w);
96          drawing_frame.lines.append (line);
97
98
99
100         -- The lines of the title block:
101         line := (s => (200.0, margin), e => (200.0, 50.0), w => w);
102         drawing_frame.lines.append (line);
103
104         line := (s => (230.0, margin), e => (230.0, 50.0), w => w);
105         drawing_frame.lines.append (line);
106
107
108         line := (s => (200.0, 50.0),
109                  e => (paper.width - margin, 50.0), w => w);
110         drawing_frame.lines.append (line);
111
112         line := (s => (200.0, 40.0),
113                  e => (paper.width - margin, 40.0), w => w);
114         drawing_frame.lines.append (line);
115
116     end make_drawing_frame;
117
118
119
120     procedure draw_drawing_frame is
121         use cairo;
122         use demo_canvas;
123         use demo_primitive_draw_ops;
124         use pac_lines;
125
126
127         procedure query_line (lc : in pac_lines.cursor) is
128             -- The candidate line being handled:
129             line : type_line renames element (lc);
130         begin
131             --put_line ("query_line");
132
133             -- Usually the drawing frame has lines with different
134             -- widths. For this reason each line is drawn with
135             -- an explicit stroke:
136             draw_line (line, drawing_frame.position, true);
137         end query_line;
138
139
140     begin
141         --put_line ("draw_drawing_frame");
142
143         -- Set the color:
144         set_source_rgb (context, 0.5, 0.5, 0.5); -- gray
145
146         drawing_frame.lines.iterate (query_line'access);
147         -- CS iterate the lines of the title block separately ?
148
149         -- CS texts
150
```

```
151      end draw_drawing_frame;
152
153
154 end demo_frame;
```

9.3 Das Paket demo_objects

Dieses Paket ist zuständig für

1. die darzustellenden primitiven und komplexen Objekte und ihre Typdeklarationen
2. geometrische Operationen mit komplexen Objekten
3. Begrenzungsrechtecke von primitiven Objekten
4. Skalierung von komplexen Objekten entsprechend des angewendeten Maßstabes M
5. Erzeugung der Datenbanken komplexer Objekte
6. Hinzufügen und Löschen von komplexen Objekten
7. Zeichnen von Objekten auf der Leinwand

9.3.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                                OBJECTS                                   --
6  --                                                                          --
7  --                                S p e c                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.containers;             use ada.containers;
40 with ada.containers.doubly_linked_lists;
41
42 with demo_logical_pixels;        use demo_logical_pixels;
43 with demo_geometry;              use demo_geometry;
44
45
46 package demo_objects is
47
48 -- ORIGIN:
49
50     -- The origin of a complex object is a small cross
```

```
51      -- that marks the center and the position of the object.
52
53      -- This switch determines whether the origin is
54      -- drawn with a fixed or a variable size:
55      origin_fixed_size : boolean := false;
56
57      variable_origin_arm_length : constant type_distance_model_positive := 2.0;
58      variable_origin_linewidth  : constant type_distance_model_positive := 0.2;
59
60      -- the arm-length:
61      fixed_origin_arm_lenght : constant type_logical_pixels_positive := 10.0;
62      fixed_origin_linewidth  : constant type_logical_pixels_positive := 1.0;
63
64
65
66      -- The simplest object in the model world is a line:
67      type type_line is record
68          s, e : type_vector_model; -- start and end point
69          w : type_distance_model_positive := 0.1; -- linewidth
70      end record;
71
72
73      -- CS: If lines are to be divided in
74      -- contours and non-contour related, then this
75      -- type declaration might be a start:
76      -- type type_line_2 (contour : boolean) is record
77      --  s, e : type_vector_model; -- start and end point
78      --  case contour is
79      --      when TRUE =>
80      --          null;
81      --      when FALSE =>
82      --          w : type_distance_model_positive := 0.1; -- linewidth
83      --  end case;
84      -- end record;
85
86
87      -- Returns the bounding-box of the given line.
88      -- It respects the linewidth and assumes that the line ends
89      -- have round caps:
90      function get_bounding_box (
91          line : in type_line)
92          return type_area;
93
94
95      -- Moves a line by the given offset:
96      procedure move_line (
97          line    : in out type_line;
98          offset  : in type_vector_model);
99
100
101
102     -- Another primitive object is a circle:
103     type type_circle is record
104         c : type_vector_model;   -- the center
105         r : type_distance_model_positive; -- the radius
106         w : type_distance_model_positive; -- the linewidth
107         -- CS: fill status
108     end record;
109
110
111     -- CS: If circle are to be divided in
112     -- contours and non-contour related, then this
113     -- type declaration might be a start:
114     -- type type_circle_2 (contour : boolean) is record
115     --  c : type_vector_model;   -- the center
116     --  r : type_distance_model_positive; -- the radius
117     --  case contour is
118     --      when TRUE =>
119     --          null;
120     --      when FALSE =>
121     --          w : type_distance_model_positive := 0.1; -- linewidth
122     --  end case;
123     -- end record;
124
125
126
```

```
127     -- Returns the bounding-box of the given circle.
128     -- It respects the linewidth of the circumfence:
129     function get_bounding_box (
130         circle : in type_circle)
131         return type_area;
132
133
134     -- Moves a circle by the given offset:
135     procedure move_circle (
136         circle  : in out type_circle;
137         offset  : in type_vector_model);
138
139
140     -- CS arc ?
141
142
143
144     -- An object in general has a position in x and y:
145     type type_object is abstract tagged record
146         position : type_vector_model;
147     end record;
148
149
150
151     package pac_lines is new doubly_linked_lists
152         (element_type => type_line);
153
154     package pac_circles is new doubly_linked_lists
155         (element_type => type_circle);
156
157     type type_complex_object is new type_object with record
158         lines   : pac_lines.list;
159         circles : pac_circles.list;
160     end record;
161
162     package pac_objects is new doubly_linked_lists
163         (element_type => type_complex_object);
164
165
166
167     -- This is the database that contains the objects
168     -- of the real world (without scale):
169     objects_database_reality : pac_objects.list;
170
171     -- This is the database that contains the scaled
172     -- objects of the model:
173     objects_database_model : pac_objects.list;
174
175
176
177     -- Generates the database objects_database_model
178     -- from the database objects_database_reality:
179     procedure scale_objects;
180
181     -- This procedure generates a dummy database with
182     -- some useless dummy objects: A square, a triangle
183     -- and a circle.
184     -- Recommendation for proper displaying:
185     -- Set the scale to 1 and grid width to 10mm.
186     procedure make_database_1;
187
188     -- This procedure generates a dummy database with
189     -- many useless dummy objects: Hundreds of squares.
190     -- It is intended for performance testing.
191     -- Recommendation for proper displaying:
192     -- Set the scale to 1 and grid width to 10mm.
193     procedure make_database_2;
194
195     -- This procedure generates a dummy database with
196     -- a single object: A bridge with a span of 10 meters
197     -- and a height of 2 meters.
198     -- Recommendation for proper displaying:
199     -- Set the scale to 50 and grid width to 100mm.
200     procedure make_database_3;
201
202     -- This procedure generates a dummy database with
```

```
203      -- a single object: A small screw of 7mm length.
204      -- Use it to demostrate the application of a scale.
205      -- Recommendation for proper displaying:
206      -- Set the scale to 0.04 and grid width to 1mm.
207      procedure make_database_4;
208
209
210      -- This procedure adds a new object to the database.
211      -- The object is a simple square:
212      procedure add_object;
213
214
215      -- This procedure deletes the last object that has been
216      -- added to the database:
217      procedure delete_object;
218
219
220
221      -- Draws all model objects. Parses the model database
222      -- and draws objects one by one:
223      procedure draw_objects;
224
225  end demo_objects;
```

9.3.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                                OBJECTS                                   --
6  --                                                                          --
7  --                                B o d y                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;               use ada.text_io;
40 with cairo;
41
42 with demo_scale;
43 with demo_zoom;
44 with demo_conversions;
45 with demo_primitive_draw_ops;
46 with demo_canvas;
47
48
49 package body demo_objects is
50
51     function get_bounding_box (
52         line : in type_line)
53         return type_area
54     is
55         -- CS: Optimization required. Compiler options ?
56
57         result : type_area;
58         w, h : type_distance_model;
59
60         d : constant type_distance_model := line.w / 2.0;
61     begin
62         -- x-axis:
63         w := line.e.x - line.s.x;
64
65         if w > 0.0 then -- line runs from left to right
66             result.position.x := line.s.x;
67             result.width := w;
68         else -- line runs from right to left or vertically
69             result.position.x := line.e.x;
70             result.width := -w;
71         end if;
72
73         -- y-axis:
74         h := line.e.y - line.s.y;
```

```
75
76          if h > 0.0 then -- line runs upwards
77              result.position.y := line.s.y;
78              result.height := h;
79          else -- line runs downwards or horizontally
80              result.position.y := line.e.y;
81              result.height := -h;
82          end if;
83
84          -- extend the box by the linewidth:
85          result.width  := result.width  + line.w;
86          result.height := result.height + line.w;
87
88          -- shift the box by half the linewidth down and left:
89          result.position.x := result.position.x - d;
90          result.position.y := result.position.y - d;
91          return result;
92      end get_bounding_box;
93
94
95      procedure move_line (
96          line    : in out type_line;
97          offset  : in type_vector_model)
98      is begin
99          -- CS: Optimization required. Compiler options ?
100         move_by (line.s, offset);
101         move_by (line.e, offset);
102     end move_line;
103
104
105
106     function get_bounding_box (
107         circle : in type_circle)
108         return type_area
109     is
110         -- CS: Optimization required. Compiler options ?
111
112         result : type_area;
113         w : type_distance_model;
114
115         d : constant type_distance_model := circle.w / 2.0;
116     begin
117         w := 2.0 * (circle.r + d);
118
119         result.width := w;
120         result.height := w;
121
122         result.position.x := circle.c.x - w / 2.0;
123         result.position.y := circle.c.y - w / 2.0;
124
125         return result;
126     end get_bounding_box;
127
128
129
130     procedure move_circle (
131         circle  : in out type_circle;
132         offset  : in type_vector_model)
133     is begin
134         -- CS: Optimization required. Compiler options ?
135         move_by (circle.c, offset);
136     end move_circle;
137
138
139
140
141     procedure scale_objects is
142         use demo_scale;
143         use pac_objects;
144
145
146         procedure query_object (oc : in pac_objects.cursor) is
147             o_old : type_complex_object renames element (oc);
148             o_new : type_complex_object;
149
150             use pac_lines;
```

```
151             use pac_circles;
152
153             procedure query_line (lc : in pac_lines.cursor) is
154                 l_old : type_line renames element (lc);
155                 l_new : type_line;
156             begin
157                 l_new.w := to_model (l_old.w);
158                 l_new.s := to_model (l_old.s);
159                 l_new.e := to_model (l_old.e);
160
161                 o_new.lines.append (l_new);
162             end query_line;
163
164
165             procedure query_circle (cc : in pac_circles.cursor) is
166                 c_old : type_circle renames element (cc);
167                 c_new : type_circle;
168             begin
169                 c_new.w := to_model (c_old.w);
170                 c_new.r := to_model (c_old.r);
171                 c_new.c := to_model (c_old.c);
172
173                 o_new.circles.append (c_new);
174             end query_circle;
175
176
177
178         begin
179             o_new.position := to_model (o_old.position);
180
181             o_old.lines.iterate (query_line'access);
182             o_old.circles.iterate (query_circle'access);
183
184             objects_database_model.append (o_new);
185         end query_object;
186
187     begin
188         put_line ("scale_objects");
189         --put_line (" M 1:" & to_string (M));
190         put_line ("␣M:␣" & to_string (M));
191
192         objects_database_model.clear;
193
194         objects_database_reality.iterate (query_object'access);
195     end scale_objects;
196
197
198
199     procedure make_database_1 is
200         use pac_lines;
201         use pac_circles;
202         use pac_objects;
203
204         object  : type_complex_object;
205         line    : type_line;
206         circle  : type_circle;
207
208     begin
209         put_line ("make_database_1");
210
211         -- goto l_end;
212
213         -- POSITION:
214
215         -- Mind drawing_frame_position in package demo_frame
216         -- for the origin of the drawing.
217
218         -- The first dummy object is a square:
219         -- Define the position of the square:
220         object.position := (35.0, 30.0);
221
222
223         -- PRIMITIVE OBJECTS:
224
225         -- IMPORTANT: The primitve objects are defined as if
226         -- the object was placed on position (0;0).
```

```
227         -- When the object is drawn on the canvas or when
228         -- the bounding-box is computed, then the primitve
229         -- objects are moved by the object position (see assignment above).
230
231         line := (s => (-15.0, -10.0), e => (15.0, -10.0), w => 1.0);
232         object.lines.append (line);
233
234         line := (s => (15.0, -10.0), e => (15.0, 10.0), w => 0.1);
235         object.lines.append (line);
236
237         line := (s => (15.0, 10.0), e => (-15.0, 10.0), w => 1.0);
238         object.lines.append (line);
239
240         line := (s => (-15.0, 10.0), e => (-15.0, -10.0), w => 1.0);
241         object.lines.append (line);
242
243         objects_database_reality.append (object);
244         -------------------------------------
245         -- goto l_end;
246
247         object.lines.clear;
248         object.circles.clear;
249
250         -- The 2nd dummy object is a triangle:
251
252         -- POSITION:
253
254         -- Define the position of the triangle:
255         object.position := (-30.0, 0.0);
256
257
258         -- PRIMITIVE OBJECTS:
259
260         -- IMPORTANT: The primitve objects are defined as if
261         -- the object was placed on position (0;0).
262         -- When the object is drawn on the canvas or when
263         -- the bounding-box is computed, then the primitve
264         -- objects are moved by the object position (see assignment above).
265
266         line := (s => (-10.0, -10.0), e => (10.0, -10.0), w => 1.0);
267         object.lines.append (line);
268
269         line := (s => (10.0, -10.0), e => (0.0, 20.0), w => 1.0);
270         object.lines.append (line);
271
272         line := (s => (0.0, 20.0), e => (-10.0, -10.0), w => 1.0);
273         object.lines.append (line);
274
275
276         objects_database_reality.append (object);
277
278         -------------------------------------
279         -- goto l_end;
280
281         object.lines.clear;
282         object.circles.clear;
283
284         -- The 3rd dummy object is a circle:
285         object.position := (30.0, -20.0);
286
287         circle := (c => (0.0, 0.0), r => 10.0, w => 1.0);
288         object.circles.append (circle);
289
290         objects_database_reality.append (object);
291
292         -- Now the database with the real world objects is complete.
293         -- The objects must now be scaled according to the
294         -- scale that was specified by the operator:
295         scale_objects;
296
297     end make_database_1;
298
299
300
301     procedure make_database_2 is
302
```

```
303         procedure make_object (
304             destination : in type_vector_model)
305         is
306             use pac_lines;
307             use pac_circles;
308             use pac_objects;
309
310             O : type_complex_object;
311             L : type_line;
312         begin
313             O.position := destination;
314
315             -- IMPORTANT: The primitve objects are defined as if
316             -- the object was placed on position (0;0).
317             -- When the object is drawn on the canvas or when
318             -- the bounding-box is computed, then the primitve
319             -- objects are moved by the object position
320             -- (see assignment above).
321
322             -- The object to be created is a square:
323             L := (s => (-5.0, -5.0), e => (5.0, -5.0), w => 1.0);
324             O.lines.append (L);
325
326             L := (s => (5.0, -5.0), e => (5.0, 5.0), w => 1.0);
327             O.lines.append (L);
328
329             L := (s => (5.0, 5.0), e => (-5.0, 5.0), w => 1.0);
330             O.lines.append (L);
331
332             L := (s => (-5.0, 5.0), e => (-5.0, -5.0), w => 1.0);
333             O.lines.append (L);
334
335             objects_database_reality.append (O);
336         end make_object;
337
338         -- The first object will be placed here:
339         -- position : type_vector_model := (10.0, 10.0);
340         position : type_vector_model := (-140.0, -90.0);
341
342     begin
343         put_line ("make_database_2");
344
345         -- This creates some squares spread across the sheet:
346 --      for column in 1 .. 10 loop
347 --          for row in 1 .. 15 loop
348 --              make_object (position);
349 --              position.x := position.x + 20.0;
350 --          end loop;
351 --
352 --          position.x := 10.0;
353 --          position.y := position.y + 20.0;
354 --      end loop;
355
356         -- CS usa a switch to activate:
357
358         -- This creates 10.000 squares:
359         for column in 1 .. 400 loop
360             for row in 1 .. 400 loop
361                 make_object (position);
362                 position.x := position.x + 1.0;
363             end loop;
364
365             position.x := -140.0;
366             position.y := position.y + 1.0;
367         end loop;
368
369
370         -- Now the database with the real world objects is complete.
371         -- The objects must now be scaled according to the
372         -- scale that was specified by the operator:
373         scale_objects;
374
375     end make_database_2;
376
377
378
```

```
379     procedure make_database_3 is
380         use pac_lines;
381         use pac_objects;
382
383         object  : type_complex_object;
384         line    : type_line;
385
386     begin
387         put_line ("make_database_3");
388
389         -- Mind drawing_frame_position in package demo_frame
390         -- for the origin of the drawing.
391
392         -- The first dummy object is a square:
393         -- Define the position of the square:
394         object.position := (0.0, 0.0);
395
396         -- PRIMITIVE OBJECTS:
397
398         -- IMPORTANT: The primitve objects are defined as if
399         -- the object was placed on position (0;0).
400         -- When the object is drawn on the canvas or when
401         -- the bounding-box is computed, then the primitve
402         -- objects are moved by the object position (see assignment above).
403
404         line := (s => (-5000.0, 0.0), e => (-5000.0, 2000.0), w => 100.0);
405         object.lines.append (line);
406
407         line := (s => (-5000.0, 2000.0), e => (5000.0, 2000.0), w => 100.0);
408         object.lines.append (line);
409
410         line := (s => (5000.0, 2000.0), e => (5000.0, 0.0), w => 100.0);
411         object.lines.append (line);
412
413         line := (s => (-5000.0, 1000.0), e => (-500.0, 2000.0), w => 50.0);
414         object.lines.append (line);
415
416         line := (s => (5000.0, 1000.0), e => (500.0, 2000.0), w => 50.0);
417         object.lines.append (line);
418
419         objects_database_reality.append (object);
420
421         -- Now the database with the real world objects is complete.
422         -- The objects must now be scaled according to the
423         -- scale that was specified by the operator:
424         scale_objects;
425
426     end make_database_3;
427
428
429     procedure make_database_4 is
430         use pac_lines;
431         use pac_objects;
432
433         object  : type_complex_object;
434         line    : type_line;
435
436     begin
437         put_line ("make_database_4");
438
439         -- Mind drawing_frame_position in package demo_frame
440         -- for the origin of the drawing.
441
442         -- The first dummy object is a square:
443         -- Define the position of the square:
444         object.position := (-1.0, 0.0);
445
446         -- PRIMITIVE OBJECTS:
447
448         -- IMPORTANT: The primitve objects are defined as if
449         -- the object was placed on position (0;0).
450         -- When the object is drawn on the canvas or when
451         -- the bounding-box is computed, then the primitve
452         -- objects are moved by the object position (see assignment above).
453
454         line := (s => (-2.0, -2.0), e => (0.0, -2.0), w => 0.1);
```

```
455         object.lines.append (line);
456
457         line := (s => (0.0, -2.0), e => (0.0, -1.0), w => 0.1);
458         object.lines.append (line);
459
460         line := (s => (-0.0, -1.0), e => (5.0, -1.0), w => 0.1);
461         object.lines.append (line);
462
463         line := (s => (5.0, -1.0), e => (5.0,  1.0), w => 0.1);
464         object.lines.append (line);
465
466         line := (s => (5.0, 1.0), e => (0.0, 1.0), w => 0.1);
467         object.lines.append (line);
468
469         line := (s => (0.0, 1.0), e => (0.0, 2.0), w => 0.1);
470         object.lines.append (line);
471
472         line := (s => (0.0, 2.0), e => (-2.0, 2.0), w => 0.1);
473         object.lines.append (line);
474
475         line := (s => (-2.0, 2.0), e => (-2.0, -2.0), w => 0.1);
476         object.lines.append (line);
477
478         objects_database_reality.append (object);
479
480         -- Now the database with the real world objects is complete.
481         -- The objects must now be scaled according to the
482         -- scale that was specified by the operator:
483         scale_objects;
484
485     end make_database_4;
486
487
488
489
490     procedure add_object is
491         use pac_lines;
492         use pac_circles;
493         use pac_objects;
494
495         object : type_complex_object;
496         line : type_line;
497
498     begin
499         -- put_line ("add_object");
500
501         -- The 1st object is a square:
502
503
504         -- POSITION:
505
506         -- Define the position of the square:
507         object.position := (-200.0, 0.0);
508
509         -- In order to simulate a violation of the maximum
510         -- allowed bounding-box dimensions try this:
511         -- object.p := (2500.0, -250.0); -- width exceeded
512         -- object.p := (500.0, -1000.0); -- height exceeded
513
514
515         -- PRIMITIVE OBJECTS:
516
517         -- IMPORTANT: The primitve objects are defined as if
518         -- the object was placed on position (0;0).
519         -- When the object is drawn on the canvas or when
520         -- the bounding-box is computed, then the primitve
521         -- objects are moved by the object position (see assignment above).
522
523         line := (s => (-10.0, -10.0), e => (10.0, -10.0), w => 1.0);
524         object.lines.append (line);
525
526         line := (s => (10.0, -10.0), e => (10.0, 10.0), w => 1.0);
527         object.lines.append (line);
528
529         line := (s => (10.0, 10.0), e => (-10.0, 10.0), w => 1.0);
530         object.lines.append (line);
```

```
531
532         line := (s => (-10.0, 10.0), e => (-10.0, -10.0), w => 1.0);
533         object.lines.append (line);
534
535         objects_database_reality.append (object);
536
537         -- Now the database with the real world objects is complete.
538         -- The objects must now be scaled according to the
539         -- scale that was specified by the operator:
540         scale_objects;
541
542     end add_object;
543
544
545     procedure delete_object is
546         use pac_objects;
547     begin
548         put_line ("delete_object");
549
550         objects_database_reality.delete_last;
551
552         -- Now the database with the real world objects is complete.
553         -- The objects must now be scaled according to the
554         -- scale that was specified by the operator:
555         scale_objects;
556
557     end delete_object;
558
559
560
561     procedure draw_objects is
562         use cairo;
563         use demo_primitive_draw_ops;
564         use demo_canvas;
565         use demo_conversions;
566         use pac_lines;
567         use pac_circles;
568         use pac_objects;
569
570
571         procedure query_object (oc : in pac_objects.cursor) is
572             object : type_complex_object renames element (oc);
573
574
575             procedure draw_origin is
576
577                 -- This procedure draws the origin as a cross of fixed size.
578                 -- It is independed of the zoom factor.
579                 -- The drawing is done directly with canvas coordinates:
580                 procedure fixed_size is
581                     use demo_zoom;
582                     cp : type_logical_pixels_vector;
583                 begin
584                     -- Compute the center of the origin as canvas point:
585                     cp := real_to_canvas (object.position, S);
586
587                     -- Set the linewidth:
588                     set_line_width (context,
589                         to_gdouble (fixed_origin_linewidth));
590
591                     -- Draw the horizontal line from left to right:
592                     move_to (context,
593                              to_gdouble (cp.x - fixed_origin_arm_lenght),
594                              to_gdouble (cp.y));
595
596                     line_to (context,
597                              to_gdouble (cp.x + fixed_origin_arm_lenght),
598                              to_gdouble (cp.y));
599
600                     -- Draw the vertical line from top to bottom:
601                     move_to (context,
602                              to_gdouble (cp.x),
603                              to_gdouble (cp.y - fixed_origin_arm_lenght));
604
605                     line_to (context,
606                              to_gdouble (cp.x),
```

```
607                         to_gdouble (cp.y + fixed_origin_arm_lenght));
608
609                 stroke;
610             end fixed_size;
611
612
613
614             -- This procedure draws the origin as a cross of variable size.
615             -- The size depends on the zoom factor.
616             -- The drawing is in model coordinates:
617             procedure variable_size is
618                 -- The linewidth can be set here. Start and end point
619                 -- will follow later:
620                 line : type_line := (w => variable_origin_linewidth,
621                                      others => <>);
622             begin
623                 -- horizontal line:
624                 line.s := (x => - variable_origin_arm_length,
625                            y => 0.0); -- start
626
627                 line.e := (x => + variable_origin_arm_length,
628                            y => 0.0); -- end
629
630                 draw_line (line, object.position, true);
631
632                 -- vertical line:
633                 line.s := (x => 0.0,
634                            y => - variable_origin_arm_length); -- start
635
636                 line.e := (x => 0.0,
637                            y => + variable_origin_arm_length); -- end
638
639                 draw_line (line, object.position, true);
640             end variable_size;
641
642
643         begin
644             -- Set the color of the origin:
645             set_source_rgb (context, 0.5, 0.5, 0.5); -- gray
646
647             -- Here we decide whether to draw the origin with a
648             -- variable or a fixed size:
649             if origin_fixed_size then
650                 fixed_size;
651             else
652                 variable_size;
653             end if;
654
655         end draw_origin;
656
657
658         procedure query_line (lc : in pac_lines.cursor) is
659             line : type_line renames element (lc);
660         begin
661             --put_line ("query_line");
662
663             -- If the line candidate has a special color,
664             -- then the color must be set here.
665
666             -- If the line candidate has a special color
667             -- or a special linewidth then the draw routine
668             -- must perfom a dedicated stroke:
669             draw_line (line, object.position, true);
670
671             -- If lots of lines have the same linewidth
672             -- and color then this call is sufficient
673             -- and takes less time to execute:
674             -- draw_line (line, object.position);
675
676             -- If the line candidate has a special color,
677             -- then a dedicated stroke command is required here.
678         end query_line;
679
680
681         procedure query_circle (cc : in pac_circles.cursor) is
682             circle : type_circle renames element (cc);
```

```
683             begin
684                 -- put_line ("query_circle");
685
686                 -- If the circle candidate has a special color,
687                 -- then the color must be set here.
688
689                 -- If the circle candidate has a special color
690                 -- or a special linewidth then the draw routine
691                 -- must perfom a dedicated stroke:
692                 draw_circle (circle, object.position, true);
693
694                 -- If lots of arcs have the same linewidth
695                 -- and color then this call is sufficient
696                 -- and takes less time to execute:
697                 -- draw_circle (circle, object.position);
698
699                 -- If the circle candidate has a special color,
700                 -- then a dedicated stroke command is required here.
701             end query_circle;
702
703
704         begin
705             --put_line ("query_object");
706             draw_origin;
707
708             -- If lots of primitive objects are to be drawn
709             -- with all having the same color then do:
710             set_source_rgb (context, 1.0, 0.0, 0.0);
711
712             -- If lots of primitive objects are to be drawn
713             -- with all having the same linewidth then do:
714             -- set_line_width (context, to_gdouble (to_distance (1.0)));
715
716             object.lines.iterate (query_line'access);
717             object.circles.iterate (query_circle'access);
718
719             -- If lots of primitive objects are to be drawn
720             -- with all having the same linewidth and color
721             -- then a single stroke command is sufficient:
722             -- stroke;
723         end query_object;
724
725
726     begin
727         --put_line ("draw_objects");
728
729         objects_database_model.iterate (query_object'access);
730     end draw_objects;
731
732
733 end demo_objects;
```

9.4 Das Paket demo_grid

Betreffend des Zeichnungsrasters sind hier die Deklarationen und Unterprogramme zu finden:

1. Darstellung als Linien oder Punkte
2. Größe des Rasters
3. Fangfunktionen[6]
4. Zeichnen des Rasters auf der Leinwand

9.4.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                                 GRID                                     --
6  --                                                                          --
7  --                               S p e c                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with demo_logical_pixels;       use demo_logical_pixels;
40 with demo_geometry;             use demo_geometry;
41
42
43 package demo_grid is
44
45     -- The grid helps the operator to align or place objects:
46     type type_grid_on_off is (GRID_ON, GRID_OFF);
47     type type_grid_style is (STYLE_DOTS, STYLE_LINES);
48
49     -- The linewidth of the grid lines:
50     grid_width_lines : constant type_logical_pixels_positive := 0.5;
51
52     -- The linewidth of the circles which form the grid dots:
53     grid_width_dots : constant type_logical_pixels_positive := 1.0;
54     grid_radius_dots : constant type_logical_pixels_positive := 0.5;
55
```

[6] engl. snap to grid

```
56
57      -- The arm length of a grid point if drawn as a cross:
58      grid_cross_arm_length : constant type_logical_pixels_positive := 1.0;
59
60
61      -- The default grid size in in the model domain:
62      grid_spacing_default : constant type_distance_model_positive := 10.0;
63      -- use it for the example with the rectangle, triangle and circle
64
65      -- grid_spacing_default : constant type_distance_model_positive := 100.0;
66      -- use it for the bridge example
67
68      -- grid_spacing_default : constant type_distance_model_positive := 1.0;
69      -- use it for the screw example
70
71      -- If the displayed grid is too dense, then it makes no
72      -- sense to draw a grid. For this reason we define a minimum
73      -- distance between grid rows and columns. If the spacing becomes
74      -- greater than this threshold then the grid will be drawn:
75      grid_spacing_min : constant type_logical_pixels_positive := 10.0;
76
77
78      type type_grid is record
79          on      : type_grid_on_off := GRID_ON;
80          -- on       : type_grid_on_off := GRID_OFF;
81          spacing : type_vector_model := (others => grid_spacing_default);
82          style   : type_grid_style := STYLE_DOTS;
83          --style : type_grid_style := STYLE_LINES;
84      end record;
85
86
87      -- This is the grid used by this demo program:
88      grid : type_grid;
89
90
91      -- This procedure sets the grid spacing
92      -- according to the scale specified
93      -- by the operator (see package spec. demo_scale):
94      procedure set_grid_to_scale;
95
96
97      -- This function returns the grid point that is
98      -- closest to the given model point;
99      function snap_to_grid (
100         point : in type_vector_model)
101         return type_vector_model;
102
103
104
105
106     -- This function returns the space between
107     -- the grid columns or rows. It returns the lesser
108     -- spacing of them. It calculates the spacing by this
109     -- equation:
110     -- x = grid.spacing.x * zoom factor
111     -- y = grid.spacing.y * zoom factor
112     -- Then the lesser one, either x or y will be returned:
113     function get_grid_spacing (
114         grid : in type_grid)
115         return type_logical_pixels_positive;
116
117
118
119     -- This procedure draws the grid in the visible area.
120     -- Outside the visible area nothing is drawn in order to save time.
121     -- The procedure works as follows:
122     -- 1. Define the begin and end of the visible area in
123     --    x and y direction.
124     -- 2. Find the first column that comes after the begin of
125     --    the visible area (in x direction).
126     -- 3. Find the last column that comes before the end of the
127     --    visible area (in x direction).
128     -- 4. Find the first row that comes after the begin of the
129     --    visible area (in y direction).
130     -- 5. Find the last row that comes before the end of the
131     --    visible area (in y direction).
```

```
132      -- 6. Draw the grid as dots or lines, depending on the user specified
133      --    settings.
134      procedure draw_grid;
135
136
137  end demo_grid;
```

9.4.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                                GRID                                      --
6  --                                                                          --
7  --                               B o d y                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;                use ada.text_io;
40
41 with cairo;
42 with demo_zoom;                  use demo_zoom;
43 with demo_visible_area;
44 with demo_conversions;
45 with demo_canvas;
46 with demo_scale;
47
48 package body demo_grid is
49
50
51     procedure set_grid_to_scale is
52         use demo_scale;
53     begin
54         to_model (grid.spacing.x);
55         to_model (grid.spacing.y);
56     end set_grid_to_scale;
57
58
59
60     function snap_to_grid (
61         point : in type_vector_model)
62         return type_vector_model
63     is
64         n : integer;
65         type type_float is new float; -- CS refinement required
66         f : type_float;
67         result : type_vector_model;
68     begin
69         n := integer (point.x / grid.spacing.x);
70         f := type_float (n) * type_float (grid.spacing.x);
71         result.x := type_distance_model (f);
72
73         n := integer (point.y / grid.spacing.y);
74         f := type_float (n) * type_float (grid.spacing.y);
```

```
75          result.y := type_distance_model (f);
76
77          return result;
78      end snap_to_grid;
79
80
81      function get_grid_spacing (
82          grid : in type_grid)
83          return type_logical_pixels_positive
84      is
85          sg : constant type_logical_pixels := type_logical_pixels (S);
86          x, y : type_logical_pixels;
87      begin
88          x := type_logical_pixels (grid.spacing.x) * sg;
89          y := type_logical_pixels (grid.spacing.y) * sg;
90          return type_logical_pixels_positive'min (x, y);
91      end get_grid_spacing;
92
93
94
95      procedure draw_grid is
96          use cairo;
97          use demo_conversions;
98          use demo_grid;
99          use demo_visible_area;
100         use demo_canvas;
101
102         type type_float_grid is new float; -- CS refinement required
103
104         -- X-AXIS:
105
106         -- The first and the last column:
107         x1, x2 : type_distance_model;
108
109         -- The start and the end of the visible area:
110         ax1 : constant type_float_grid :=
111             type_float_grid (visible_area.position.x);
112
113         ax2 : constant type_float_grid :=
114             ax1 + type_float_grid (visible_area.width);
115
116         -- The grid spacing:
117         gx : constant type_float_grid :=
118             type_float_grid (grid.spacing.x);
119
120
121         -- Y-AXIS:
122
123         -- The first and the last row:
124         y1, y2 : type_distance_model;
125
126         -- The start and the end of the visible area:
127         ay1 : constant type_float_grid :=
128             type_float_grid (visible_area.position.y);
129
130         ay2 : constant type_float_grid :=
131             ay1 + type_float_grid (visible_area.height);
132
133         -- The grid spacing:
134         gy : constant type_float_grid :=
135             type_float_grid (grid.spacing.y);
136
137         c : type_float_grid;
138
139         -- debug : boolean := false;
140
141
142         procedure compute_first_and_last_column is begin
143             -- Compute the first column:
144             -- put_line (" ax1 " & type_float_grid'image (ax1));
145             c := type_float_grid'floor (ax1 / gx);
146             x1 := type_distance_model ((gx * c) + gx);
147             -- put_line (" x1  " & type_distance_model'image (x1));
148
149             -- Compute the last column:
150             -- put_line (" ax2 " & type_float_grid'image (ax2));
```

```
151             c := type_float_grid'floor (ax2 / gx);
152             x2 := type_distance_model (gx * c);
153             -- put_line (" x2  " & type_distance_model'image (x2));
154         end compute_first_and_last_column;
155
156
157         procedure compute_first_and_last_row is begin
158             -- Compute the first row:
159             -- put_line (" ay1 " & type_float_grid'image (ay1));
160             c := type_float_grid'floor (ay1 / gy);
161             y1 := type_distance_model ((gy * c) + gy);
162             -- put_line (" y1  " & type_distance_model'image (y1));
163
164             -- Compute the last row:
165             -- put_line (" ay2 " & type_float_grid'image (ay2));
166             c := type_float_grid'floor (ay2 / gy);
167             y2 := type_distance_model (gy * c);
168             -- put_line (" y2  " & type_distance_model'image (y2));
169         end compute_first_and_last_row;
170
171
172         -- This procedure draws the dots of the grid:
173         -- 1. Assemble from the first row and the first colum a real
174         --    model point MP.
175         -- 2. Advance PM from row to row and column to column in a
176         --    matrix like order.
177         -- 3. Draw a very small circle, which will appear like a dot,
178         --    (or alternatively a very small cross) at MP.
179         procedure draw_dots is
180             MP : type_vector_model;
181             CP : type_logical_pixels_vector;
182         begin
183             -- Set the linewidth of the dots:
184             set_line_width (context, to_gdouble (grid_width_dots));
185
186             -- Compose a model point from the first column and
187             -- the first row:
188             MP := (x1, y1);
189
190             -- Advance PM from column to column:
191             while MP.x <= x2 loop
192
193                 -- Advance PM from row to row:
194                 MP.y := y1;
195                 while MP.y <= y2 loop
196                     -- Convert the current real model point MP to a
197                     -- point on the canvas:
198                     CP := real_to_canvas (MP, S);
199
200                     -- Draw a very small circle with its center at CP:
201                     -- arc (context, CP.x, CP.y,
202                     --   radius => grid_radius_dots, angle1 => 0.0,
203                     --    angle2 => 6.3);
204                     -- stroke (context);
205
206                     -- Alternatively, draw a very small cross at CP.
207                     -- This could be more efficient than a circle:
208
209                     -- horizontal line:
210                     move_to (context,
211                         to_gdouble (CP.x - grid_cross_arm_length),
212                         to_gdouble (CP.y));
213
214                     line_to (context,
215                         to_gdouble (CP.x + grid_cross_arm_length),
216                         to_gdouble (CP.y));
217
218
219                     -- vertical line:
220                     move_to (context,
221                         to_gdouble (CP.x),
222                         to_gdouble (CP.y - grid_cross_arm_length));
223
224                     line_to (context,
225                         to_gdouble (CP.x),
226                         to_gdouble (CP.y + grid_cross_arm_length));
```

```
227
228
229                 -- Advance one row up:
230                 MP.y := MP.y + grid.spacing.y;
231             end loop;
232
233             -- Advance one column to the right:
234             MP.x := MP.x + grid.spacing.x;
235         end loop;
236     end draw_dots;
237
238
239     -- This procedure draws the lines of the grid:
240     procedure draw_lines is
241         MP1 : type_vector_model;
242         MP2 : type_vector_model;
243
244         CP1 : type_logical_pixels_vector;
245         CP2 : type_logical_pixels_vector;
246
247         ax1f : type_distance_model := visible_area.position.x;
248         ax2f : type_distance_model := ax1f + visible_area.width;
249
250         ay1f : type_distance_model := visible_area.position.y;
251         ay2f : type_distance_model := ay1f + visible_area.height;
252     begin
253         -- Set the linewidth of the lines:
254         set_line_width (context, to_gdouble (grid_width_lines));
255
256         -- VERTICAL LINES:
257
258         -- All vertical lines start at the bottom of
259         -- the visible area:
260         MP1 := (x1, ay1f);
261
262         -- All vertical lines end at the top of the
263         -- visible area:
264         MP2 := (x1, ay2f);
265
266         -- The first vertical line runs along the first column.
267         -- The last vertical line runs along the last column.
268         -- This loop advances from one column to the next and
269         -- draws a vertical line:
270         while MP1.x <= x2 loop
271             CP1 := real_to_canvas (MP1, S);
272             CP2 := real_to_canvas (MP2, S);
273
274             move_to (context,
275                 to_gdouble (CP1.x), to_gdouble (CP1.y));
276
277             line_to (context,
278                 to_gdouble (CP2.x), to_gdouble (CP2.y));
279
280             MP1.x := MP1.x + grid.spacing.x;
281             MP2.x := MP2.x + grid.spacing.x;
282         end loop;
283
284
285         -- HORIZONTAL LINES:
286
287         -- All horizontal lines start at the left edge of the
288         -- visible area:
289         MP1 := (ax1f, y1);
290
291         -- All horizontal lines end at the right edge of the
292         -- visible area:
293         MP2 := (ax2f, y1);
294
295         -- The first horizontal line runs along the first row.
296         -- The last horizontal line runs along the last row.
297         -- This loop advances from one row to the next and
298         -- draws a horizontal line:
299         while MP1.y <= y2 loop
300             CP1 := real_to_canvas (MP1, S);
301             CP2 := real_to_canvas (MP2, S);
302
```

```
303                 move_to (context,
304                     to_gdouble (CP1.x), to_gdouble (CP1.y));
305
306                 line_to (context,
307                     to_gdouble (CP2.x), to_gdouble (CP2.y));
308
309                 MP1.y := MP1.y + grid.spacing.y;
310                 MP2.y := MP2.y + grid.spacing.y;
311             end loop;
312         end draw_lines;
313
314
315     begin -- draw_grid
316
317         -- Draw the grid if it is enabled and if the spacing
318         -- is greater than the minimal required spacing:
319         if grid.on = GRID_ON and then
320             get_grid_spacing (grid) >= grid_spacing_min then
321
322
323             -- put_line ("draw_grid");
324             compute_first_and_last_column;
325             compute_first_and_last_row;
326
327             -- Set the color of the grid:
328             set_source_rgb (context, 0.5, 0.5, 0.5); -- gray
329
330             case grid.style is
331                 when STYLE_DOTS =>
332                     draw_dots;
333
334                 when STYLE_LINES =>
335                     draw_lines;
336             end case;
337
338             -- Since all dots or lines are
339             -- drawn with the same linewidth and color
340             -- this single stroke command is sufficient:
341             stroke;
342         end if;
343
344     end draw_grid;
345
346
347 end demo_grid;
```

9.5 Das Paket demo_canvas

Dieses Paket ist zuständig für die Leinwand:

1. Deklaration und Erzeugung der Leinwand
2. Größe der Leinwand
3. Verschiebung der Ansicht mittels des Rollfensters

9.5.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                                CANVAS                                    --
6  --                                                                          --
7  --                               S p e c                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with gtk.widget;                 use gtk.widget;
40 with gtk.drawing_area;           use gtk.drawing_area;
41 with cairo;
42 with demo_logical_pixels;        use demo_logical_pixels;
43 with demo_geometry;              use demo_geometry;
44 with demo_window_dimensions;     use demo_window_dimensions;
45
46
47 package demo_canvas is
48
49     -- This is the canvas where all the drawing takes place:
50     canvas : gtk_drawing_area;
51
52
53     -- This is the global drawing context.
54     -- It is updated by the function cb_draw_objects:
55     context : cairo.cairo_context;
56
57
58     -- Strokes the global context (see above):
59     procedure stroke;
60
```

```
61
62      -- This is the size of the canvas in device pixels.
63      -- It is set by procedure compute_canvas_size on system startup:
64      canvas_size : type_window_size;
65
66
67      -- This procedure should be called in order to
68      -- explicitley schedule
69      -- a refresh (or redraw) of the canvas.
70      -- It calls the procedure queue_draw, which in turn
71      -- issues the "on_draw"-signal. The "on_draw"-signal
72      -- in turn triggers the callback procedure "cb_draw" (see
73      -- package demo_callbacks):
74      procedure refresh;
75
76
77      -- This procedure computes the dimensions of the canvas
78      -- and assigns them to variable canvas_size.
79      -- The computation bases on the maximum allowed width
80      -- and height of the bounding-box and the maximal
81      -- zoom factor.
82      -- CS: The computed dimensions may be not sufficient
83      -- with very very large screens:
84      procedure compute_canvas_size;
85
86
87      -- This procedure outputs the current dimensions
88      -- of the canvas on the console:
89      procedure show_canvas_size;
90
91
92      -- This procedure creates the canvas, sets its size,
93      -- and adds it into the scrolled window.
94      -- It adds events the canvas is to respond to.
95      -- It also inserts the scrolled window (swin) into box_h:
96      procedure create_canvas;
97
98
99      -- Shifts the scrolled window into the given direction
100     -- by the given distance.
101     -- If the scrolled window moves to the right, then the
102     -- drawing area on the right becomes visible. At the same
103     -- time drawing area on the left becomes invisible.
104     -- Likewise, this applies if the scrolled window is moved left,
105     -- down or up:
106     procedure shift_swin (
107         direction   : type_direction;
108         distance    : type_distance_model);
109
110 end demo_canvas;
```

9.5.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                                CANVAS                                    --
6  --                                                                          --
7  --                                B o d y                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with gdk.event;
40 with glib;
41 with ada.text_io;                use ada.text_io;
42
43 with demo_main_window;
44 with demo_scrolled_window;
45 with demo_bounding_box;
46 with demo_zoom;                  use demo_zoom;
47
48
49 package body demo_canvas is
50
51     procedure stroke is begin
52         cairo.stroke (context);
53     end stroke;
54
55
56     procedure refresh is begin
57         -- Uncomment the next statement in order
58         -- to monitor when this procedure is called:
59
60         -- put_line ("refresh");
61
62         canvas.queue_draw;
63     end refresh;
64
65
66     procedure compute_canvas_size is
67         use demo_logical_pixels;
68         use demo_bounding_box;
69
70         debug : boolean := true;
71
72         -- The maximal base-offset:
73         F_max : type_logical_pixels_vector;
74
```

```
75          -- The maximum zoom factor:
76          S_max : constant type_logical_pixels :=
77              type_logical_pixels (type_zoom_factor'last);
78
79          -- The maximum width and height of the bounding-box:
80          Bw : constant type_logical_pixels :=
81              type_logical_pixels (bounding_box_width_max);
82
83          Bh : constant type_logical_pixels :=
84              type_logical_pixels (bounding_box_height_max);
85      begin
86          if debug then
87              put_line ("compute_canvas_size");
88              put_line ("␣S_max␣:␣" & to_string (S_max));
89              put_line ("␣Bw_max:␣" & to_string (Bw));
90              put_line ("␣Bh_max:␣" & to_string (Bh));
91          end if;
92
93          -- compute the maximal base-offset:
94          F_max.x :=   Bw * (S_max - 1.0);
95          F_max.y := - Bh * S_max;
96
97          if debug then
98              put_line ("␣F_max␣:␣" & to_string (F_max));
99          end if;
100
101         -- compute the canvas width and height:
102         canvas_size.width  := positive (  F_max.x + Bw * S_max);
103         canvas_size.height := positive (- F_max.y + Bh * (S_max - 1.0));
104
105         if debug then
106             put_line ("␣Cw␣␣␣␣:␣" & positive'image (canvas_size.width));
107             put_line ("␣Ch␣␣␣␣:␣" & positive'image (canvas_size.height));
108         end if;
109     end compute_canvas_size;
110
111
112
113
114     procedure show_canvas_size is
115         use glib;
116         a : gtk_allocation;
117         width, height : gint;
118     begin
119         canvas.get_allocation (a);
120         put_line ("canvas␣size␣allocated␣(w/h):"
121             & gint'image (a.width) & "␣/" & gint'image (a.height));
122
123         canvas.get_size_request (width, height);
124         put_line ("canvas␣size␣minimum␣␣␣(w/h):"
125             & gint'image (width) & "␣/" & gint'image (height));
126     end show_canvas_size;
127
128
129
130     procedure create_canvas is
131         use demo_main_window;
132         use demo_scrolled_window;
133         use gdk.event;
134         use glib;
135     begin
136         -- Set up the drawing area:
137         gtk_new (canvas);
138
139         -- Set the size (width and height) of the canvas:
140         canvas.set_size_request (
141             gint (canvas_size.width), gint (canvas_size.height));
142
143         show_canvas_size;
144
145         -- Make the canvas responding to mouse button clicks:
146         canvas.add_events (gdk.event.button_press_mask);
147         canvas.add_events (gdk.event.button_release_mask);
148
149         -- Make the canvas responding to mouse movement:
150         canvas.add_events (gdk.event.pointer_motion_mask);
```

```
151
152         -- Make the canvas responding to the mouse wheel:
153         canvas.add_events (gdk.event.scroll_mask);
154
155         -- Make the canvas responding to the keyboard:
156         canvas.set_can_focus (true);
157         canvas.add_events (key_press_mask);
158
159
160         -- Add the canvas as a child to the scrolled window:
161         put_line ("add␣canvas␣to␣scrolled␣window");
162         swin.add (canvas);
163
164         -- Insert the scrolled window in box_h:
165         put_line ("add␣scrolled␣window␣to␣box_h");
166         box_h.pack_start (swin);
167         -- The argument "expand" is true by default which
168         -- makes swin changing its size along with box_h.
169
170     end create_canvas;
171
172
173
174     procedure shift_swin (
175         direction   : type_direction;
176         distance    : type_distance_model)
177     is
178         use demo_scrolled_window;
179
180         -- Convert the given model distance to
181         -- a canvas distance:
182         d : constant type_logical_pixels :=
183             type_logical_pixels (distance) * type_logical_pixels (S);
184
185         -- Scratch values for upper limit, lower limit and value
186         -- of scrollbars:
187         u, l, v : type_logical_pixels;
188     begin
189         case direction is
190             when DIR_RIGHT =>
191                 -- Get the maximal allowed value for the
192                 -- horizontal scrollbar:
193                 u := to_lp (scrollbar_h_adj.get_upper);
194
195                 -- Compute the required scrollbar position:
196                 v := to_lp (scrollbar_h_adj.get_value) + d;
197
198                 -- Clip the required position if necesary,
199                 -- then apply it to the scrollbar:
200                 clip_max (v, u);
201                 scrollbar_h_adj.set_value (to_gdouble (v));
202
203
204             when DIR_LEFT =>
205                 -- Get the minimal allowed value for the
206                 -- horizontal scrollbar:
207                 l := to_lp (scrollbar_h_adj.get_lower);
208
209                 -- Compute the required scrollbar position:
210                 v := to_lp (scrollbar_h_adj.get_value) - d;
211
212                 -- Clip the required position if necesary,
213                 -- then apply it to the scrollbar:
214                 clip_min (v, l);
215                 scrollbar_h_adj.set_value (to_gdouble (v));
216
217
218             when DIR_UP =>
219                 -- Get the minimal allowed value for the
220                 -- vertical scrollbar:
221                 l := to_lp (scrollbar_v_adj.get_lower);
222
223                 -- Compute the required scrollbar position:
224                 v := to_lp (scrollbar_v_adj.get_value) - d;
225
226                 -- Clip the required position if necesary,
```

```
227                 -- then apply it to the scrollbar:
228                 clip_min (v, l);
229                 scrollbar_v_adj.set_value (to_gdouble (v));
230
231
232             when DIR_DOWN =>
233                 -- Get the maximal allowed value for the
234                 -- vertical scrollbar:
235                 u := to_lp (scrollbar_v_adj.get_upper);
236
237                 -- Compute the required scrollbar position:
238                 v := to_lp (scrollbar_v_adj.get_value) + d;
239
240                 -- Clip the required position if necesary,
241                 -- then apply it to the scrollbar:
242                 clip_max (v, u);
243                 scrollbar_v_adj.set_value (to_gdouble (v));
244
245         end case;
246
247         backup_scrollbar_settings;
248     end shift_swin;
249
250
251 end demo_canvas;
```

9.6 Das Paket demo_bounding_box

Wie der Name des Pakets schon andeutet, geht es hier um das Begrenzungsrechteck B des Modelles:

1. Deklaration, Grenzwerte und Größe
2. Berechnung des Begrenzungsrechtecks B

9.6.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                             BOUNDING BOX                                 --
6  --                                                                          --
7  --                                S p e c                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with demo_geometry;             use demo_geometry;
40
41
42 package demo_bounding_box is
43
44     -- This is the bounding-box of the model. It is a rectangle
45     -- that encloses all objects of the model and the margins
46     -- around the model:
47     bounding_box : type_area;
48
49
50     -- These are the system limits for the width and height
51     -- of the bounding-box of the model:
52     bounding_box_width_max  : constant
53         type_distance_model_positive := 2_000.0;
54
55     bounding_box_height_max : constant
56         type_distance_model_positive := 1_000.0;
57
58
59
60     -- Indicates that the bounding_box has changed after calling procedure
61     -- compute_bounding_box:
```

```
62      bounding_box_changed : boolean := false;
63
64
65      -- In order to handle bouding-box related errors this
66      -- composite type is required:
67      type type_bounding_box_error is record
68          size_exceeded   : boolean := false;
69          width           : type_distance_model_positive := 0.0;
70          height          : type_distance_model_positive := 0.0;
71          -- CS ? position : type_vector_model;
72      end record;
73
74
75      -- Here we store bounding-box related errors:
76      bounding_box_error : type_bounding_box_error;
77
78
79
80      -- This procedure parses the whole database of model objects
81      -- and the primitive objects of the drawing frame,
82      -- detects the smallest and greatest x and y values used by the model
83      -- and sets the global variable bounding_box accordingly.
84      -- If the bounding_box has changed, then the flag bounding_box_changed is
85      -- set (See below).
86      --
87      -- It modifies following global veriables:
88      -- - bounding_box
89      -- - bounding_box_changed
90      -- - bounding_box_error
91      --
92      -- The arguments can be used to:
93      -- - Abort on first error. Means NOT to parse the whole database but to
94      --   abort the parsing on the first violation of the maximal allowed
95      --   dimensions (width and height).
96      -- - Ignore errors. Means to generate a bounding-box that might be
97      --   wider or taller than actually allowed. This is useful for debugging
98      --   and testing the effects of violations of maximal bounding-box
99      --   dimensions.
100     -- - Test only. Means to simulate the compuation of the bounding-box only.
101     --   The global variable bounding_box will NOT be touched in any case.
102     procedure compute_bounding_box (
103         abort_on_first_error    : in boolean := false;
104         -- CS currently not implemented
105
106         ignore_errors           : in boolean := false;
107         test_only               : in boolean := false);
108
109
110 end demo_bounding_box;
```

9.6.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                             BOUNDING BOX                                 --
6  --                                                                          --
7  --                                B o d y                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;                use ada.text_io;
40 with demo_objects;               use demo_objects;
41 with demo_frame;
42
43
44 package body demo_bounding_box is
45
46
47     procedure compute_bounding_box (
48         abort_on_first_error    : in boolean := false;
49         ignore_errors           : in boolean := false;
50         test_only               : in boolean := false)
51     is
52         use pac_lines;
53         use pac_circles;
54         use pac_objects;
55
56         -- debug : boolean := false;
57         debug : boolean := true;
58
59
60         -- In order to detect whether the bounding-box has
61         -- changed we take a copy of the current bounding-box:
62         bbox_old : type_area := bounding_box;
63
64         -- This is the temporary bounding-box we are going to build
65         -- in the course of this procedure:
66         bbox_new : type_area;
67
68         -- The first primitie object encountered will be the
69         -- seed for bbox_new. All other objects cause
70         -- this bbox_new to expand. After the first object,
71         -- this flag is cleared:
72         first_object : boolean := true;
73
74
```

```
75          -- This procedure iterates through all primitive objects
76          -- of the drawing frame and adds them to the temporary
77          -- bounding-box bbox_new:
78          procedure parse_drawing_frame is
79              use demo_frame;
80
81              procedure query_line (l : in pac_lines.cursor) is
82                  -- The candidate line being handled:
83                  line : type_line renames element (l);
84
85                  -- Compute the preliminary bounding-box of the line:
86                  b : type_area := get_bounding_box (line);
87              begin
88                  -- Move the box by the position of the
89                  -- drawing frame to get the final bounding-box
90                  -- of the line candidate:
91                  move_by (b.position, drawing_frame.position);
92
93                  -- If this is the first primitive object,
94                  -- then use its bounding-box as seed to start from:
95                  if first_object then
96                      bbox_new := b;
97                      first_object := false;
98                  else
99                  -- Otherwise, merge the box b with the box being built:
100                     merge_areas (bbox_new, b);
101                 end if;
102             end query_line;
103
104
105         begin
106             -- CS: This simple solution iterates through all lines
107             -- incl. the title block. But since the title block
108             -- is always inside the drawing frame, the elements
109             -- of the title block can be omitted.
110             drawing_frame.lines.iterate (query_line'access);
111
112             -- CS texts
113         end parse_drawing_frame;
114
115
116
117         -- This procedure is called each time an object of the database
118         -- is processed:
119         procedure query_object (oc : in pac_objects.cursor) is
120             -- This is the complex candidate object being handled:
121             object : type_complex_object renames element (oc);
122
123
124             -- This procedure computes the bounding-box of a line:
125             procedure query_line (lc : in pac_lines.cursor) is
126                 -- The candidate line being handled:
127                 line : type_line renames element (lc);
128
129                 -- Compute the preliminary bounding-box of the line:
130                 b : type_area := get_bounding_box (line);
131             begin
132                 -- Move the box by the position of the
133                 -- complex object to get the final bounding-box
134                 -- of the line candidate:
135                 move_by (b.position, object.position);
136
137                 -- If this is the first primitive object,
138                 -- then use its bounding-box as seed to start from:
139                 if first_object then
140                     bbox_new := b;
141                     first_object := false;
142                 else
143                 -- Otherwise, merge the box b with the box being built:
144                     merge_areas (bbox_new, b);
145                 end if;
146             end query_line;
147
148
149             -- This procedure computes the bounding-box of a circle:
150             procedure query_circle (cc : in pac_circles.cursor) is
```

```
151                 -- The candidate circle being handled:
152                 circle : type_circle renames element (cc);
153
154                 -- Compute the preliminary bounding-box of the circle:
155                 b : type_area := get_bounding_box (circle);
156             begin
157                 -- Move the box by the position of the
158                 -- complex object to get the final bounding-box
159                 -- of the circle candidate:
160                 move_by (b.position, object.position);
161
162                 -- If this is the first primitive object,
163                 -- then use its bounding-box as seed to start from:
164                 if first_object then
165                     bbox_new := b;
166                     first_object := false;
167                 else
168                 -- Otherwise, merge the box b with the box being built:
169                     merge_areas (bbox_new, b);
170                 end if;
171             end query_circle;
172
173
174         begin
175             -- Iterate the lines, circles and other primitive
176             -- components of the current object:
177             object.lines.iterate (query_line'access);
178             object.circles.iterate (query_circle'access);
179             -- CS arcs
180         end query_object;
181
182
183         -- This procedure updates the bounding-box and
184         -- sets the bounding_box_changed flag
185         -- in NON-TEST-MODE (which is default by argument test_only).
186         -- In TEST-mode the bounding_box_changed flag is cleared:
187         procedure update_global_bounding_box is begin
188             if test_only then
189                 put_line ("TEST␣ONLY␣mode.␣Bounding-box␣not␣changed.");
190                 bounding_box_changed := false;
191             else
192                 -- Update the global bounding-box:
193                 bounding_box := bbox_new;
194
195                 -- The new bounding-box differs from the old one.
196                 -- Set the global flag bounding_box_changed:
197                 bounding_box_changed := true;
198             end if;
199         end update_global_bounding_box;
200
201
202
203         procedure add_margin is
204             use demo_frame;
205
206             -- The offset due to the margin:
207             margin_offset : type_vector_model;
208         begin
209             bbox_new.width  := bbox_new.width  + 2.0 * margin;
210             bbox_new.height := bbox_new.height + 2.0 * margin;
211
212             -- Since we regard the margin as inside the bounding-box,
213             -- we must move the bounding-box position towards bottom-left
214             -- by the inverted margin_offset:
215             margin_offset := (x => margin, y => margin);
216             move_by (bbox_new.position, invert (margin_offset));
217         end add_margin;
218
219
220     begin
221         put_line ("compute_bounding_box");
222
223         -- The drawing frame is regarded as part of the model.
224         -- Iterate through all primitive objects of the
225         -- drawing frame:
226         parse_drawing_frame;
```

```
227
228         -- The database that contains all objects of the model
229         -- must be parsed. This is the call of an iteration through
230         -- all objects of the database:
231         objects_database_model.iterate (query_object'access);
232
233         -- The temporary bounding-box bbox_new in its current
234         -- state is the so called "inner bounding-box" (IB).
235
236         -- Now, we expand the temporary bounding-box by the margin.
237         -- The area around the drawing frame frame is regarded
238         -- as part of the model and thus inside the bounding-box:
239         add_margin;
240         -- Now, bbox_new has become the "outer bounding-box" (OB).
241
242         -- Compare the new bounding-box with the old
243         -- bounding-box to detect a change:
244         if bbox_new /= bbox_old then
245
246             -- Do the size check of the new bounding-box. If it is
247             -- too large, then restore the old bounding-box:
248             if bbox_new.width  >= bounding_box_width_max or
249                 bbox_new.height >= bounding_box_height_max then
250
251                 -- output limits and computed box dimensions:
252                 put_line ("WARNING: Bounding-box size limit exceeded !");
253                 put_line (" max. width : "
254                     & to_string (bounding_box_width_max));
255
256                 put_line (" max. height : "
257                     & to_string (bounding_box_height_max));
258
259                 put_line (" detected    : "
260                     & to_string (bbox_new));
261
262                 -- Set the error flag:
263                 bounding_box_error := (
264                     size_exceeded => true,
265                     width  => bbox_new.width,
266                     height => bbox_new.height);
267
268
269                 if ignore_errors then
270                     put_line (" Errors ignored !");
271
272                     -- Override old global bounding-box with
273                     -- the faulty box bbox_new:
274                     update_global_bounding_box;
275
276                 else -- By default errors are NOT ignored.
277                     put_line (" Discarded. Global bounding-box NOT changed.");
278
279                     -- Clear the global flag bounding_box_changed
280                     -- because we discard the new bounding-box (due to
281                     -- a size error) and
282                     -- leave the current global bounding-box untouched:
283                     bounding_box_changed := false;
284
285                 end if;
286
287
288             else -- size ok, no errors
289                 -- Reset error flag:
290                 bounding_box_error := (others => <>);
291
292                 update_global_bounding_box;
293             end if;
294
295
296         else -- No change.
297             -- Clear the global flag bounding_box_changed:
298             bounding_box_changed := false;
299
300             -- Reset error flag:
301             bounding_box_error := (others => <>);
302         end if;
```

```
303
304
305         if debug then
306             put_line ("bounding-box: " & to_string (bounding_box));
307
308             if bounding_box_changed then
309                 put_line (" has changed");
310             end if;
311         end if;
312     end compute_bounding_box;
313
314
315 end demo_bounding_box;
```

9.7 Das Paket demo_base_offset

Dieses Paket ist zuständig für die Berechnung des Basis-Offsets F.

9.7.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                              BASE OFFSET                                 --
6  --                                                                          --
7  --                               S p e c                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --    For correct displaying set tab width in your editor to 4.
27
28 --    The two letters "CS" indicate a "construction site" where things are not
29 --    finished yet or intended for the future.
30
31 --    Please send your questions and comments to:
32 --
33 --    info@blunk-electronic.de
34 --    or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --    history of changes:
37 --
38
39 with demo_logical_pixels;           use demo_logical_pixels;
40
41
42 package demo_base_offset is
43
44     -- The place on the canvas where the model
45     -- coordinates system has its origin.
46     -- It is a global variable. We call it "base-offset":
47     F : type_logical_pixels_vector;
48
49     -- Sets the global base-offset F according to the current
50     -- bounding-box and the maximal allowed zoom factor:
51     procedure set_base_offset;
52
53 end demo_base_offset;
```

9.7.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                              BASE OFFSET                                 --
6  --                                                                          --
7  --                               B o d y                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;                use ada.text_io;
40 with demo_logical_pixels;        use demo_logical_pixels;
41 with demo_bounding_box;          use demo_bounding_box;
42 with demo_canvas;                use demo_canvas;
43 with demo_zoom;                  use demo_zoom;
44
45 package body demo_base_offset is
46
47
48     procedure set_base_offset is
49         debug : boolean := false;
50
51         x, y : type_logical_pixels;
52
53         -- The maximum zoom factor:
54         S_max : constant type_logical_pixels :=
55             type_logical_pixels (type_zoom_factor'last);
56
57         -- The width and height of the bounding-box:
58         Bh : constant type_logical_pixels :=
59             type_logical_pixels (bounding_box.height);
60
61         Bw : constant type_logical_pixels :=
62             type_logical_pixels (bounding_box.width);
63
64     begin
65         x :=   Bw * (S_max - 1.0);
66         y := - Bh * S_max;
67
68         -- Set the base-offset:
69         F := (x, y);
70
71         -- Output a warning if the base-offset is outside
72         -- the canvas dimensions:
73         if  x >   type_logical_pixels (canvas_size.width) or
74             y < - type_logical_pixels (canvas_size.height) then
```

```
75
76              put_line ("WARNING: base-offset outside canvas !");
77              put_line ("␣F:␣" & to_string (F));
78          end if;
79
80
81          if debug then
82              put_line ("base␣offset:␣" & to_string (F));
83          end if;
84      end set_base_offset;
85
86
87  end demo_base_offset;
```

9.8 Das Paket demo_callbacks

Immer wenn der Bediener ein Bedienelement - wie einen Knopf oder einen Rollbalken - betätigt, wird eine sogenannte Rückrufroutine in diesem Paket aufgerufen. Darum haben alle diese Routinen auch das Prefix "cb_". Darüber hinaus sind im Körper des Paketes noch Prozeduren zur Einrichtung des Hauptfensters, des Rollfensters, der Rollbalken, der Leinwand und der Knöpfe untergebracht.

9.8.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                    CALLBACK FUNCTIONS AND PROCEDURES                     --
6  --                                                                          --
7  --                               S p e c                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with glib;                          use glib;
40 with gdk.event;                     use gdk.event;
41 with gtk.window;                    use gtk.window;
42 with gtk.widget;                    use gtk.widget;
43 with gtk.button;                    use gtk.button;
44 with gtk.adjustment;                use gtk.adjustment;
45 with cairo;                         use cairo;
46
47 with demo_buttons;
48 with demo_logical_pixels;           use demo_logical_pixels;
49 with demo_geometry;                 use demo_geometry;
50 with demo_window_dimensions;        use demo_window_dimensions;
51 with demo_canvas;                   use demo_canvas;
52 with demo_main_window;              use demo_main_window;
53 with demo_zoom;                     use demo_zoom;
54
55
56 package demo_callbacks is
57
58
59     -- This callback procedure is called each time the
60     -- button "zoom fit" is clicked.
61     procedure cb_zoom_to_fit (
```

```
62          button : access gtk_button_record'class);
63
64
65      -- This callback procedure is called each time the
66      -- button "zoom area" is clicked.
67      procedure cb_zoom_area (
68          button : access gtk_button_record'class);
69
70
71      -- This callback procedure is called each time the
72      -- button "add" is clicked.
73      procedure cb_add (
74          button : access gtk_button_record'class);
75
76
77      -- This callback procedure is called each time the
78      -- button "delete is clicked.
79      procedure cb_delete (
80          button : access gtk_button_record'class);
81
82
83      -- This callback procedure is called each time the
84      -- button "move" is clicked.
85      procedure cb_move (
86          button : access gtk_button_record'class);
87
88
89      -- This callback procedure is called each time the
90      -- button "export" is clicked:
91      procedure cb_export (
92          button : access gtk_button_record'class);
93
94
95
96
97  -- MAIN WINDOW:
98
99      -- This procedure is called when the operator terminates
100     -- the demo program by clicking the X in the upper right corner
101     -- of the main window:
102     procedure cb_terminate (
103         window : access gtk_widget_record'class);
104
105
106     procedure cb_window_focus (
107         window : access gtk_window_record'class);
108
109
110     -- This function is called each time the operator
111     -- presses a mouse button.
112     function cb_window_button_pressed (
113         window  : access gtk_widget_record'class;
114         event   : gdk_event_button)
115         return boolean;
116
117
118     -- This function is called each time the operator
119     -- hits a key on the keyboard. It does not matter
120     -- which widget inside the main window has the focus.
121     -- This callback function is at the top of the event-chain.
122     -- It is called at first on a key-press event.
123     -- If it returns false, then it signals to the
124     -- next widget in the chain downwards to handle the event
125     -- further.
126     -- The return should depend on the severity of the key.
127     -- For example in case of an "emergency-exit"
128     -- the operator hits the ESC key, which causes the abort of
129     -- all pending operations. In this case the return would be true
130     -- and the event would not be passed on to any widgets down
131     -- the chain.
132     function cb_window_key_pressed (
133         window  : access gtk_widget_record'class;
134         event   : gdk_event_key)
135         return boolean;
136
137
```

```
138
139     -- This callback procedure is called each time the
140     -- size_allocate signal is emitted by the main window:
141     procedure cb_main_window_size_allocate (
142         window      : access gtk_widget_record'class;
143         allocation  : gtk_allocation);
144
145
146     -- This procedure instantiates the main window,
147     -- sets the title bar, connects signals with callback
148     -- subprograms and creates boxes inside the window:
149     procedure set_up_main_window;
150
151
152
153
154
155 -- SCROLLED WINDOW AND SCROLLBARS:
156
157
158     -- This callback procedure is called each time the size_allocate signal
159     -- is emitted by the scrolled window.
160     procedure cb_swin_size_allocate (
161         swin        : access gtk_widget_record'class;
162         allocation  : gtk_allocation);
163
164
165     -- This procedure creates the scrolled window,
166     -- assigns to it the initial size (widht and height)
167     -- and sets the behaviour of them.
168     -- It also connects the signals emitted by the scrollbars
169     -- with the callback subprograms.
170     procedure set_up_swin_and_scrollbars;
171
172
173
174     -- This procedure is called whenever the horizontal scrollbar is moved,
175     -- either by the operator or by internal calls.
176     procedure cb_horizontal_moved (
177         scrollbar : access gtk_adjustment_record'class);
178
179
180     -- This procedure is called whenever the vertical scrollbar is moved,
181     -- either by the operator or by internal calls.
182     procedure cb_vertical_moved (
183         scrollbar : access gtk_adjustment_record'class);
184
185
186     -- This procedure is called when the operator clicks
187     -- on the vertical scrollbar:
188     function cb_scrollbar_v_pressed (
189         bar     : access gtk_widget_record'class;
190         event   : gdk_event_button)
191         return boolean;
192
193
194     -- This procedure is called when the operator releases
195     -- the mouse button after clicking on the vertical scrollbar:
196     function cb_scrollbar_v_released (
197         bar     : access gtk_widget_record'class;
198         event   : gdk_event_button)
199         return boolean;
200
201
202     -- This procedure is called when the operator clicks
203     -- on the horizontal scrollbar:
204     function cb_scrollbar_h_pressed (
205         bar     : access gtk_widget_record'class;
206         event   : gdk_event_button)
207         return boolean;
208
209
210     -- This procedure is called when the operator releases
211     -- the mouse button after clicking on the horizontal scrollbar:
212     function cb_scrollbar_h_released (
213         bar     : access gtk_widget_record'class;
```

```
214         event   : gdk_event_button)
215         return boolean;
216
217
218
219 -- CANVAS:
220
221     -- This procedure is called when the canvas changes
222     -- its size. It is not used currently because the canvas
223     -- has a fixed size in this demo program (see below):
224     procedure cb_canvas_size_allocate (
225         canvas      : access gtk_widget_record'class;
226         allocation  : gtk_allocation);
227
228
229     -- This procedure creates the canvas, assigns to
230     -- it a fixed size and connects its signals
231     -- with callback subprograms:
232     procedure set_up_canvas;
233
234
235     -- This callback function is called each time the operator
236     -- clicks on the canvas.
237     -- It sets the focus on the canvas and moves the cursor
238     -- to the place where the operator has clicked the canvas.
239     function cb_canvas_button_pressed (
240         canvas  : access gtk_widget_record'class;
241         event   : gdk_event_button)
242         return boolean;
243
244
245     -- This callback function is called each time the operator
246     -- releases a mouse button after clicking on the canvas.
247     function cb_canvas_button_released (
248         canvas  : access gtk_widget_record'class;
249         event   : gdk_event_button)
250         return boolean;
251
252
253     -- This callback function is called each time the operator
254     -- moves the pointer (or the mouse) inside the canvas:
255     function cb_canvas_mouse_moved (
256         canvas  : access gtk_widget_record'class;
257         event   : gdk_event_motion)
258         return boolean;
259
260
261     -- This callback function is called each time the
262     -- operator hits a key and if the canvas has the focus:
263     function cb_canvas_key_pressed (
264         canvas  : access gtk_widget_record'class;
265         event   : gdk_event_key)
266         return boolean;
267
268
269     -- This function is called each time the mouse wheel is
270     -- rolled inside the canvas by the operator:
271     function cb_mouse_wheel_rolled (
272         canvas  : access gtk_widget_record'class;
273         event   : gdk_event_scroll)
274         return boolean;
275
276
277     -- This function is called each time the canvas
278     -- is to be refreshed when it gets the
279     -- "on_draw"-signal. See also the body of procedure "set_up_canvas"
280     -- where the signal connection is made.
281     --
282     -- NOTE: This function is also called by other signals, such as
283     -- "grab_focus". The corresponding connection is active by default.
284     --
285     -- It draws everything: frame, grid, cursor, objects
286     function cb_draw (
287         canvas      : access gtk_widget_record'class;
288         context_in  : in cairo_context)
289         return boolean;
```

```
290
291
292 end demo_callbacks;
```

9.8.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                     CALLBACK FUNCTIONS AND PROCEDURES                    --
6  --                                                                          --
7  --                               B o d y                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with gdk.types;
40 with gdk.types.keysyms;
41 with gtk.accel_group;
42 with gtk.enums;                  use gtk.enums;
43 with gtk.misc;                   use gtk.misc;
44
45 with ada.text_io;                use ada.text_io;
46 with ada.calendar;               use ada.calendar;
47 with ada.calendar.formatting;    use ada.calendar.formatting;
48
49 with gtk.main;                   use gtk.main;
50
51 with demo_grid;
52 with demo_frame;
53 with demo_scale;
54 with demo_conversions;           use demo_conversions;
55 with demo_base_offset;           use demo_base_offset;
56 with demo_translate_offset;      use demo_translate_offset;
57 with demo_scrolled_window;       use demo_scrolled_window;
58 with demo_visible_area;
59 with demo_coordinates_display;   use demo_coordinates_display;
60 with demo_drawing_origin;
61 with demo_cursor;                use demo_cursor;
62 with demo_objects;               use demo_objects;
63
64 package body demo_callbacks is
65
66
67     procedure cb_zoom_to_fit (
68         button : access gtk_button_record'class)
69     is
70         -- debug : boolean := true;
71         debug : boolean := false;
72     begin
73         put_line ("cb_zoom_to_fit");
74
```

```
75          zoom_to_fit_all;
76      end cb_zoom_to_fit;
77
78
79
80      procedure cb_zoom_area (
81          button : access gtk_button_record'class)
82      is
83          -- debug : boolean := true;
84          debug : boolean := false;
85      begin
86          put_line ("cb_zoom_area");
87
88          zoom_area.active := true;
89      end cb_zoom_area;
90
91
92
93      procedure cb_add (
94          button : access gtk_button_record'class)
95      is begin
96          put_line ("cb_add");
97          add_object;
98
99          -- Redraw the canvas:
100         refresh;
101     end cb_add;
102
103
104     procedure cb_delete (
105         button : access gtk_button_record'class)
106     is begin
107         put_line ("cb_delete");
108         delete_object;
109
110         -- Redraw the canvas:
111         refresh;
112     end cb_delete;
113
114
115     procedure cb_move (
116         button : access gtk_button_record'class)
117     is begin
118         put_line ("cb_move");
119         -- CS
120     end cb_move;
121
122
123     procedure cb_export (
124         button : access gtk_button_record'class)
125     is
126     begin
127         put_line ("cb_export");
128         -- CS
129     end cb_export;
130
131
132
133
134
135 -- MAIN WINDOW:
136
137     procedure cb_terminate (
138         window : access gtk_widget_record'class)
139     is begin
140         put_line ("cb_terminate");
141         gtk.main.main_quit;
142     end cb_terminate;
143
144
145     procedure cb_window_focus (
146         window : access gtk_window_record'class)
147     is begin
148         put_line ("cb_window_focus");
149     end cb_window_focus;
150
```

```
151
152     function cb_window_button_pressed (
153         window  : access gtk_widget_record'class;
154         event   : gdk_event_button)
155         return boolean
156     is
157         use glib;
158         event_handled : boolean := true;
159
160         -- The point where the operator has clicked:
161         point : constant type_logical_pixels_vector :=
162             (to_lp (event.x), to_lp (event.y));
163     begin
164         null;
165
166         -- Output the button id, x and y position:
167         put_line ("cb_window_button_pressed "
168             & "button" & guint'image (event.button) & " "
169             & to_string (point));
170
171         return event_handled;
172     end cb_window_button_pressed;
173
174
175
176     function cb_window_key_pressed (
177         window  : access gtk_widget_record'class;
178         event   : gdk_event_key)
179         return boolean
180     is
181         event_handled : boolean := false;
182
183         use gdk.types;
184         use gdk.types.keysyms;
185
186         key_ctrl    : gdk_modifier_type := event.state and control_mask;
187         key_shift   : gdk_modifier_type := event.state and shift_mask;
188         key         : gdk_key_type := event.keyval;
189
190     begin
191         -- Output the the gdk_key_type (which is
192         -- just a number (see gdk.types und gdk.types.keysyms)):
193         put_line ("cb_window_key_pressed "
194             & " key " & gdk_key_type'image (event.keyval));
195
196         if key_ctrl = control_mask then
197             case key is
198
199                 when others => null;
200             end case;
201
202         else
203             case key is
204                 when GDK_ESCAPE =>
205                     -- Here the commands to abort any pending
206                     -- operations should be placed:
207
208                     -- Abort the zoom-to-area operation:
209                     reset_zoom_area;
210
211                     -- Do not pass this event further
212                     -- to widgets down the chain.
213                     -- Prosssing the event stops here.
214                     event_handled := true;
215
216
217                 when GDK_F5 =>
218                     zoom_to_fit_all;
219
220                     -- Do not pass this event further
221                     -- to widgets down the chain.
222                     -- Prosssing the event stops here.
223                     event_handled := true;
224
225
226                 when others => null;
```

```
227             end case;
228         end if;
229
230         return event_handled;
231     end cb_window_key_pressed;
232
233
234
235
236     procedure cb_main_window_size_allocate (
237         window      : access gtk_widget_record'class;
238         allocation  : gtk_allocation)
239     is
240     begin
241         null;
242         -- put_line ("cb_main_window_size_allocate " & image (clock));
243
244         -- put_line ("cb_window_size_allocate. (x/y/w/h): "
245         --  & gint'image (allocation.x)
246         --  & " /" & gint'image (allocation.y)
247         --  & " /" & gint'image (allocation.width)
248         --  & " /" & gint'image (allocation.height));
249     end cb_main_window_size_allocate;
250
251
252
253     function cb_main_window_configure (
254         window      : access gtk_widget_record'class;
255         event       : gdk.event.gdk_event_configure)
256         return boolean
257     is
258         result : boolean := false;
259     begin
260         -- put_line ("cb_main_window_configure " & image (clock));
261         return result;
262     end cb_main_window_configure;
263
264
265     procedure cb_main_window_realize (
266         window  : access gtk_widget_record'class)
267     is begin
268         null;
269         -- put_line ("cb_main_window_realize " & image (clock));
270     end cb_main_window_realize;
271
272
273     function cb_main_window_state_change (
274         window      : access gtk_widget_record'class;
275         event       : gdk.event.gdk_event_window_state)
276         return boolean
277     is
278         result : boolean := false;
279     begin
280         -- put_line ("cb_main_window_state_change " & image (clock));
281         return result;
282     end cb_main_window_state_change;
283
284
285     procedure cb_main_window_activate (
286         window      : access gtk_window_record'class)
287     is begin
288         null;
289         -- put_line ("cb_main_window_activate " & image (clock));
290     end cb_main_window_activate;
291
292
293     procedure set_up_command_buttons is
294         use demo_buttons;
295     begin
296         put_line ("set_up_command_buttons");
297
298         create_buttons;
299
300         -- Connect button signals with subprograms:
301         button_zoom_fit.on_clicked (cb_zoom_to_fit'access);
302         button_zoom_area.on_clicked (cb_zoom_area'access);
```

```
303         button_add.on_clicked (cb_add'access);
304         button_delete.on_clicked (cb_delete'access);
305         button_move.on_clicked (cb_move'access);
306         button_export.on_clicked (cb_export'access);
307
308     end set_up_command_buttons;
309
310
311
312     procedure set_up_main_window is begin
313         put_line ("set_up_main_window");
314
315         create_window; -- incl. boxes and a separator
316
317         -- connect signals:
318         main_window.on_destroy (cb_terminate'access);
319         main_window.on_size_allocate (cb_main_window_size_allocate'access);
320         main_window.on_button_press_event (cb_window_button_pressed'access);
321         main_window.on_key_press_event (cb_window_key_pressed'access);
322         main_window.on_configure_event (cb_main_window_configure'access);
323
324         main_window.on_window_state_event (
325             cb_main_window_state_change'access);
326
327         main_window.on_realize (cb_main_window_realize'access);
328         main_window.on_activate_default (cb_main_window_activate'access);
329
330         -- Not used:
331         -- main_window.on_activate_focus (cb_window_focus'access);
332
333         set_up_command_buttons;
334
335     end set_up_main_window;
336
337
338
339
340
341     procedure cb_swin_size_allocate (
342         swin        : access gtk_widget_record'class;
343         allocation  : gtk_allocation)
344     is
345         use demo_visible_area;
346
347         -- Each time ths procedure is called, the argument "allocation"
348         -- provides the new size of the scrolled window. Later this size will
349         -- be compared with the old size (stored in global
350         -- variable scrolled_window_size):
351         new_size : constant type_window_size := (
352             width   => positive (allocation.width),
353             height  => positive (allocation.height));
354
355         -- This is the difference between new width and old width:
356         dW : type_logical_pixels;
357
358         -- This is the difference between new height and old height:
359         dH : type_logical_pixels;
360
361
362         -- For debugging. Outputs the dimensions and size
363         -- changes of the main window:
364         procedure show_size is begin
365             put_line ("␣size␣old␣(w/h):␣"
366                 & positive'image (swin_size.width)
367                 & "␣/" & positive'image (swin_size.height));
368
369             put_line ("␣size␣new␣(w/h):␣"
370                 & positive'image (new_size.width)
371                 & "␣/" & positive'image (new_size.height));
372
373             put_line ("␣dW␣:␣" & to_string (dW));
374             put_line ("␣dH␣:␣" & to_string (dH));
375
376             -- put_line ("S1:" & to_string (S1));
377         end show_size;
378
```

```
379
380         -- When the scrolled window is resized, then it expands away
381         -- from its top left corner or it shrinks toward its top-left
382         -- corner. In both cases the bottom of the
383         -- window moves down or up. So the bottom of the canvas must
384         -- follow the bottom of the scrolled window. This procedure
385         -- moves the bottom of the canvas by the same
386         -- extent as the bottom of the scrolled window:
387         procedure move_canvas_bottom is begin
388             -- Approach 1:
389             -- One way to move the canvas is to change the y-component
390             -- of the base-offset.
391             F.y := F.y - dh;
392
393             -- Schedule a refresh to make the size change appear smoothly:
394             refresh;
395
396             -- Approach 2: -- CS never tried
397             -- Modify the y-component of the translate-offset
398         end move_canvas_bottom;
399
400
401
402
403         -- This procedure zooms to the area, stored in last_visible_area,
404         -- so that it fits into the current scrolled window.
405         -- It is required for MODE_3_ZOOM_FIT:
406         procedure zoom_visible_area is
407             -- Get the corners of the bounding-box on the canvas before
408             -- and after zooming:
409             C1, C2 : type_bounding_box_corners;
410         begin
411             C1 := get_bounding_box_corners;
412
413             -- Reset the translate-offset:
414             T := (0.0, 0.0);
415
416             -- Fit the last visible area into the current
417             -- scrolled window:
418             zoom_to_fit (last_visible_area);
419
420             C2 := get_bounding_box_corners;
421             update_scrollbar_limits (C1, C2);
422             backup_scrollbar_settings;
423         end zoom_visible_area;
424
425
426         -- This procedure moves the canvas so that the center of the visible
427         -- area remains in the center.
428         -- This procedure is required when zoom mode MODE_KEEP_CENTER is
429         -- enabled:
430         procedure move_center is
431         begin
432             F.x := F.x + dW * 0.5;
433             F.y := F.y + dH * 0.5;
434             -- put_line ("F : " & to_string (F));
435         end move_center;
436
437
438     begin -- cb_swin_size_allocate
439
440         -- put_line ("cb_swin_size_allocate " & image (clock));
441         -- put_line ("cb_swin_size_allocate. (x/y/w/h): "
442         --  & gint'image (allocation.x)
443         --  & " /" & gint'image (allocation.y)
444         --  & " /" & gint'image (allocation.width)
445         --  & " /" & gint'image (allocation.height));
446
447         -- This procedure is called on many occasions. We are interested
448         -- only in cases where the size changes.
449         -- So we watch for changes of width and height only:
450
451         -- Compare the new size with the old size. The global variable
452         -- swin_size provides the size of the window BEFORE this
453         -- procedure has been called. If the size has changed, then proceed
454         -- with other actions. If the size has not changed, then nothing
```

```
455         -- happens:
456         if new_size /= swin_size then
457             new_line;
458             put_line ("scrolled window size changed");
459 
460             -- Opon resizing the scrolled window, the settings of the
461             -- scrollbars (upper, lower and page size) adapt to the size of
462             -- the scrolled window. But we do NOT want this behaviour.
463             -- Instead we restore the settings
464             -- as they where BEFORE this procedure has been called:
465             restore_scrollbar_settings;
466             -- show_adjustments_h;
467             -- show_adjustments_v;
468 
469             -- Compute the change of width and height
470             -- and convert it to logical pixels:
471             dW := type_logical_pixels (new_size.width - swin_size.width);
472             dH := type_logical_pixels (new_size.height - swin_size.height);
473 
474             -- for debugging:
475             -- show_size;
476 
477             -- Move the canvas so that its bottom follows
478             -- the bottom of the scrolled window:
479             move_canvas_bottom;
480 
481 
482             case zoom_mode is
483                 when MODE_1_EXPOSE_CANVAS =>
484                     null; -- nothing more to do
485 
486                 when MODE_2_KEEP_CENTER =>
487                     move_center;
488 
489                 when MODE_3_ZOOM_FIT =>
490                     zoom_visible_area;
491 
492             end case;
493 
494 
495             -- Update the swin_size which is required
496             -- for the next time this procedure is called:
497             swin_size := new_size;
498 
499         end if;
500     end cb_swin_size_allocate;
501 
502 
503 
504 
505     procedure set_up_swin_and_scrollbars is begin
506         put_line ("set_up_swin_and_scrollbars");
507 
508         create_scrolled_window_and_scrollbars;
509 
510 
511         -- connect signals:
512         swin.on_size_allocate (cb_swin_size_allocate'access);
513         -- After executing procedure cb_swin_size_allocate
514         -- the canvas is refreshed (similar to refresh (canvas))
515         -- automatically.
516 
517         -- Connect the signal "value-changed" of the scrollbars with
518         -- procedures cb_vertical_moved and cb_horizontal_moved. So the user
519         -- can watch how the signals are emitted:
520         scrollbar_v_adj.on_value_changed (cb_vertical_moved'access);
521         scrollbar_h_adj.on_value_changed (cb_horizontal_moved'access);
522 
523         scrollbar_v := swin.get_vscrollbar;
524         scrollbar_v.on_button_press_event (cb_scrollbar_v_pressed'access);
525         scrollbar_v.on_button_release_event (cb_scrollbar_v_released'access);
526 
527         scrollbar_h := swin.get_hscrollbar;
528         scrollbar_h.on_button_press_event (cb_scrollbar_h_pressed'access);
529         scrollbar_h.on_button_release_event (cb_scrollbar_h_released'access);
530 
```

```
531
532         update_cursor_coordinates;
533     end set_up_swin_and_scrollbars;
534
535
536
537
538     procedure cb_horizontal_moved (
539         scrollbar : access gtk_adjustment_record'class)
540     is begin
541         -- put_line ("horizontal moved " & image (clock));
542         -- show_adjustments_h;
543         null;
544
545         -- CS: If the following statement is commented out,
546         -- then the refresh is triggered solely implicitly by GTK.
547         -- The drawback is that the signal "on_draw" is sent
548         -- not frequently enough. Another strange effect is that
549         -- the callback procedure cb_draw does not work
550         -- properly any more. Currently there is no explanation
551         -- for this strange behaviour.
552
553         refresh;
554
555     end cb_horizontal_moved;
556
557
558     procedure cb_vertical_moved (
559         scrollbar : access gtk_adjustment_record'class)
560     is begin
561         -- put_line ("vertical moved " & image (clock));
562         -- show_adjustments_v;
563         refresh;
564     end cb_vertical_moved;
565
566
567     function cb_scrollbar_v_pressed (
568         bar     : access gtk_widget_record'class;
569         event   : gdk_event_button)
570         return boolean
571     is
572         event_handled : boolean := false;
573     begin
574         -- put_line ("cb_scrollbar_v_pressed");
575         return event_handled;
576     end cb_scrollbar_v_pressed;
577
578
579     function cb_scrollbar_v_released (
580         bar     : access gtk_widget_record'class;
581         event   : gdk_event_button)
582         return boolean
583     is
584         use demo_visible_area;
585         event_handled : boolean := false;
586     begin
587         -- put_line ("cb_scrollbar_v_released");
588         backup_scrollbar_settings;
589
590         backup_visible_area (get_visible_area (canvas));
591         return event_handled;
592     end cb_scrollbar_v_released;
593
594
595
596     function cb_scrollbar_h_pressed (
597         bar     : access gtk_widget_record'class;
598         event   : gdk_event_button)
599         return boolean
600     is
601         event_handled : boolean := false;
602     begin
603         -- put_line ("cb_scrollbar_h_pressed");
604         return event_handled;
605     end cb_scrollbar_h_pressed;
606
```

```
607
608     function cb_scrollbar_h_released (
609         bar     : access gtk_widget_record'class;
610         event   : gdk_event_button)
611         return boolean
612     is
613         use demo_visible_area;
614         event_handled : boolean := false;
615     begin
616         -- put_line ("cb_scrollbar_h_released");
617         backup_scrollbar_settings;
618
619         backup_visible_area (get_visible_area (canvas));
620         return event_handled;
621     end cb_scrollbar_h_released;
622
623
624
625
626 -- CANVAS:
627
628
629     procedure cb_canvas_size_allocate (
630         canvas      : access gtk_widget_record'class;
631         allocation  : gtk_allocation)
632     is begin
633         null;
634         -- new_line;
635         -- put_line ("cb_canvas_size_allocate");
636
637         -- put_line ("cb_canvas_size_allocate. (x/y/w/h): "
638         --  & gint'image (allocation.x)
639         --  & " /" & gint'image (allocation.y)
640         --  & " /" & gint'image (allocation.width)
641         --  & " /" & gint'image (allocation.height));
642
643     end cb_canvas_size_allocate;
644
645
646
647
648     procedure set_up_canvas is begin
649         put_line ("set_up_canvas");
650
651         create_canvas;
652
653         -- Connect signals:
654
655         -- Not used:
656         -- canvas.on_size_allocate (cb_canvas_size_allocate'access);
657         -- canvas.set_redraw_on_allocate (false);
658
659         -- Connect the canvas with the "on_draw"-signal:
660         canvas.on_draw (cb_draw'access);
661         -- NOTE: No context is declared here, because the canvas widget
662         -- passes its own context to the callback procedure cb_draw.
663
664         canvas.on_button_press_event (cb_canvas_button_pressed'access);
665         canvas.on_button_release_event (cb_canvas_button_released'access);
666         canvas.on_motion_notify_event (cb_canvas_mouse_moved'access);
667         canvas.on_scroll_event (cb_mouse_wheel_rolled'access);
668         canvas.on_key_press_event (cb_canvas_key_pressed'access);
669
670     end set_up_canvas;
671
672
673
674
675
676     function cb_canvas_button_pressed (
677         canvas  : access gtk_widget_record'class;
678         event   : gdk_event_button)
679         return boolean
680     is
681         use demo_grid;
682         use glib;
```

```
683         event_handled : boolean := true;
684
685         -- This is the point in the canvas where the operator
686         -- has clicked:
687         cp : constant type_logical_pixels_vector :=
688             (to_lp (event.x), to_lp (event.y));
689
690         -- Convert the canvas point to the corresponding
691         -- real model point:
692         mp : constant type_vector_model := canvas_to_real (cp, S);
693
694         -- CS: For some reason the value of the scrollbars
695         -- must be saved and restored if the canvas grabs the focus:
696         h, v : type_logical_pixels;
697         -- A solution might be:
698         -- <https://stackoverflow.com/questions/26693042/
699         -- gtkscrolledwindow-disable-scroll-to-focused-child>
700         -- or
701         -- <https://discourse.gnome.org/t/disable-auto-scrolling-in-
702         -- gtkscrolledwindow-when-grab-focus-in-children/13058>
703     begin
704         put_line ("cb_canvas_button_pressed");
705
706         -- Output the button id, x and y position:
707         -- put_line ("cb_canvas_button_pressed "
708         --  & " button" & guint'image (event.button) & " "
709         --  & to_string (cp));
710
711         -- Output the model point in the terminal:
712         put_line (to_string (mp));
713
714
715         -- Set the focus on the canvas,
716         -- But first save the scrollbar values:
717         h := to_lp (scrollbar_h_adj.get_value);
718         v := to_lp (scrollbar_v_adj.get_value);
719         -- CS: backup_scrollbar_settings does not work for some reason.
720         -- put_line (to_string (v));
721
722         canvas.grab_focus;
723
724         scrollbar_h_adj.set_value (to_gdouble (h));
725         scrollbar_v_adj.set_value (to_gdouble (v));
726         -- CS: restore_scrollbar_settings does not work for some reason.
727         -- put_line (to_string (v));
728
729
730         -- If no zoom-to-area operation is active, then
731         -- just place the cursor where the operator has clicked the canvas.
732         -- If the operator has started a zoom-to-area operation, then
733         -- set the first corner of the area:
734         if zoom_area.active then
735             zoom_area.k1 := mp;
736             --put_line ("zoom area k1: " & to_string (zoom_area.k1));
737
738             -- For the routine that draws a rectangle around the
739             -- selected area: Indicate that a selection has started
740             -- and a start point has been defined:
741             zoom_area.started := true;
742             zoom_area.l1 := cp;
743             --put_line ("zoom area l1: " & to_string (zoom_area.l1));
744         else
745         -- Otherwise move the cursor to the nearest grid point:
746             move_cursor (snap_to_grid (mp));
747         end if;
748
749
750         refresh;
751
752         return event_handled;
753     end cb_canvas_button_pressed;
754
755
756
757
758     function cb_canvas_button_released (
```

```
759         canvas  : access gtk_widget_record'class;
760         event   : gdk_event_button)
761         return boolean
762     is
763         use demo_visible_area;
764         use glib;
765
766         event_handled : boolean := true;
767
768         debug : boolean := false;
769
770
771         -- This is the point in the canvas where the operator
772         -- released the button:
773         cp : constant type_logical_pixels_vector :=
774             (to_lp (event.x), to_lp (event.y));
775
776         -- Convert the canvas point to the corresponding
777         -- real model point:
778         mp : constant type_vector_model := canvas_to_real (cp, S);
779
780         -- The corners of the bounding-box on the canvas before
781         -- and after zooming:
782         C1, C2 : type_bounding_box_corners;
783
784     begin
785         put_line ("cb_canvas_button_released");
786
787
788         -- Output the button id, x and y position:
789         -- put_line ("cb_canvas_button_pressed "
790         --  & " button" & guint'image (event.button) & " "
791         --  & to_string (cp));
792
793         -- Output the model point in the terminal:
794         -- put_line (to_string (mp));
795
796         -- Move the cursor to the nearest grid point:
797         -- move_cursor (snap_to_grid (mp));
798
799
800         -- If the operator is finishing a zoom-to-area operation,
801         -- then the actual area of interest is computed here
802         -- and passed to procedure zoom_to_fit.
803         -- If start and end point of the area are equal,
804         -- then nothing happens here.
805         if zoom_area.active then
806             C1 := get_bounding_box_corners;
807
808             -- Set the second corner of the zoom-area:
809             zoom_area.k2 := mp;
810
811             -- Compute the area from the corner points k1 and k2
812             -- if they are different. Otherwise nothing happens here:
813             if zoom_area.k1 /= zoom_area.k2 then
814
815                 if debug then
816                     put_line ("zoom area c1: " & to_string (zoom_area.k1));
817                     put_line ("zoom area c2: " & to_string (zoom_area.k2));
818                 end if;
819
820
821                 -- x-position:
822                 if zoom_area.k1.x < zoom_area.k2.x then
823                     zoom_area.area.position.x := zoom_area.k1.x;
824                 else
825                     zoom_area.area.position.x := zoom_area.k2.x;
826                 end if;
827
828                 -- y-position:
829                 if zoom_area.k1.y < zoom_area.k2.y then
830                     zoom_area.area.position.y := zoom_area.k1.y;
831                 else
832                     zoom_area.area.position.y := zoom_area.k2.y;
833                 end if;
834
```

```
835                 -- width and height:
836                 zoom_area.area.width  :=
837                     abs (zoom_area.k2.x - zoom_area.k1.x);
838
839                 zoom_area.area.height :=
840                     abs (zoom_area.k2.y - zoom_area.k1.y);
841
842                 if debug then
843                     put_line ("zoom " & to_string (zoom_area.area));
844                 end if;
845
846
847
848                 -- Reset the translate-offset:
849                 T := (0.0, 0.0);
850                 zoom_to_fit (zoom_area.area);
851
852                 C2 := get_bounding_box_corners;
853                 update_scrollbar_limits (C1, C2);
854                 backup_scrollbar_settings;
855
856                 -- The operation comes to an end here:
857                 zoom_area.active := false;
858
859                 -- For the routine that draws a rectangle around the
860                 -- selected area: Indicate that the rectangle shall
861                 -- no longer be drawn:
862                 zoom_area.started := false;
863
864
865                 backup_visible_area (zoom_area.area);
866             end if;
867         end if;
868
869
870         refresh;
871
872         return event_handled;
873     end cb_canvas_button_released;
874
875
876
877
878
879     function cb_canvas_mouse_moved (
880         canvas  : access gtk_widget_record'class;
881         event   : gdk_event_motion)
882         return boolean
883     is
884         use demo_coordinates_display;
885         use demo_scale;
886
887         use glib;
888         event_handled : boolean := true;
889
890         cp : constant type_logical_pixels_vector :=
891             (to_lp (event.x), to_lp (event.y));
892
893         -- Get the real model coordinates:
894         mp : constant type_vector_model := canvas_to_real (cp, S);
895     begin
896         -- put_line ("cb_canvas_mouse_moved");
897
898         -- output on the terminal:
899         -- Output the x/y position of the pointer
900         -- in logical and model coordinates:
901         -- put_line (
902             -- to_string (cp)
903             -- & " " & to_string (mp)
904
905         -- Update the coordinates display with the pointer position:
906         -- x-axis:
907         --pointer_x_buf.set_text (to_string (mp.x));
908         pointer_x_buf.set_text (to_string (to_reality (mp.x)));
909         pointer_x_value.set_buffer (pointer_x_buf);
910
```

```
911         -- y-axis:
912         pointer_y_buf.set_text (to_string (to_reality (mp.y)));
913         pointer_y_value.set_buffer (pointer_y_buf);
914         -- CS move to demo_coordinates_display
915
916         update_distances_display;
917
918         -- While a zoom-to-area operation is active,
919         -- set the end point of the selected area.
920         -- The routine that draws the rectangle uses this
921         -- point to compute the rectangle on the fly:
922         if zoom_area.active then
923             zoom_area.l2 := cp;
924             --put_line ("zoom area l2: " & to_string (zoom_area.l2));
925
926             -- The canvas must be refreshed in order to
927             -- show the rectangle as the mouse is being moved:
928             refresh;
929         end if;
930
931         return event_handled;
932     end cb_canvas_mouse_moved;
933
934
935
936
937
938     function cb_canvas_key_pressed (
939         canvas  : access gtk_widget_record'class;
940         event   : gdk_event_key)
941         return boolean
942     is
943         use demo_visible_area;
944         use demo_grid;
945
946         event_handled : boolean := true;
947
948         use gdk.types;
949         use gdk.types.keysyms;
950
951         key_ctrl    : gdk_modifier_type := event.state and control_mask;
952         key_shift   : gdk_modifier_type := event.state and shift_mask;
953         key         : gdk_key_type := event.keyval;
954
955     begin
956         -- Output the the gdk_key_type (which is
957         -- just a number (see gdk.types und gdk.types.keysyms)):
958         put_line ("cb_canvas_key_pressed␣"
959             & "␣key␣" & gdk_key_type'image (event.keyval));
960
961         if key_ctrl = control_mask then
962             case key is
963                 when GDK_KP_ADD | GDK_PLUS =>
964                     zoom_on_cursor (ZOOM_IN);
965
966                 when GDK_KP_SUBTRACT | GDK_MINUS =>
967                     zoom_on_cursor (ZOOM_OUT);
968
969                 when others => null;
970             end case;
971
972         else
973             case key is
974                 when GDK_ESCAPE =>
975                     -- Here the commands to abort any pending
976                     -- operations related to the canvas should be placed:
977
978                     null;
979
980
981                 when GDK_Right =>
982                     move_cursor (DIR_RIGHT);
983
984                 when GDK_Left =>
985                     move_cursor (DIR_LEFT);
986
```

```
987                 when GDK_Up =>
988                     move_cursor (DIR_UP);
989
990                 when GDK_Down =>
991                     move_cursor (DIR_DOWN);
992
993                 when GDK_HOME | GDK_KP_HOME =>
994                     -- Move the cursor to the grid point that
995                     -- is nearest to the center of the visible area:
996                     put_line ("move␣cursor␣to␣center");
997                     move_cursor (snap_to_grid (get_center (visible_area)));
998                     refresh;
999
1000                -- when GDK_F2 =>
1001
1002
1003                when others => null;
1004            end case;
1005        end if;
1006
1007        return event_handled;
1008    end cb_canvas_key_pressed;
1009
1010
1011
1012
1013
1014    function cb_mouse_wheel_rolled (
1015        canvas  : access gtk_widget_record'class;
1016        event   : gdk_event_scroll)
1017        return boolean
1018    is
1019        --debug : boolean := false;
1020        debug : boolean := true;
1021
1022        use glib;
1023        use gdk.types;
1024        use gtk.accel_group;
1025        event_handled : boolean := true;
1026
1027        accel_mask : constant gdk_modifier_type :=
1028            get_default_mod_mask;
1029
1030        -- The direction at which the operator is turning the wheel:
1031        wheel_direction : constant gdk_scroll_direction :=
1032            event.direction;
1033
1034
1035        procedure zoom is
1036            use demo_visible_area;
1037
1038            -- The given point on the canvas where the operator is
1039            -- zooming in or out:
1040            Z : constant type_logical_pixels_vector :=
1041                (to_lp (event.x), to_lp (event.y));
1042
1043            -- The corners of the bounding-box on the canvas before
1044            -- and after zooming:
1045            C1, C2 : type_bounding_box_corners;
1046            S1 : constant type_zoom_factor := S;
1047
1048        begin -- zoom
1049            -- put_line (" zoom old" & to_string (S));
1050
1051            C1 := get_bounding_box_corners;
1052
1053            case wheel_direction is
1054                when SCROLL_UP =>
1055                    increase_zoom_factor;
1056                    if debug then
1057                        put_line ("␣zoom␣in");
1058                    end if;
1059                    update_zoom_display;
1060
1061                when SCROLL_DOWN =>
1062                    decrease_zoom_factor;
```

```
1063                    if debug then
1064                        put_line ("␣zoom␣out");
1065                    end if;
1066                    update_zoom_display;
1067
1068                when others => null;
1069            end case;
1070
1071
1072            if debug then
1073                put_line ("␣S" & to_string (S));
1074            end if;
1075
1076            -- After changing the zoom factor, the translate-offset must
1077            -- be calculated anew. When the actual drawing takes
1078            -- place (see function cb_draw)
1079            -- then the drawing will be dragged back by the
1080            -- translate-offset so that the operator gets the impression
1081            -- of a zoom-into or zoom-out effect.
1082            -- Without applying a translate-offset the drawing would be
1083            -- appearing as expanding to the upper-right (on zoom-in)
1084            -- or shrinking toward the lower-left:
1085            set_translation_for_zoom (S1, S, Z);
1086
1087            -- show_adjustments_v;
1088            -- backup_scrollbar_settings;
1089
1090            C2 := get_bounding_box_corners;
1091            update_scrollbar_limits (C1, C2);
1092
1093            backup_visible_area (get_visible_area (canvas));
1094
1095            -- schedule a redraw:
1096            refresh;
1097        end zoom;
1098
1099
1100        procedure scroll (
1101            direction : in type_scroll_direction)
1102        is
1103            use demo_visible_area;
1104
1105            v1, dv, v2 : type_logical_pixels;
1106
1107            -- This procedure computes the amount
1108            -- by which the scrollbar value is to be changed:
1109            procedure set_delta is begin
1110                null;
1111                -- CS: This is emperical for the time being.
1112                -- Rework required.
1113                dv := 10.0 * type_logical_pixels (S);
1114            end set_delta;
1115
1116        begin
1117            if debug then
1118                put_line ("␣" & type_scroll_direction'image (direction));
1119            end if;
1120
1121
1122            case direction is
1123                when SCROLL_DOWN =>
1124                    -- Get the current value of the scrollbar:
1125                    v1 := to_lp (scrollbar_v_adj.get_value);
1126
1127                    -- Compute the amout by which the
1128                    -- scrollbar is to be moved:
1129                    set_delta;
1130
1131                    -- Compute the new value of the scrollbar:
1132                    v2 := v1 + dv;
1133
1134                    -- Set the new value of the scrollbar:
1135                    scrollbar_v_adj.set_value (to_gdouble (v2));
1136
1137
1138                when SCROLL_UP =>
```

```
1139                    v1 := to_lp (scrollbar_v_adj.get_value);
1140                    set_delta;
1141                    v2 := v1 - dv;
1142                    scrollbar_v_adj.set_value (to_gdouble (v2));
1143
1144                when SCROLL_RIGHT =>
1145                    v1 := to_lp (scrollbar_h_adj.get_value);
1146                    set_delta;
1147                    v2 := v1 + dv;
1148                    scrollbar_h_adj.set_value (to_gdouble (v2));
1149
1150                when SCROLL_LEFT =>
1151                    v1 := to_lp (scrollbar_h_adj.get_value);
1152                    set_delta;
1153                    v2 := v1 - dv;
1154                    scrollbar_h_adj.set_value (to_gdouble (v2));
1155
1156                -- CS clip ?
1157            end case;
1158
1159            backup_visible_area (get_visible_area (canvas));
1160        end scroll;
1161
1162
1163    begin -- cb_mouse_wheel_rolled
1164
1165        if debug then
1166            put_line ("cb_mouse_wheel_rolled");
1167            -- put_line (" direction "
1168            -- & gdk_scroll_direction'image (wheel_direction));
1169        end if;
1170
1171
1172        -- If CTRL is being pressed, then zoom in or out:
1173        if (event.state and accel_mask) = control_mask then
1174            zoom;
1175
1176        -- If SHIFT is being pressed, then scroll up or down:
1177        elsif (event.state and accel_mask) = shift_mask then
1178            case wheel_direction is
1179                when SCROLL_UP =>
1180                    scroll (SCROLL_RIGHT);
1181
1182                when SCROLL_DOWN =>
1183                    scroll (SCROLL_LEFT);
1184
1185                when others => null;
1186            end case;
1187
1188        -- If no key is being pressed, then scroll right or left:
1189        else
1190            case wheel_direction is
1191                when SCROLL_UP =>
1192                    scroll (SCROLL_UP);
1193
1194                when SCROLL_DOWN =>
1195                    scroll (SCROLL_DOWN);
1196
1197                when others => null;
1198            end case;
1199        end if;
1200
1201        backup_scrollbar_settings;
1202
1203        -- update_visible_area (canvas);
1204
1205        return event_handled;
1206    end cb_mouse_wheel_rolled;
1207
1208
1209
1210
1211    function cb_draw (
1212        canvas      : access gtk_widget_record'class;
1213        context_in  : in cairo_context)
1214        return boolean
```

```
1215     is
1216         event_handled : boolean := true;
1217
1218         use demo_visible_area;
1219
1220     begin
1221         -- Uncomment the next statement in order
1222         -- to monitor when this procedure is called:
1223
1224         put_line ("cb_draw " & image (clock));
1225
1226         -- Update the global context:
1227         context := context_in;
1228
1229
1230         -- Update the global visible_area:
1231         visible_area := get_visible_area (canvas);
1232         -- put_line (" visible " & to_string (visible_area));
1233
1234         -- Set the background color:
1235         -- set_source_rgb (context, 0.0, 0.0, 0.0); -- black
1236         set_source_rgb (context, 1.0, 1.0, 1.0); -- white
1237         paint (context);
1238
1239         -- The ends of all kinds of lines are round:
1240         set_line_cap (context, cairo_line_cap_round);
1241
1242         demo_grid.draw_grid;
1243         demo_drawing_origin.draw_origin;
1244         draw_cursor;
1245         draw_zoom_area;
1246         demo_frame.draw_drawing_frame;
1247         draw_objects;
1248
1249         return event_handled;
1250     end cb_draw;
1251
1252
1253 end demo_callbacks;
```

9.9 Das Paket demo_main_window

Dieses Paket ist zuständig für die Erzeugung des Hauptfensters und der Anordnung von Elementen darin.

9.9.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                              MAIN WINDOW                                 --
6  --                                                                          --
7  --                                S p e c                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with gtk.widget;                use gtk.widget;
40 with gtk.window;                use gtk.window;
41 with gtk.separator;             use gtk.separator;
42 with gtk.box;                   use gtk.box;
43
44 package demo_main_window is
45
46     main_window : gtk_window;
47
48     -- The main box that contains everything:
49     box_h               : gtk_hbox;
50
51
52     -- These items will be inserted in box_h
53     -- from the left to the right:
54     box_v1              : gtk_vbox; -- for the coordinates display
55     separator           : gtk_separator;
56     box_v0              : gtk_vbox; -- for the buttons
57
58
59     -- This procedure creates the main window and
60     -- the boxes box_h, box_v1 and the separator:
61     procedure create_window;
62
63 end demo_main_window;
```

9.9.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                              MAIN WINDOW                                 --
6  --                                                                          --
7  --                                B o d y                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;               use ada.text_io;
40
41 with gtk.enums;                 use gtk.enums;
42
43
44 package body demo_main_window is
45
46
47     procedure create_window is begin
48         put_line ("create_window");
49
50
51         main_window := gtk_window_new (WINDOW_TOPLEVEL);
52         main_window.set_title ("Demo Canvas");
53         main_window.set_border_width (10);
54
55         -- CS: Set the minimum size of the main window ?
56         -- CS show main window size
57         -- main_window.set_size_request (1000, 500);
58
59         -- main_window.set_redraw_on_allocate (false);
60
61         -- Create the main box which contains everything:
62         gtk_new_hbox (box_h);
63
64
65         -- vertical box for coordinates display:
66         gtk_new_vbox (box_v1);
67         box_v1.set_border_width (10);
68
69         -- The box_v1 (on the left, containing the coordinates display)
70         -- must NOT change its width when the main window is resized:
71         box_h.pack_start (box_v1, expand => false);
72
73         -- Place a separator right of box_v1.
74         -- It must also have a FIXED width:
```

```
75          separator := gtk_separator_new (ORIENTATION_VERTICAL);
76          box_h.pack_start (separator, expand => false);
77
78          -- Later, in box_h will be added right of the separator:
79          -- - box_v0 (containing the command buttons) and
80          -- - swin (the scrolled window containing the canvas)
81          -- The scrolled window will
82          -- occupy the remaining space right of box_v0.
83
84          -- Add the main box to the main window:
85          main_window.add (box_h);
86
87       end create_window;
88
89
90    end demo_main_window;
```

9.10 Das Paket demo_scrolled_window

Alle wesentlichen Dinge betreffend des Rollfenster sind hier zu finden:

1. Deklaration des Rollfensters
2. Rollbalken und deren Einstellungen
3. Größe des Rollfensters
4. Betriebsart der Ansicht

9.10.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                               DEMO CANVAS                                --
4  --                                                                          --
5  --                            SCROLLED WINDOW                               --
6  --                                                                          --
7  --                                S p e c                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --    For correct displaying set tab width in your editor to 4.
27
28 --    The two letters "CS" indicate a "construction site" where things are not
29 --    finished yet or intended for the future.
30
31 --    Please send your questions and comments to:
32 --
33 --    info@blunk-electronic.de
34 --    or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --    history of changes:
37 --
38
39 with glib;                          use glib;
40 with gtk.widget;                    use gtk.widget;
41 with gtk.window;                    use gtk.window;
42 with gtk.scrolled_window;           use gtk.scrolled_window;
43 with gtk.adjustment;                use gtk.adjustment;
44 with gtk.scrollbar;                 use gtk.scrollbar;
45
46 with demo_logical_pixels;           use demo_logical_pixels;
47 with demo_geometry;                 use demo_geometry;
48 with demo_zoom;                     use demo_zoom;
49 with demo_window_dimensions;        use demo_window_dimensions;
50 with demo_conversions;              use demo_conversions;
51
52
53 package demo_scrolled_window is
54
55 -- SCROLLED WINDOW:
56
57     swin : gtk_scrolled_window;
```

```
58
59      -- Inside the scrolled window the canvas exists.
60
61
62  -- SCROLLBARS:
63
64      scrollbar_h_adj, scrollbar_v_adj : gtk_adjustment;
65      scrollbar_v, scrollbar_h : gtk_scrollbar;
66
67
68      -- This composite type contains the settings
69      -- of a scrollbar:
70      type type_scrollbar_settings is record
71          lower       : type_logical_pixels_positive;
72          upper       : type_logical_pixels_positive;
73          value       : type_logical_pixels_positive;
74          page_size   : type_logical_pixels_positive;
75      end record;
76
77      -- These are the places where the initial settings of
78      -- the scrollbars are stored:
79      scrollbar_v_init : type_scrollbar_settings;
80      scrollbar_h_init : type_scrollbar_settings;
81
82      -- These are the places where we backup the settings of
83      -- the scrollbars:
84      scrollbar_h_backup, scrollbar_v_backup : type_scrollbar_settings;
85
86
87
88      type type_scroll_direction is (
89          SCROLL_UP,
90          SCROLL_DOWN,
91          SCROLL_RIGHT,
92          SCROLL_LEFT);
93
94
95
96
97      -- This is the initial size of the scrolled window.
98      -- IMPORTANT: The height must be greater than the sum
99      -- of the height of all other widgets in the main window !
100     -- Otherwise the canvas may freeze and stop emitting signals.
101     swin_size_initial : constant type_window_size := (
102         width   => 400,
103         height  => 400);
104
105     -- The current size of the scrolled window. It gets updated
106     -- in procedure set_up_swin_and_scrollbars and
107     -- in cb_swin_size_allocate. This variable is required
108     -- in order to detect size changes of the scrolled window:
109     swin_size : type_window_size;
110
111
112
113
114     -- When the scrolled window is resized, then the canvas can
115     -- operate in in several ways. Currently these modes are defined:
116     type type_scrolled_window_zoom_mode is (
117         -- No zoom. No moving. Just more or less of
118         -- the canvas area is exposed:
119         MODE_1_EXPOSE_CANVAS,
120
121         -- Center of visible canvas area remains in the center.
122         -- Around the center more or less of the canvas area is exposed:
123         MODE_2_KEEP_CENTER,
124
125         -- The visible area remains fit into the scrolled window.
126         MODE_3_ZOOM_FIT);
127
128
129     -- Here the zoom mode of the scrolled window is fixed:
130     zoom_mode : constant type_scrolled_window_zoom_mode :=
131         -- MODE_1_EXPOSE_CANVAS;
132         -- MODE_2_KEEP_CENTER;
133         MODE_3_ZOOM_FIT;
```

```
134
135
136
137
138
139     -- Creates the scrolled window and its scrollbars:
140     procedure create_scrolled_window_and_scrollbars;
141
142
143
144     -- This procedure does a backup of the current settings
145     -- of both the horizontal and the vertical scrollbar:
146     procedure backup_scrollbar_settings;
147
148
149     -- This procedure restores the settings of the vertical
150     -- and horizontal scrollbar from the backup:
151     procedure restore_scrollbar_settings;
152
153
154
155     -- Sets the initial scrollbar settings based on
156     -- current base-offset and bounding-box:
157     procedure set_initial_scrollbar_settings;
158
159
160     -- For debugging, these procedures output the settings
161     -- of the scrollbars on the console:
162     procedure show_adjustments_v;
163     procedure show_adjustments_h;
164
165
166     -- This function calculates the zoom factor required to
167     -- fit the given area into the current scrolled window.
168     -- The scrolled window has an initial size on startup. Later, when
169     -- the operator resizes the main window, the scrolled window gets
170     -- larger or smaller. This results in a situation depended zoom factor:
171     function get_ratio (
172         area : in type_area)
173         return type_zoom_factor;
174
175
176     -- Updates the limits of the scrollbars.
177     -- The argument C1 provides the old corners of the
178     -- bounding-box on the canvas and C2 the new corners:
179     procedure update_scrollbar_limits (
180         C1, C2 : in type_bounding_box_corners);
181
182
183 end demo_scrolled_window;
```

9.10.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                            SCROLLED WINDOW                               --
6  --                                                                          --
7  --                                B o d y                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;               use ada.text_io;
40
41 -- with glib;                       use glib;
42 with gtk.widget;                with gtk.widget;
43 with gtk.window;                with gtk.window;
44 with gtk.enums;                 with gtk.enums;
45 with gtk.scrolled_window;       with gtk.scrolled_window;
46 with gtk.adjustment;            with gtk.adjustment;
47 with gtk.scrollbar;             with gtk.scrollbar;
48
49 with demo_window_dimensions;    with demo_window_dimensions;
50 with demo_base_offset;
51 with demo_bounding_box;
52
53
54
55 package body demo_scrolled_window is
56
57
58     procedure create_scrolled_window_and_scrollbars is
59     begin
60         put_line ("create_scrolled_window");
61
62         -- Create a scrolled window:
63         swin := gtk_scrolled_window_new (
64             hadjustment => null, vadjustment => null);
65
66         -- Set the minimum size of the scrolled window and
67         -- the global swin_size variable.
68         -- There are two ways to do that:
69         --
70         -- 1. Basing on the global bounding-box which has been calculated
71         --    by parsing the model database. This causes the scrolled window
72         --    to adapt on startup to the model.
73         --    IMPORTANT: The height must be greater than the sum
74         --    of the height of all other widgets in the main window !
```

```
75          --    Otherwise the canvas may freeze and stop emitting signals:
76          --
77          -- swin.set_size_request (
78          --  gint (bounding_box.width),
79          --  gint (bounding_box.height)); -- Mind a minimal height !
80          --  -- See above comment.
81          --
82          -- swin_size := (
83          --  width   => positive (bounding_box.width),
84          --  height  => positive (bounding_box.height));
85
86
87          -- 2. A static startup-configuration based on a certain
88          --    minimal width and height. This ensures that the scrolled
89          --    window has a predictable and well defined size.
90          --    This is to be prefered over approach 1 (see above):
91          swin.set_size_request (
92              gint (swin_size_initial.width),
93              gint (swin_size_initial.height));
94
95          swin_size := (
96              width   => swin_size_initial.width,
97              height  => swin_size_initial.height);
98
99
100
101         -- CS show window size
102
103         put_line ("scrolled window zoom mode: "
104             & type_scrolled_window_zoom_mode'image (zoom_mode));
105
106
107         -- swin.set_border_width (10);
108         -- swin.set_redraw_on_allocate (false);
109
110
111         scrollbar_h_adj := swin.get_hadjustment;
112         scrollbar_v_adj := swin.get_vadjustment;
113
114
115
116
117         -- behaviour:
118         swin.set_policy ( -- for scrollbars
119             hscrollbar_policy => gtk.enums.POLICY_AUTOMATIC,
120             -- hscrollbar_policy => gtk.enums.POLICY_NEVER,
121             vscrollbar_policy => gtk.enums.POLICY_AUTOMATIC);
122             -- vscrollbar_policy => gtk.enums.POLICY_NEVER);
123
124
125         -- CS: Attempt to disable auto-scrolling of scrollbars
126         -- when the canvas get the focus:
127         -- set_focus_hadjustment (
128         --  container   => swin,
129         --  adjustment  => scrollbar_h_adj);
130
131         -- scrollbar_h.set_can_focus (false);
132         -- swin.grab_focus;
133
134         -- swin.set_propagate_natural_height (true);
135
136
137     end create_scrolled_window_and_scrollbars;
138
139
140
141     procedure backup_scrollbar_settings is begin
142         --put_line ("backup_scrollbar_settings");
143         scrollbar_h_backup.lower := to_lp (scrollbar_h_adj.get_lower);
144         scrollbar_h_backup.value := to_lp (scrollbar_h_adj.get_value);
145         scrollbar_h_backup.page_size := to_lp (scrollbar_h_adj.get_page_size);
146         scrollbar_h_backup.upper := to_lp (scrollbar_h_adj.get_upper);
147
148         scrollbar_v_backup.lower := to_lp (scrollbar_v_adj.get_lower);
149         scrollbar_v_backup.value := to_lp (scrollbar_v_adj.get_value);
150         scrollbar_v_backup.page_size := to_lp (scrollbar_v_adj.get_page_size);
```

```
151         scrollbar_v_backup.upper := to_lp (scrollbar_v_adj.get_upper);
152     end backup_scrollbar_settings;
153
154
155     procedure restore_scrollbar_settings is begin
156         scrollbar_h_adj.set_lower (to_gdouble (scrollbar_h_backup.lower));
157         scrollbar_h_adj.set_value (to_gdouble (scrollbar_h_backup.value));
158
159         scrollbar_h_adj.set_page_size (
160             to_gdouble (scrollbar_h_backup.page_size));
161
162         scrollbar_h_adj.set_upper (to_gdouble (scrollbar_h_backup.upper));
163
164         scrollbar_v_adj.set_lower (to_gdouble (scrollbar_v_backup.lower));
165         scrollbar_v_adj.set_value (to_gdouble (scrollbar_v_backup.value));
166
167         scrollbar_v_adj.set_page_size (
168             to_gdouble (scrollbar_v_backup.page_size));
169
170         scrollbar_v_adj.set_upper (to_gdouble (scrollbar_v_backup.upper));
171     end restore_scrollbar_settings;
172
173
174
175     procedure set_initial_scrollbar_settings is
176         use demo_base_offset;
177         use demo_bounding_box;
178
179         debug : boolean := false;
180         -- debug : boolean := true;
181     begin
182         put_line ("set␣initial␣scrollbar␣settings");
183
184         scrollbar_v_init.upper := - F.y;
185
186         scrollbar_v_init.lower := scrollbar_v_init.upper -
187             type_logical_pixels (bounding_box.height);
188
189         scrollbar_v_init.page_size :=
190             type_logical_pixels (bounding_box.height);
191
192         scrollbar_v_init.value := scrollbar_v_init.lower;
193
194         if debug then
195             put_line ("␣vertical:");
196             put_line ("␣␣lower" &
197                 to_string (scrollbar_v_init.lower));
198
199             put_line ("␣␣upper" &
200                 to_string (scrollbar_v_init.upper));
201
202             put_line ("␣␣page␣" &
203                 to_string (scrollbar_v_init.page_size));
204
205             put_line ("␣␣value" &
206                 to_string (scrollbar_v_init.value));
207         end if;
208
209         scrollbar_h_init.lower := F.x;
210         scrollbar_h_init.upper := scrollbar_h_init.lower +
211             type_logical_pixels (bounding_box.width);
212
213         scrollbar_h_init.page_size :=
214             type_logical_pixels (bounding_box.width);
215
216         scrollbar_h_init.value := scrollbar_h_init.lower;
217
218         if debug then
219             put_line ("␣horizontal:");
220             put_line ("␣␣lower" &
221                 to_string (scrollbar_h_init.lower));
222
223             put_line ("␣␣upper" &
224                 to_string (scrollbar_h_init.upper));
225
226             put_line ("␣␣page␣" &
```

```
227                 to_string (scrollbar_h_init.page_size));
228
229             put_line ("␣␣value" &
230                 to_string (scrollbar_h_init.value));
231         end if;
232
233
234         ------------------------------------------------------------------------
235         -- CS: This code is experimental in order to make the canvas
236         -- dimensions adjust DYNAMICALLY to the scrollbar limits. So far this
237         -- was not successful because the canvas size can not be changed
238         -- for some unknown reason after initialization:
239
240 --      declare
241 --          w, h : gint;
242 --          a : gtk_allocation;
243 --      begin
244 --          w := gint (scrollbar_h_init.lower + scrollbar_h_init.upper);
245 --          h := gint (scrollbar_v_init.lower + scrollbar_v_init.upper);
246 --
247 --          canvas.get_allocation (a);
248 --          a.width := w;
249 --          a.height := h;
250 --          -- canvas.set_allocation (a);
251 --          -- canvas.size_allocate (a);
252 --          -- canvas.set_size_request (w, h);
253 --
254 --          if debug then
255 --              show_canvas_size;
256 --              -- put_line ("x/y : " & gint'image (a.x) & "/"
257 --                  & gint'image (a.y));
258 --          end if;
259 --      end;
260         ------------------------------------------------------------------------
261
262
263         -- put_line ("vertical:");
264         scrollbar_v_adj.set_upper (to_gdouble (scrollbar_v_init.upper));
265         scrollbar_v_adj.set_lower (to_gdouble (scrollbar_v_init.lower));
266
267         scrollbar_v_adj.set_page_size (
268             to_gdouble (scrollbar_v_init.page_size));
269
270         scrollbar_v_adj.set_value (to_gdouble (scrollbar_v_init.value));
271
272         -- put_line ("horizontal:");
273         scrollbar_h_adj.set_upper (to_gdouble (scrollbar_h_init.upper));
274         scrollbar_h_adj.set_lower (to_gdouble (scrollbar_h_init.lower));
275
276         scrollbar_h_adj.set_page_size (
277             to_gdouble (scrollbar_h_init.page_size));
278
279         scrollbar_h_adj.set_value (to_gdouble (scrollbar_h_init.value));
280
281         -- show_adjustments_h;
282         -- show_adjustments_v;
283
284         backup_scrollbar_settings;
285
286     end set_initial_scrollbar_settings;
287
288
289
290
291     procedure show_adjustments_v is
292         v_lower : type_logical_pixels :=
293             to_lp (scrollbar_v_adj.get_lower);
294
295         v_value : type_logical_pixels :=
296             to_lp (scrollbar_v_adj.get_value);
297
298         v_upper : type_logical_pixels :=
299             to_lp (scrollbar_v_adj.get_upper);
300
301         v_page  : type_logical_pixels :=
302             to_lp (scrollbar_v_adj.get_page_size);
```

```
303
304     begin
305         put_line ("vertical␣scrollbar␣adjustments:");
306         put_line ("␣lower" & to_string (v_lower));
307         put_line ("␣value" & to_string (v_value));
308         put_line ("␣page␣" & to_string (v_page));
309         put_line ("␣upper" & to_string (v_upper));
310     end show_adjustments_v;
311
312
313     procedure show_adjustments_h is
314         h_lower : type_logical_pixels := to_lp (scrollbar_h_adj.get_lower);
315         h_value : type_logical_pixels := to_lp (scrollbar_h_adj.get_value);
316         h_upper : type_logical_pixels := to_lp (scrollbar_h_adj.get_upper);
317
318         h_page  : type_logical_pixels :=
319             to_lp (scrollbar_h_adj.get_page_size);
320     begin
321         put_line ("horizontal␣scrollbar␣adjustments:");
322         put_line ("␣lower" & to_string (h_lower));
323         put_line ("␣value" & to_string (h_value));
324         put_line ("␣page␣" & to_string (h_page));
325         put_line ("␣upper" & to_string (h_upper));
326     end show_adjustments_h;
327
328
329
330     function get_ratio (
331         area : in type_area)
332         return type_zoom_factor
333     is
334         -- The allocation of the scrolled window provides
335         -- its width and height:
336         a : gtk_allocation;
337
338         -- The two zoom factors: one based on the width and another
339         -- based on the height of the given area:
340         sw, sh : type_zoom_factor;
341     begin
342         -- put_line ("get_ratio");
343
344         -- Get the current width and height of the scrolled window:
345         swin.get_allocation (a);
346
347         -- Get the ratio of width and height based on the current dimensions
348         -- of the scrolled window:
349         sw := type_zoom_factor
350             (type_distance_model (a.width) / area.width);
351
352         sh := type_zoom_factor
353             (type_distance_model (a.height) / area.height);
354
355         -- CS: Alternatively the ratio can be based on the initial dimensions
356         -- of the scrolled window. A boolean argument for this function
357         -- could be used to switch between current dimensions and initial
358         -- dimensions:
359         -- sw := type_zoom_factor
360         --  (type_distance_model (swin_size_initial.width) / area.width);
361         -- sh := type_zoom_factor
362         --  (type_distance_model (swin_size_initial.height) / area.height);
363
364         -- put_line ("sw: " & to_string (sw));
365         -- put_line ("sh: " & to_string (sh));
366
367         -- The smaller of sw and sh has the final say:
368         return type_zoom_factor'min (sw, sh);
369     end get_ratio;
370
371
372
373     procedure update_scrollbar_limits (
374         C1, C2 : in type_bounding_box_corners)
375     is
376         debug : boolean := false;
377         scratch : type_logical_pixels;
378
```

```
379         HL : type_logical_pixels := to_lp (scrollbar_h_adj.get_lower);
380         HU : type_logical_pixels := to_lp (scrollbar_h_adj.get_upper);
381
382         VL : type_logical_pixels := to_lp (scrollbar_v_adj.get_lower);
383         VU : type_logical_pixels := to_lp (scrollbar_v_adj.get_upper);
384
385         dHL, dHU : type_logical_pixels;
386         dVL, dVU : type_logical_pixels;
387     begin
388         if debug then
389             put_line ("VL␣␣␣␣␣" & to_string (VL));
390             put_line ("VU␣␣␣␣␣" & to_string (VU));
391
392             put_line ("C1.TL.y" & to_string (C1.TL.y));
393             put_line ("C1.BL.y" & to_string (C1.BL.y));
394
395             put_line ("C2.TL.y" & to_string (C2.TL.y));
396             put_line ("C2.BL.y" & to_string (C2.BL.y));
397         end if;
398
399         dHL := C2.BL.x - C1.BL.x;
400         dHU := C2.BR.x - C1.BR.x;
401
402         dVL := C2.TL.y - C1.TL.y;
403         dVU := C2.BL.y - C1.BL.y;
404
405         if debug then
406             put_line ("dVL␣␣␣␣" & to_string (dVL));
407             put_line ("dVU␣␣␣␣" & to_string (dVU));
408         end if;
409
410
411         -- horizontal:
412
413         -- The left end of the scrollbar is the same as the position
414         -- (value) of the scrollbar.
415         -- If the left edge of the bounding-box is farther to the
416         -- left than the left end of the bar, then the lower limit
417         -- moves to the left. It assumes the value of the left edge
418         -- of the bounding-box:
419         HL := HL + dHL;
420         if HL <= to_lp (scrollbar_h_adj.get_value) then
421             clip_min (HL, 0.0); -- suppress negative value
422             scrollbar_h_adj.set_lower (to_gdouble (HL));
423         else
424         -- If the left edge of the box is farther to the right than
425         -- the left end of the bar, then the lower limit can not be
426         -- moved further to the right. So the lower limit can at most assume
427         -- the value of the left end of the bar:
428             scrollbar_h_adj.set_lower (scrollbar_h_adj.get_value);
429         end if;
430
431         -- The right end of the scrollbar is the sum of its position (value)
432         -- and its length (page size):
433         scratch := to_lp (scrollbar_h_adj.get_value +
434                           scrollbar_h_adj.get_page_size);
435
436         HU := HU + dHU;
437         -- CS clip_max (HU, type_logical_pixels (scrolled_window_size.width));
438         -- If the right edge of the bounding-box is farther to the
439         -- right than the right end of the bar, then the upper limit
440         -- moves to the right. It assumes the value of the right edge
441         -- of the bounding-box:
442         if HU >= scratch then
443             scrollbar_h_adj.set_upper (to_gdouble (HU));
444         else
445         -- If the right edge of the box is farther to the left than
446         -- the right end of the bar, then the upper limit can not be
447         -- moved further to the left. So the upper limit can at most assume
448         -- the value of the right end of the bar:
449             scrollbar_h_adj.set_upper (to_gdouble (scratch));
450         end if;
451
452
453         -- vertical:
454
```

```
455         -- The upper end of the scrollbar is the same as the position
456         -- (value) of the scrollbar.
457         -- If the upper edge of the bounding-box is higher
458         -- than the upper end of the bar, then the lower limit
459         -- moves upwards. It assumes the value of the upper edge
460         -- of the bounding-box:
461         VL := VL + dVL;
462         if VL <= to_lp (scrollbar_v_adj.get_value) then
463             clip_min (VL, 0.0); -- suppress negative value
464             scrollbar_v_adj.set_lower (to_gdouble (VL));
465         else
466         -- If the upper edge of the box is below
467         -- the upper end of the bar, then the lower limit can not be
468         -- moved further upwards. So the lower limit can at most assume
469         -- the value of the upper end of the bar:
470             scrollbar_v_adj.set_lower (scrollbar_v_adj.get_value);
471         end if;
472
473         -- The lower end of the scrollbar is the sum of its position (value)
474         -- and its length (page size):
475         scratch := to_lp (scrollbar_v_adj.get_value +
476                           scrollbar_v_adj.get_page_size);
477
478         VU := VU + dVU;
479         -- CS clip_max (VU,
480         --   type_logical_pixels (scrolled_window_size.height));
481         -- If the lower edge of the bounding-box is below the
482         -- lower end of the bar, then the upper limit
483         -- moves further downwards. It assumes the value of the lower edge
484         -- of the bounding-box:
485         if VU >= scratch then
486             scrollbar_v_adj.set_upper (to_gdouble (VU));
487         else
488         -- If the lower edge of the box is above
489         -- the lower end of the bar, then the upper limit can not be
490         -- moved further downwards. So the upper limit can at most assume
491         -- the value of the lower end of the bar:
492             scrollbar_v_adj.set_upper (to_gdouble (scratch));
493         end if;
494
495         -- show_adjustments_v;
496     end update_scrollbar_limits;
497
498
499 end demo_scrolled_window;
```

9.11 Das Paket demo_coordinates_display

Hier geht es um die Erzeugung und Aktualisierung der Koordinatenanzeige.

9.11.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                           COORDINATES DISPLAY                            --
6  --                                                                          --
7  --                                S p e c                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with gtk.widget;                 use gtk.widget;
40 with gtk.box;                    use gtk.box;
41 with gtk.table;                  use gtk.table;
42 with gtk.label;                  use gtk.label;
43 with gtk.text_view;              use gtk.text_view;
44 with gtk.text_buffer;            use gtk.text_buffer;
45
46 package demo_coordinates_display is
47
48     table : gtk_table;
49
50     pointer_header                          : gtk_label;
51     pointer_x_label, pointer_y_label        : gtk_label;
52     pointer_x_value, pointer_y_value        : gtk_text_view;
53     pointer_x_buf, pointer_y_buf            : gtk_text_buffer;
54
55     cursor_header                           : gtk_label;
56     cursor_x_label, cursor_y_label          : gtk_label;
57     cursor_x_value, cursor_y_value          : gtk_text_view;
58     cursor_x_buf, cursor_y_buf              : gtk_text_buffer;
59
60     distances_header                        : gtk_label;
61     distances_dx_label, distances_dy_label  : gtk_label;
62     distances_absolute_label                : gtk_label;
63     distances_angle_label                   : gtk_label;
64     distances_dx_value, distances_dy_value  : gtk_text_view;
65     distances_absolute_value                : gtk_text_view;
66     distances_angle_value                   : gtk_text_view;
67     distances_dx_buf, distances_dy_buf      : gtk_text_buffer;
```

```
68      distances_absolute_buf                  : gtk_text_buffer;
69      distances_angle_buf                     : gtk_text_buffer;
70
71      grid_header                             : gtk_label;
72      grid_x_label, grid_y_label              : gtk_label;
73      grid_x_value, grid_y_value              : gtk_text_view;
74      grid_x_buf, grid_y_buf                  : gtk_text_buffer;
75
76      zoom_label                              : gtk_label;
77      zoom_value                              : gtk_text_view;
78      zoom_buf                                : gtk_text_buffer;
79
80      scale_label                             : gtk_label;
81      scale_value                             : gtk_text_view;
82      scale_buf                               : gtk_text_buffer;
83
84
85      -- Creates the display for pointer/mouse and cursor position,
86      -- distances and angle:
87      procedure set_up_coordinates_display;
88
89
90      -- Updates the cursor coordinates display
91      -- by the current cursor position:
92      procedure update_cursor_coordinates;
93
94
95      -- Updates the distances display:
96      procedure update_distances_display;
97
98
99      -- Updates the zoom display:
100     procedure update_zoom_display;
101
102
103     -- Updates the grid display:
104     procedure update_grid_display;
105
106
107     -- Updates the scale display:
108     procedure update_scale_display;
109
110 end demo_coordinates_display;
```

9.11.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                           COORDINATES DISPLAY                            --
6  --                                                                          --
7  --                               B o d y                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;                 use ada.text_io;
40
41 with glib;
42 with gtk.box;                     use gtk.box;
43 with gtk.enums;                   use gtk.enums;
44
45 with demo_logical_pixels;         use demo_logical_pixels;
46 with demo_geometry;               use demo_geometry;
47 with demo_zoom;                   use demo_zoom;
48 with demo_main_window;            use demo_main_window;
49 with demo_cursor;
50 with demo_canvas;
51 with demo_grid;
52 with demo_conversions;            use demo_conversions;
53 with demo_scale;
54
55
56 package body demo_coordinates_display is
57
58
59     procedure set_up_coordinates_display is
60         use glib;
61
62         -- The width of the text view shall be wide enough
63         -- to fit the greatest numbers:
64         pos_field_width_min : constant gint := 80;
65     begin
66         -- CS To disable focus use
67         -- procedure Set_Focus_On_Click
68         --    (Widget         : not null access Gtk_Widget_Record;
69         --     Focus_On_Click : Boolean);
70
71         -- Create a table, that contains headers, text labels
72         -- and text views for the actual coordinates:
73         gtk_new (table, rows => 11, columns => 2,
74             homogeneous => false);
```

```
75          -- table.set_col_spacings (50);
76          -- table.set_border_width (10);
77
78          -- The table shall not expand downward:
79          box_v1.pack_start (table, expand => false);
80
81
82          -- POINTER / MOUSE:
83          gtk_new (pointer_header, "POINTER");
84          gtk_new (pointer_x_label, "x:"); -- create a text label
85
86          -- The label shall be aligned in the column.
87          -- The discussion at:
88          -- <https://stackoverflow.com/questions/26345989/
89          -- gtk-how-to-align-a-label-to-the-left-in-a-table>
90          -- gave the solution. See also package gtk.misc for details:
91          pointer_x_label.set_alignment (0.0, 0.0);
92          gtk_new (pointer_x_value); -- create a text view vor the value
93          -- A minimum width must be set for the text.
94          -- Setting the size request is one way. The height is
95          -- not affected, therefore the value -1:
96          pointer_x_value.set_size_request (pos_field_width_min, -1);
97          -- See also discussion at:
98          -- <https://stackoverflow.com/questions/24412859/
99          -- gtk-how-can-the-size-of-a-textview-be-set-manually>
100         -- for a way to achieve this using a tag.
101
102         gtk_new (pointer_x_buf); -- create a text buffer
103
104         -- align the value left
105         pointer_x_value.set_justification (JUSTIFY_RIGHT);
106         pointer_x_value.set_editable (false); -- the value is not editable
107         pointer_x_value.set_cursor_visible (false); -- do not show a cursor
108
109         gtk_new (pointer_y_label, "y:"); -- create a text label
110         pointer_y_label.set_alignment (0.0, 0.0);
111         gtk_new (pointer_y_value);
112         pointer_y_value.set_size_request (pos_field_width_min, -1);
113         gtk_new (pointer_y_buf); -- create a text buffer
114
115         -- align the value left
116         pointer_y_value.set_justification (JUSTIFY_RIGHT);
117         pointer_y_value.set_editable (false); -- the value is not editable
118         pointer_y_value.set_cursor_visible (false); -- do not show a cursor
119
120         ----------------------------------------------------------------------
121
122         -- CURSOR
123         gtk_new (cursor_header, "CURSOR");
124
125         gtk_new (cursor_x_label, "x:");
126         cursor_x_label.set_alignment (0.0, 0.0);
127         gtk_new (cursor_x_value);
128         cursor_x_value.set_size_request (pos_field_width_min, -1);
129
130         gtk_new (cursor_x_buf);
131         cursor_x_value.set_justification (JUSTIFY_RIGHT);
132         cursor_x_value.set_editable (false);
133         cursor_x_value.set_cursor_visible (false);
134
135         gtk_new (cursor_y_label, "y:");
136         cursor_y_label.set_alignment (0.0, 0.0);
137         gtk_new (cursor_y_value);
138         cursor_y_value.set_size_request (pos_field_width_min, -1);
139         gtk_new (cursor_y_buf);
140         cursor_y_value.set_justification (JUSTIFY_RIGHT);
141         cursor_y_value.set_editable (false);
142         cursor_y_value.set_cursor_visible (false);
143
144         ----------------------------------------------------------------------
145         -- DISTANCES
146         gtk_new (distances_header, "DISTANCE");
147         gtk_new (distances_dx_label, "dx:");
148         distances_dx_label.set_alignment (0.0, 0.0);
149         gtk_new (distances_dx_value);
150         distances_dx_value.set_size_request (pos_field_width_min, -1);
```

```
151
152         gtk_new (distances_dx_buf);
153         distances_dx_value.set_justification (JUSTIFY_RIGHT);
154         distances_dx_value.set_editable (false);
155         distances_dx_value.set_cursor_visible (false);
156
157
158         gtk_new (distances_dy_label, "dy:");
159         distances_dy_label.set_alignment (0.0, 0.0);
160         gtk_new (distances_dy_value);
161         distances_dy_value.set_size_request (pos_field_width_min, -1);
162
163         gtk_new (distances_dy_buf);
164         distances_dy_value.set_justification (JUSTIFY_RIGHT);
165         distances_dy_value.set_editable (false);
166         distances_dy_value.set_cursor_visible (false);
167
168
169         gtk_new (distances_absolute_label, "abs:");
170         distances_absolute_label.set_alignment (0.0, 0.0);
171         gtk_new (distances_absolute_value);
172         distances_absolute_value.set_size_request (pos_field_width_min, -1);
173
174         gtk_new (distances_absolute_buf);
175         distances_absolute_value.set_justification (JUSTIFY_RIGHT);
176         distances_absolute_value.set_editable (false);
177         distances_absolute_value.set_cursor_visible (false);
178
179
180         gtk_new (distances_angle_label, "angle:");
181         distances_angle_label.set_alignment (0.0, 0.0);
182         gtk_new (distances_angle_value);
183         distances_angle_value.set_size_request (pos_field_width_min, -1);
184
185         gtk_new (distances_angle_buf);
186         distances_angle_value.set_justification (JUSTIFY_RIGHT);
187         distances_angle_value.set_editable (false);
188         distances_angle_value.set_cursor_visible (false);
189
190         ------------------------------------------------------------------------
191         -- GRID
192         gtk_new (grid_header, "GRID");
193         gtk_new (grid_x_label, "x:");
194         grid_x_label.set_alignment (0.0, 0.0);
195         gtk_new (grid_x_value);
196         grid_x_value.set_size_request (pos_field_width_min, -1);
197
198         gtk_new (grid_x_buf);
199         grid_x_value.set_justification (JUSTIFY_RIGHT);
200         grid_x_value.set_editable (false);
201         grid_x_value.set_cursor_visible (false);
202
203
204         gtk_new (grid_y_label, "y:");
205         grid_y_label.set_alignment (0.0, 0.0);
206         gtk_new (grid_y_value);
207         grid_x_value.set_size_request (pos_field_width_min, -1);
208
209         gtk_new (grid_y_buf);
210         grid_y_value.set_justification (JUSTIFY_RIGHT);
211         grid_y_value.set_editable (false);
212         grid_y_value.set_cursor_visible (false);
213
214         ------------------------------------------------------------------------
215         -- ZOOM FACTOR
216         -- gtk_new (zoom_header, "ZOOM");
217         gtk_new (zoom_label, "zoom:");
218         zoom_label.set_alignment (0.0, 0.0);
219         gtk_new (zoom_value);
220         zoom_value.set_size_request (pos_field_width_min, -1);
221
222         gtk_new (zoom_buf);
223         zoom_value.set_justification (JUSTIFY_RIGHT);
224         zoom_value.set_editable (false);
225         zoom_value.set_cursor_visible (false);
226
```

```
227         ------------------------------------------------------------------------
228         -- SCALE
229         gtk_new (scale_label, "scale:");
230         scale_label.set_alignment (0.0, 0.0);
231         gtk_new (scale_value);
232         scale_value.set_size_request (pos_field_width_min, -1);
233
234         gtk_new (scale_buf);
235         scale_value.set_justification (JUSTIFY_RIGHT);
236         scale_value.set_editable (false);
237         scale_value.set_cursor_visible (false);
238
239         ------------------------------------------------------------------------
240         -- Put the items in the table:
241
242         -- MOUSE / POINTER:
243         table.attach (pointer_header,
244             left_attach => 0, right_attach  => 2,
245             top_attach  => 0, bottom_attach => 1);
246
247         -- x-coordinate:
248         table.attach (pointer_x_label,
249             left_attach => 0, right_attach  => 1,
250             top_attach  => 1, bottom_attach => 2);
251
252         table.attach (pointer_x_value,
253             left_attach => 1, right_attach  => 2,
254             top_attach  => 1, bottom_attach => 2);
255
256         -- y-coordinate:
257         table.attach (pointer_y_label,
258             left_attach => 0, right_attach  => 1,
259             top_attach  => 2, bottom_attach => 3);
260
261         table.attach (pointer_y_value,
262             left_attach => 1, right_attach  => 2,
263             top_attach  => 2, bottom_attach => 3);
264
265
266         -- CURSOR:
267         table.attach (cursor_header,
268             left_attach => 0, right_attach  => 2,
269             top_attach  => 3, bottom_attach => 4);
270
271         -- x-coordinate:
272         table.attach (cursor_x_label,
273             left_attach => 0, right_attach  => 1,
274             top_attach  => 4, bottom_attach => 5);
275
276         table.attach (cursor_x_value,
277             left_attach => 1, right_attach  => 2,
278             top_attach  => 4, bottom_attach => 5);
279
280         -- y-coordinate:
281         table.attach (cursor_y_label,
282             left_attach => 0, right_attach  => 1,
283             top_attach  => 5, bottom_attach => 6);
284
285         table.attach (cursor_y_value,
286             left_attach => 1, right_attach  => 2,
287             top_attach  => 5, bottom_attach => 6);
288
289
290
291         -- DISTANCES:
292         table.attach (distances_header,
293             left_attach => 0, right_attach  => 2,
294             top_attach  => 6, bottom_attach => 7);
295
296         -- x-coordinate:
297         table.attach (distances_dx_label,
298             left_attach => 0, right_attach  => 1,
299             top_attach  => 7, bottom_attach => 8);
300
301         table.attach (distances_dx_value,
302             left_attach => 1, right_attach  => 2,
```

```
303             top_attach  => 7, bottom_attach => 8);
304
305         -- y-coordinate:
306         table.attach (distances_dy_label,
307             left_attach => 0, right_attach  => 1,
308             top_attach  => 9, bottom_attach => 10);
309
310         table.attach (distances_dy_value,
311             left_attach => 1, right_attach  => 2,
312             top_attach  => 9, bottom_attach => 10);
313
314         -- absolute:
315         table.attach (distances_absolute_label,
316             left_attach => 0, right_attach  => 1,
317             top_attach  => 10, bottom_attach => 11);
318
319         table.attach (distances_absolute_value,
320             left_attach => 1, right_attach  => 2,
321             top_attach  => 10, bottom_attach => 11);
322
323         -- angle:
324         table.attach (distances_angle_label,
325             left_attach => 0, right_attach  => 1,
326             top_attach  => 11, bottom_attach => 12);
327
328         table.attach (distances_angle_value,
329             left_attach => 1, right_attach  => 2,
330             top_attach  => 11, bottom_attach => 12);
331
332
333
334         -- GRID:
335         table.attach (grid_header,
336             left_attach => 0, right_attach  => 2,
337             top_attach  => 12, bottom_attach => 13);
338
339         -- x-axis:
340         table.attach (grid_x_label,
341             left_attach => 0, right_attach  => 1,
342             top_attach  => 13, bottom_attach => 14);
343
344         table.attach (grid_x_value,
345             left_attach => 1, right_attach  => 2,
346             top_attach  => 13, bottom_attach => 14);
347
348         -- y-axis:
349         table.attach (grid_y_label,
350             left_attach => 0, right_attach  => 1,
351             top_attach  => 14, bottom_attach => 15);
352
353         table.attach (grid_y_value,
354             left_attach => 1, right_attach  => 2,
355             top_attach  => 14, bottom_attach => 15);
356
357
358         -- ZOOM:
359         table.attach (zoom_label,
360             left_attach => 0, right_attach  => 1,
361             top_attach  => 15, bottom_attach => 16);
362
363         table.attach (zoom_value,
364             left_attach => 1, right_attach  => 2,
365             top_attach  => 15, bottom_attach => 16);
366
367
368         -- SCALE:
369         table.attach (scale_label,
370             left_attach => 0, right_attach  => 1,
371             top_attach  => 16, bottom_attach => 17);
372
373         table.attach (scale_value,
374             left_attach => 1, right_attach  => 2,
375             top_attach  => 16, bottom_attach => 17);
376
377     end set_up_coordinates_display;
378
```

```
379
380     procedure update_cursor_coordinates is
381         use demo_scale;
382         use demo_cursor;
383     begin
384         -- x-axis:
385         cursor_x_buf.set_text (to_string (to_reality (cursor.position.x)));
386         cursor_x_value.set_buffer (cursor_x_buf);
387
388         -- y-axis:
389         cursor_y_buf.set_text (to_string (to_reality (cursor.position.y)));
390         cursor_y_value.set_buffer (cursor_y_buf);
391     end update_cursor_coordinates;
392
393
394
395     procedure update_distances_display is
396         use glib;
397         use demo_cursor;
398         use demo_canvas;
399         use demo_scale;
400
401         px, py : gint; -- the pointer position
402         cp : type_logical_pixels_vector;
403         mp : type_vector_model;
404
405         dx, dy : type_distance_model;
406         dabs : type_distance_model;
407         angle : type_rotation_model;
408     begin
409         -- Get the current pointer/mouse position:
410         canvas.get_pointer (px, py);
411         cp := (type_logical_pixels (px), type_logical_pixels (py));
412
413         -- Convert the pointer position to a real
414         -- point in the model:
415         mp := canvas_to_real (cp, S);
416
417         -- Compute the relative distance from cursor
418         -- to pointer:
419         dx := mp.x - cursor.position.x;
420         dy := mp.y - cursor.position.y;
421
422         -- Compute the absolute distance from
423         -- cursor to pointer:
424         dabs := get_distance (
425             p1 => (0.0, 0.0),
426             p2 => (dx, dy));
427
428         -- Compute the angle of direction from cursor
429         -- to pointer:
430         angle := get_angle (
431             p1 => (0.0, 0.0),
432             p2 => (dx, dy));
433
434
435         -- Output the relative distances on the display:
436
437         -- dx:
438         distances_dx_buf.set_text (to_string (to_reality (dx)));
439         distances_dx_value.set_buffer (distances_dx_buf);
440
441         -- dy:
442         distances_dy_buf.set_text (to_string (to_reality (dy)));
443         distances_dy_value.set_buffer (distances_dy_buf);
444
445         -- absolute:
446         distances_absolute_buf.set_text (to_string (to_reality (dabs)));
447         distances_absolute_value.set_buffer (distances_absolute_buf);
448
449         -- angle:
450         distances_angle_buf.set_text (to_string (angle));
451         distances_angle_value.set_buffer (distances_angle_buf);
452     end update_distances_display;
453
454
```

```
455
456     procedure update_zoom_display is begin
457         zoom_buf.set_text (to_string (S));
458         zoom_value.set_buffer (zoom_buf);
459     end update_zoom_display;
460
461
462
463     procedure update_grid_display is
464         use demo_grid;
465         use demo_scale;
466     begin
467         -- x-axis:
468         grid_x_buf.set_text (to_string (to_reality (grid.spacing.x)));
469         grid_x_value.set_buffer (grid_x_buf);
470
471         -- y-axis:
472         grid_y_buf.set_text (to_string (to_reality (grid.spacing.y)));
473         grid_y_value.set_buffer (grid_y_buf);
474     end update_grid_display;
475
476
477     procedure update_scale_display is
478         use demo_grid;
479         use demo_scale;
480     begin
481         scale_buf.set_text (to_string (M));
482         scale_value.set_buffer (scale_buf);
483     end update_scale_display;
484
485
486 end demo_coordinates_display;
```

9.12 Das Paket demo_zoom

Alles, was sich um den Vergrößerungsfaktor S dreht, ist in diesem Paket zu finden:

1. Typdeklaration und Wertebereich
2. Berechnung des Translation-Offsets T
3. Vergrößerung eines bestimmten Bereiches
4. Einpassen aller Objekte (zoom to fit)

9.12.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                                 ZOOM                                     --
6  --                                                                          --
7  --                                S p e c                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with demo_logical_pixels;       use demo_logical_pixels;
40 with demo_geometry;             use demo_geometry;
41
42
43 package demo_zoom is
44
45     -- This is the specification of the zoom factor
46     -- (German: Vergroesserungsfaktor)
47     type type_zoom_factor is digits 3 range 0.10 .. 100.0;
48
49
50     -- This is the global zoom factor:
51     S : type_zoom_factor := 1.0;
52
53     -- This is the multiplier that is used when the
54     -- global zoom factor is increased or decreased:
55     SM : constant type_zoom_factor := 1.2;
56
57
```

```
58
59      -- Converts the given zoom factor to a string.
60      -- CS: Since type_zoom_factor is a float type, the output is
61      -- something like 1.44E+00. Instead the output should be something
62      -- simpler like 1.44:
63      function to_string (
64          zf : in type_zoom_factor)
65          return string;
66
67
68      -- This procedure increases the global zoom factor
69      -- by multiplying it by SM:
70      procedure increase_zoom_factor;
71
72
73      -- This procedure decreases the global zoom factor
74      -- by dividing it by SM:
75      procedure decrease_zoom_factor;
76
77
78
79      -- There are two kinds of zoom-operations:
80      type type_zoom_direction is (ZOOM_IN, ZOOM_OUT);
81
82
83      -- This procedure sets the global translate-offset T that is
84      -- required for a zoom-operation.
85      -- After changing the zoom factor S (either by zoom on mouse pointer or
86      -- by zoom on cursor), the translate-offset T must
87      -- be calculated anew. The computation requires as input values
88      -- the zoom center as virtual model point (CS1) or as canvas point (CS2).
89      -- So there is a procedure set_translation_for_zoom that takes a canvas
90      -- point and another that takes a real model point.
91      -- Later, when the actual drawing takes place (see function
92      -- cb_draw_objects) the drawing will be dragged back by the
93      -- translate-offset so that the operator gets the impression of a
94      -- zoom-in or zoom-out effect.
95      -- Without applying a translate-offset the drawing would be appearing as
96      -- expanding to the upper-right (on zoom-in) or shrinking toward the
97      -- lower-left:
98      procedure set_translation_for_zoom (
99          -- The zoom factor before zoom:
100         S1  : in type_zoom_factor;
101
102         -- The zoom factor after zoom:
103         S2  : in type_zoom_factor;
104
105         -- The zoom center as canvas point:
106         Z1  : in type_logical_pixels_vector);
107
108
109
110     procedure set_translation_for_zoom (
111         -- The zoom factor before zoom:
112         S1  : in type_zoom_factor;
113
114         -- The zoom factor after zoom:
115         S2  : in type_zoom_factor;
116
117         -- The zoom center as a real model point:
118         M   : in type_vector_model);
119
120
121
122
123
124     -- This composite type provides the ingredients
125     -- required to do a zoom-to-area operation:
126     type type_zoom_area is record
127         -- This flag indicates that the operation is active.
128         -- As soon as the operator clicks the "Zoom Area" button,
129         -- this flag is set. It is cleared when the operator
130         -- releases the right mouse button after she/he has
131         -- defined the zoom-area. The zoom-area is a
132         -- rectangle:
133         active  : boolean := false;
```

```
134
135         -- This is the first corner of the area. It is assigned
136         -- when the operator presses the right mouse button
137         -- on the canvas to define the start point of the zoom-area:
138         k1      : type_vector_model;
139
140         -- This is the second corner of the area. It is assigned
141         -- when the operator releases the right mouse button
142         -- on the canvas to define end point of the the zoom-area:
143         k2      : type_vector_model;
144
145         -- This is the actual area to be zoomed to. It gets fully
146         -- specified when the operator releases the right mouse button.
147         -- The area will then be passed to the function zoom_to_fit
148         -- in order to have the area displayed on the canvas:
149         area    : type_area;
150
151         ------------------------------------------------------------
152         -- In order to display a rectangle that indicates the
153         -- currently selected area we need this stuff.
154         -- This is all in the canvas domain and has nothing to
155         -- do with the area in the model domain (see above):
156
157         -- This flag indicates that the operator has started
158         -- the selection. It is cleared when the operator is done
159         -- with the selection by releasing the right mouse button:
160         started : boolean := false;
161
162         -- The corners of the selected area:
163         l1      : type_logical_pixels_vector; -- the start point
164         l2      : type_logical_pixels_vector; -- the end point
165     end record;
166
167
168
169     -- This is the instance of the zoom-area:
170     zoom_area : type_zoom_area;
171
172
173     -- This is the linewidth of the rectangle that
174     -- marks the selected zoom area:
175     zoom_area_linewidth : constant type_logical_pixels_positive := 2.0;
176
177
178     -- This procedure resets the zoom_area (see above)
179     -- to its default values.
180     -- Use this procedure to abort a zoom-to-area operation.
181     procedure reset_zoom_area;
182
183
184
185     -- Zooms in or out at the current cursor position.
186     -- If the direction is ZOOM_IN, then the global zoom factor S
187     -- is increased by multplying it with the zoom multiplier SM.
188     -- If direction is ZOOM_OUT then it decreases by dividing
189     -- by SM:
190     procedure zoom_on_cursor (
191         direction : in type_zoom_direction);
192
193
194     -- This procedure sets the global zoom factor S and translate-offset T
195     -- so that all objects of the given area fit into the scrolled window.
196     -- The zoom center is the top-left corner of the given area.
197     procedure zoom_to_fit (
198         area : in type_area);
199
200
201     -- This procedure sets the global zoom factor S and translate-offset T
202     -- so that all objects of bounding-box fit into the scrolled window.
203     -- The zoom center is the top-left corner of bounding-box.
204     procedure zoom_to_fit_all;
205
206
207     -- If a zoom-to-area operation has started, then
208     -- this procedure draws the rectangle around the
209     -- area to be zoomed at.
```

```
210      -- The rectangle is drawn directly on the cairo_context.
211      procedure draw_zoom_area;
212
213 end demo_zoom;
```

9.12.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                                 ZOOM                                     --
6  --                                                                          --
7  --                                B o d y                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;                use ada.text_io;
40
41 with cairo;
42
43 with demo_base_offset;           use demo_base_offset;
44 with demo_translate_offset;      use demo_translate_offset;
45 with demo_conversions;           use demo_conversions;
46 with demo_visible_area;
47 with demo_coordinates_display;
48 with demo_cursor;
49 with demo_canvas;
50 with demo_scrolled_window;
51 with demo_bounding_box;
52
53
54
55
56 package body demo_zoom is
57
58
59     function to_string (
60         zf : in type_zoom_factor)
61         return string
62     is begin
63         return type_zoom_factor'image (zf);
64     end to_string;
65
66
67     procedure increase_zoom_factor is begin
68         S := S * SM;
69
70         exception
71             when constraint_error =>
72                 put_line ("upper␣zoom␣limit␣reached");
73             when others => null;
74     end increase_zoom_factor;
```

```
75
76
77      procedure decrease_zoom_factor is begin
78          S := S / SM;
79
80          exception
81              when constraint_error =>
82                  put_line ("lower␣zoom␣limit␣reached");
83              when others => null;
84      end decrease_zoom_factor;
85
86
87
88
89      procedure set_translation_for_zoom (
90          S1  : in type_zoom_factor;
91          S2  : in type_zoom_factor;
92          Z1  : in type_logical_pixels_vector) -- a canvas point
93      is
94          debug : boolean := false;
95
96          -- Convert the given canvas point to a
97          -- virtual model point according to the old zoom factor:
98          V : constant type_vector_model := canvas_to_virtual (Z1, S1);
99
100         Z2 : type_logical_pixels_vector;
101     begin
102         if debug then
103             put_line ("set_translation_for_zoom");
104         end if;
105
106         -- Starting at the virtual model point V,
107         -- compute the prospected canvas point according to the
108         -- new zoom factor.
109         -- The current translate-offset will NOT be taken into account:
110         Z2 := virtual_to_canvas (V, S2, false);
111
112         -- This is the offset from Z1 to the prospected
113         -- point Z2. The offset must be multiplied by -1 because the
114         -- drawing must be dragged-back to the given pointer position:
115         T.x := -(Z2.x - Z1.x);
116         T.y := -(Z2.y - Z1.y);
117         -- CS simplify or use vector mathematics specified in
118         -- package demo_logical_pixels
119
120         if debug then
121             put_line ("␣T:␣" & to_string (T));
122         end if;
123     end set_translation_for_zoom;
124
125
126
127
128     procedure set_translation_for_zoom (
129         S1  : in type_zoom_factor;
130         S2  : in type_zoom_factor;
131         M   : in type_vector_model) -- real model point
132     is
133         debug : boolean := false;
134
135         -- Convert the given real model point to
136         -- a virtual model point:
137         V : constant type_vector_model := to_virtual (M);
138
139         Z1, Z2 : type_logical_pixels_vector;
140     begin
141         if debug then
142             put_line ("set_translation_for_zoom");
143         end if;
144
145         -- Convert the virtual model point to a canvas point
146         -- according to the old zoom factor.
147         -- The current translate-offset will NOT be taken into account:
148         Z1 := virtual_to_canvas (V, S1, translate => true);
149
150         -- Compute the prospected canvas point according to the
```

```
151         -- new zoom factor.
152         -- The current translate-offset will NOT be taken into account:
153         Z2 := virtual_to_canvas (V, S2, translate => false);
154         -- put_line ("Z2 " & to_string (Z2));
155
156         -- This is the offset from point Z1 to the prospected
157         -- point Z2. The offset must be multiplied by -1 because the
158         -- drawing must be dragged-back to the given pointer position:
159         T.x := -(Z2.x - Z1.x);
160         T.y := -(Z2.y - Z1.y);
161         -- CS simplify or use vector mathematics specified in
162         -- package demo_logical_pixels
163
164         if debug then
165             put_line ("␣T:␣" & to_string (T));
166         end if;
167     end set_translation_for_zoom;
168
169
170
171     procedure reset_zoom_area is begin
172         put_line ("reset_zoom_area");
173         zoom_area := (others => <>);
174     end reset_zoom_area;
175
176
177
178     procedure zoom_on_cursor (
179         direction : in type_zoom_direction)
180     is
181         use demo_visible_area;
182         use demo_coordinates_display;
183         use demo_cursor;
184         use demo_canvas;
185         use demo_scrolled_window;
186
187         S1 : constant type_zoom_factor := S;
188
189         -- The corners of the bounding-box on the canvas before
190         -- and after zooming:
191         C1, C2 : type_bounding_box_corners;
192     begin
193         put_line ("zoom_on_cursor␣" & type_zoom_direction'image (direction));
194
195         C1 := get_bounding_box_corners;
196
197         case direction is
198             when ZOOM_IN =>
199                 increase_zoom_factor;
200                 put_line ("␣zoom␣in");
201
202             when ZOOM_OUT =>
203                 decrease_zoom_factor;
204                 put_line ("␣zoom␣out");
205
206             when others => null;
207         end case;
208
209         update_zoom_display;
210
211         -- put_line (" S" & to_string (S));
212
213         -- After changing the zoom factor, the translate-offset must
214         -- be calculated anew. When the actual drawing takes
215         -- place (see function cb_draw_objects)
216         -- then the drawing will be dragged back by the translate-offset
217         -- so that the operator gets the impression of a zoom-into or
218         -- zoom-out effect.
219         -- Without applying a translate-offset the drawing would be appearing
220         -- as expanding to the upper-right (on zoom-in) or shrinking toward
221         -- the lower-left:
222         set_translation_for_zoom (S1, S, cursor.position);
223
224         C2 := get_bounding_box_corners;
225         update_scrollbar_limits (C1, C2);
226
```

```
227         -- show_adjustments_v;
228
229         backup_visible_area (get_visible_area (canvas));
230
231         -- schedule a redraw:
232         refresh;
233     end zoom_on_cursor;
234
235
236
237
238     procedure zoom_to_fit (
239         area : in type_area)
240     is
241         use demo_visible_area;
242         use demo_scrolled_window;
243         use demo_coordinates_display;
244
245         debug : boolean := false;
246     begin
247         put_line ("zoom_to_fit");
248
249         -- Calculate the zoom factor that is required to
250         -- fit the given area into the scrolled window:
251         S := get_ratio (area);
252
253         if debug then
254             put_line ("␣S:␣" & type_zoom_factor'image (S));
255         end if;
256
257         update_zoom_display;
258         ------------------------------------------------------
259
260         -- Calculate the translate-offset that is required to
261         -- center the given area on the visible area:
262         center_to_visible_area (area);
263
264         if debug then
265             show_adjustments_h;
266             show_adjustments_v;
267         end if;
268
269         --backup_scrollbar_settings;
270     end zoom_to_fit;
271
272
273
274     procedure zoom_to_fit_all is
275         use demo_bounding_box;
276         use demo_visible_area;
277         use demo_scrolled_window;
278         use demo_coordinates_display;
279         use demo_canvas;
280
281         -- debug : boolean := true;
282         debug : boolean := false;
283     begin
284         -- put_line ("zoom_to_fit");
285
286         -- Reset the translate-offset:
287         T := (0.0, 0.0);
288
289         -- Compute the new bounding-box. Update global
290         -- variable bounding_box:
291         compute_bounding_box;
292
293         -- In order to simulate a violation of the maximal
294         -- size of the bounding-box try this:
295         -- compute_bounding_box (ignore_errors => true);
296         -- compute_bounding_box (test_only => true, ignore_errors => true);
297
298         -- Compute the new base-offset. Update global variable F:
299         set_base_offset;
300
301         -- Since the bounding_box has changed, the scrollbars
302         -- must be reinitialized:
```

```
303         set_initial_scrollbar_settings;
304
305         -- Calculate the zoom factor that is required to
306         -- fit all objects into the scrolled window:
307         S := get_ratio (bounding_box);
308
309
310
311         if debug then
312             put_line ("␣S:␣" & type_zoom_factor'image (S));
313         end if;
314
315         update_zoom_display;
316
317
318         -- Calculate the translate-offset that is required to
319         -- "move" all objects to the center of the visible area:
320         center_to_visible_area (bounding_box);
321
322         backup_visible_area (bounding_box);
323
324         -- Schedule a redraw of the canvas:
325         refresh;
326     end zoom_to_fit_all;
327
328
329
330     procedure draw_zoom_area is
331         use cairo;
332         use demo_canvas;
333
334         x, y : type_logical_pixels;
335         w, h : type_logical_pixels;
336
337         l1 : type_logical_pixels_vector renames zoom_area.l1;
338         l2 : type_logical_pixels_vector renames zoom_area.l2;
339     begin
340         if zoom_area.started then
341
342             -- Set the color of the rectangle:
343             set_source_rgb (context, 0.5, 0.5, 0.5); -- gray
344
345             -- Compute the position and dimensions of
346             -- the rectangle:
347
348             -- x-position:
349             if l1.x < l2.x then
350                 x := l1.x;
351             else
352                 x := l2.x;
353             end if;
354
355             -- y-position:
356             if l1.y < l2.y then
357                 y := l1.y;
358             else
359                 y := l2.y;
360             end if;
361
362             -- width and height:
363             w := abs (l1.x - l2.x);
364             h := abs (l1.y - l2.y);
365
366             set_line_width (context, to_gdouble (zoom_area_linewidth));
367
368             rectangle (context,
369                 to_gdouble (x),
370                 to_gdouble (y),
371                 to_gdouble (w),
372                 to_gdouble (h));
373
374             stroke;
375         end if;
376     end draw_zoom_area;
377
378 end demo_zoom;
```

9.13 Das Paket demo_visible_area

Dieses Paket enthält Routinen rund um den sichtbaren Bereich des Modells sowie den globalen sichtbaren Bereich als Variable.

9.13.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                             VISIBLE AREA                                 --
6  --                                                                          --
7  --                               S p e c                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with gtk.widget;                use gtk.widget;
40 with demo_geometry;             use demo_geometry;
41
42
43 package demo_visible_area is
44
45     -- Returns the currently visible area of the model.
46     -- The visible area depends the current zoom factor,
47     -- base-offset, translate-offset, dimensions of the scrolled window
48     -- and the current settings of the scrollbars.
49     function get_visible_area (
50         canvas  : access gtk_widget_record'class)
51         return type_area;
52
53
54
55     -- This visible area is a global variable.
56     -- It is updated by procedure cb_draw_objects.
57     -- Some subprograms rely on it, for example those which
58     -- move the cursor. For this reason the visible area is
59     -- stored in a global variable.
60     -- For the future: If the operator searches for a
61     -- particular object, then the result of a search could be
62     -- a message like "The object is outside the visible
63     -- area at position (x/y)."
64     visible_area : type_area;
65
66
```

```
67
68      -- This procedure sets the translate-offset so that
69      -- the given area gets centered in the visible area.
70      -- The given area can be wider or higher than the visible area:
71      procedure center_to_visible_area (
72          area : in type_area);
73
74
75
76      -- In MODE_3_ZOOM_FIT, here the last visible area
77      -- immediately before the dimensions of the scrolled window
78      -- change, is stored as reference.
79      -- In other canvas modi it has no meaning:
80      last_visible_area : type_area;
81
82
83      -- This procedure takes an area and stores it in
84      -- the global variable last_visible_area (see above).
85      -- Its purpose is to be prepared to fit the area into the
86      -- scrolled window in MODE_3_ZOOM_FIT.
87      -- It must be called after operations that result in a
88      -- new visibible area. Such operations are:
89      -- - scrollbar released
90      -- - scroll up/down/right/left
91      -- - move cursor
92      -- - zoom on cursor
93      -- - zoom to fit all
94      -- - zoom to fit area
95      -- - zoom on mouse pointer
96      -- In in other canvas modi but MODE_3_ZOOM_FIT this
97      -- procedure has no meaning:
98      procedure backup_visible_area (
99          area : in type_area);
100
101
102 end demo_visible_area;
```

9.13.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                             VISIBLE AREA                                 --
6  --                                                                          --
7  --                               B o d y                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;                use ada.text_io;
40
41 with gtk.scrolled_window;        use gtk.scrolled_window;
42
43 with demo_logical_pixels;        use demo_logical_pixels;
44 with demo_translate_offset;      use demo_translate_offset;
45 with demo_zoom;                  use demo_zoom;
46 with demo_conversions;           use demo_conversions;
47 with demo_scrolled_window;       use demo_scrolled_window;
48 with demo_canvas;                use demo_canvas;
49
50
51 package body demo_visible_area is
52
53
54     function get_visible_area (
55         canvas  : access gtk_widget_record'class)
56         return type_area
57     is
58         result : type_area;
59
60         -- The allocation of the scrolled window:
61         W : gtk_allocation;
62
63         h_start, h_length, h_end : type_logical_pixels;
64         v_start, v_length, v_end : type_logical_pixels;
65
66         -- The four corners of the visible area:
67         BL, BR, TL, TR : type_vector_model;
68     begin
69         -- Inquire the allocation of the scrolled window
70         -- inside the main window:
71         get_allocation (swin, W);
72
73
74         -- X-AXIS:
```

```
75
76          -- The visible area along the x-axis starts at the
77          -- position of the horizontal scrollbar:
78          h_start  := to_lp (scrollbar_h_adj.get_value);
79
80          -- The visible area along the x-axis is as wide as
81          -- the scrolled window:
82          h_length := type_logical_pixels (W.width);
83
84          -- The visible area ends here:
85          h_end    := h_start + h_length;
86
87
88          -- Y-AXIS:
89
90          -- The visible area along the y-axis starts at the
91          -- position of the vertical scrollbar:
92          v_start := to_lp (scrollbar_v_adj.get_value);
93
94          -- The visible area along the y-axis is as high as
95          -- the scrolled window:
96          v_length := type_logical_pixels (W.height);
97
98          -- The visible area along the y-axis ends here:
99          v_end := v_start + v_length;
100
101
102         -- Compute the corners of the visible area.
103         -- The corners are real model coordinates:
104         BL := canvas_to_real ((h_start, v_end),   S);
105         BR := canvas_to_real ((h_end, v_end),     S);
106         TL := canvas_to_real ((h_start, v_start), S);
107         TR := canvas_to_real ((h_end, v_start),   S);
108
109         -- put_line ("BL " & to_string (BL));
110         -- put_line ("BR " & to_string (BR));
111         -- put_line ("TR " & to_string (TR));
112         -- put_line ("TL " & to_string (TL));
113
114         -- The position of the visible area is the lower left
115         -- corner:
116         result.position := BL;
117
118         -- Compute the width and the height of the
119         -- visible area:
120         result.width := TR.x - TL.x;
121         result.height := TL.y - BL.y;
122
123         -- CS: more effective ?
124         -- result.width    := type_distance_model
125         --      (h_length) * type_distance_model (S);
126         -- result.height   := type_distance_model
127         --      (v_length) * type_distance_model (S);
128
129         -- put_line ("visible area " & to_string (result));
130         return result;
131     end get_visible_area;
132
133
134
135     procedure center_to_visible_area (
136         area : in type_area)
137     is
138         -- debug : boolean := true;
139         debug : boolean := false;
140
141         -- The offset required to "move" all objects into
142         -- the center of the visible area:
143         dx, dy : type_distance_model;
144
145         -- Get the currently visible model area:
146         v : constant type_area := get_visible_area (canvas);
147
148         w1 : constant type_distance_model := v.width;
149         w2 : constant type_distance_model := area.width;
150
```

```
151         h1 : constant type_distance_model := v.height;
152         h2 : constant type_distance_model := area.height;
153
154         a, b : type_distance_model;
155
156         x0 : constant type_distance_model := area.position.x;
157         y0 : constant type_distance_model := area.position.y;
158
159         x1 : constant type_distance_model := v.position.x;
160         y1 : constant type_distance_model := v.position.y;
161
162         -- The given area will end up at this target position:
163         x2, y2 : type_distance_model;
164
165     begin
166         if debug then
167             put_line ("given␣␣␣" & to_string (area));
168             put_line ("visible␣" & to_string (v));
169         end if;
170
171         a := (w1 - w2) * 0.5;
172         x2 := x1 + a;
173         dx := x2 - x0;
174
175         b := (h1 - h2) * 0.5;
176         y2 := y1 + b;
177         dy := y2 - y0;
178
179         if debug then
180             put_line ("dx:" & to_string (dx));
181             put_line ("dy:" & to_string (dy));
182         end if;
183
184         -- Convert the model offset (dx;dy) to a canvas offset
185         -- and apply it to the global translate-offset.
186         -- Regarding y: T is in the canvas system (CS2)
187         -- where the y-axis goes downward. So we must multiply by -1:
188         T.x :=   type_logical_pixels (dx) * type_logical_pixels (S);
189         T.y := - type_logical_pixels (dy) * type_logical_pixels (S);
190         if debug then
191             put_line ("T:␣" & to_string (T));
192         end if;
193
194     end center_to_visible_area;
195
196
197
198     procedure backup_visible_area (
199         area : in type_area)
200     is begin
201         last_visible_area := area;
202     end backup_visible_area;
203
204
205 end demo_visible_area;
```

9.14 Das Paket demo_buttons

Dieses Paket ist für die Erzeugung der Knöpfe zuständig.

9.14.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                                BUTTONS                                   --
6  --                                                                          --
7  --                                S p e c                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with gtk.widget;                use gtk.widget;
40 with gtk.box;                   use gtk.box;
41 with gtk.button;                use gtk.button;
42 with gtk.table;                 use gtk.table;
43
44 package demo_buttons is
45
46     buttons_table       : gtk_table;
47
48     button_zoom_fit     : gtk_button;
49     button_zoom_area    : gtk_button;
50     button_move         : gtk_button;
51     button_add          : gtk_button;
52     button_delete       : gtk_button;
53     button_export       : gtk_button;
54
55     -- This procedure creates the buttons:
56     procedure create_buttons;
57
58 end demo_buttons;
```

9.14.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                               BUTTONS                                    --
6  --                                                                          --
7  --                               B o d y                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;                  use ada.text_io;
40
41 with glib;                         use glib;
42 with gtk.box;                      use gtk.box;
43 with demo_main_window;             use demo_main_window;
44
45
46 package body demo_buttons is
47
48     procedure create_buttons is begin
49         put_line ("create_buttons");
50
51         gtk_new_vbox (box_v0);
52         box_h.pack_start (box_v0, expand => false);
53
54
55         gtk_new (buttons_table, rows => 5, columns => 1,
56             homogeneous => false);
57         -- table.set_col_spacings (50);
58         -- table_coordinates.set_border_width (10);
59
60
61         gtk_new (button_zoom_fit, "ZOOM␣FIT");
62         gtk_new (button_zoom_area, "ZOOM␣AREA");
63         gtk_new (button_add, "ADD");
64         gtk_new (button_delete, "DELETE");
65         gtk_new (button_move, "MOVE");
66         gtk_new (button_export, "EXPORT");
67         -- CS add other buttons
68
69
70
71         -- The table shall not expand downward:
72         box_v0.pack_start (buttons_table, expand => false);
73
74
```

```
75          buttons_table.attach (button_zoom_fit,
76              left_attach => 0, right_attach => 1,
77              top_attach  => 0, bottom_attach => 1);
78
79          buttons_table.attach (button_zoom_area,
80              left_attach => 0, right_attach => 1,
81              top_attach  => 1, bottom_attach => 2);
82
83          buttons_table.attach (button_add,
84              left_attach => 0, right_attach => 1,
85              top_attach  => 2, bottom_attach => 3);
86
87          buttons_table.attach (button_delete,
88              left_attach => 0, right_attach => 1,
89              top_attach  => 3, bottom_attach => 4);
90
91          buttons_table.attach (button_move,
92              left_attach => 0, right_attach => 1,
93              top_attach  => 4, bottom_attach => 5);
94
95          buttons_table.attach (button_export,
96              left_attach => 0, right_attach => 1,
97              top_attach  => 5, bottom_attach => 6);
98
99
100     end create_buttons;
101
102 end demo_buttons;
```

9.15 Das Paket demo_conversions

Hier befinden sich alle Routinen für die Umrechnung zwischen den Koordinatensystemen CS1 und CS2. Desweiteren findet hier die Berechnung der Ecken des Begrenzungsrechtecks B in Leinwandkoordinaten statt.

9.15.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                              CONVERSIONS                                 --
6  --                                                                          --
7  --                               S p e c                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with demo_logical_pixels;       use demo_logical_pixels;
40 with demo_geometry;             use demo_geometry;
41 with demo_zoom;                 use demo_zoom;
42
43 package demo_conversions is
44
45 -- REAL (CS1) <-> VIRTUAL MODEL COORDINATES (CS2):
46
47     -- Converts a virtual model point to a real model point,
48     -- both in CS1:
49     function to_real (
50         point : in type_vector_model)
51         return type_vector_model;
52
53
54     -- Converts a real model point to a virtual model point,
55     -- both in CS1:
56     function to_virtual (
57         point : in type_vector_model)
58         return type_vector_model;
59
60
61
62 -- CANVAS (CS2 <-> MODEL (CS1):
63
64     -- Converts the given model distance to
```

```
65      -- a canvas distance according to the current zoom factor S:
66      function to_distance (
67          d : in type_distance_model_positive)
68          return type_logical_pixels_positive;
69
70
71      -- Converts the given canvas distance to
72      -- a model distance according to the current zoom factor S:
73      function to_distance (
74          d : in type_logical_pixels_positive)
75          return type_distance_model_positive;
76
77
78
79
80      -- Converts a given virtual model point (CS1)
81      -- to a point on the canvas (CS2) according to
82      -- the given zoom factor.
83      -- Optionally, if argument "translate" is true, then current
84      -- tranlate-offset will be taken into account.
85      function virtual_to_canvas (
86          V           : in type_vector_model;
87          zf          : in type_zoom_factor;
88          translate   : in boolean)
89          return type_logical_pixels_vector;
90
91      -- Converts a given canvas point (CS2) back
92      -- to a virtual model point (CS1) according
93      -- to the given zoom factor.
94      -- The current tranlate-offset is ALWAYS taken into account.
95      function canvas_to_virtual (
96          P           : in type_logical_pixels_vector;
97          zf          : in type_zoom_factor)
98          return type_vector_model;
99
100
101
102
103     -- Converts a real model point (CS1) to a canvas point (CS2)
104     -- according to the given zoom factor, the current
105     -- base-offset, the current translate-offset and
106     -- then the position of the current bounding-box:
107     function real_to_canvas (
108         M   : in type_vector_model;
109         zf  : in type_zoom_factor)
110         return type_logical_pixels_vector;
111
112     -- Converts a canvas point (CS2) back to a real model point (CS1)
113     -- according to the given zoom factor, the current
114     -- base-offset, the current translate-offset and
115     -- then the position of the current bounding-box:
116     function canvas_to_real (
117         P   : in type_logical_pixels_vector;
118         zf  : in type_zoom_factor)
119         return type_vector_model;
120
121
122
123
124
125     -- In connection with zoom-operations we need the corners of the
126     -- bounding-box in canvas coordinates. This composite type serves this
127     -- purpose:
128     type type_bounding_box_corners is record
129         BL, BR, TL, TR : type_logical_pixels_vector;
130     end record;
131
132
133     -- This function returns the current corners of the bounding-box
134     -- in canvas-coordinates. The return depends on the current zoom factor S
135     -- and translate-offset:
136     function get_bounding_box_corners
137         return type_bounding_box_corners;
138
139
140 end demo_conversions;
```

9.15.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                              CONVERSIONS                                 --
6  --                                                                          --
7  --                               B o d y                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;                 use ada.text_io;
40 with demo_bounding_box;           use demo_bounding_box;
41 with demo_base_offset;            use demo_base_offset;
42 with demo_translate_offset;       use demo_translate_offset;
43
44
45 package body demo_conversions is
46
47
48     function to_real (
49         point : in type_vector_model)
50         return type_vector_model
51     is
52         use demo_bounding_box;
53         result : type_vector_model := point;
54     begin
55         move_by (result, bounding_box.position);
56         return result;
57     end to_real;
58
59
60     function to_virtual (
61         point : in type_vector_model)
62         return type_vector_model
63     is
64         use demo_bounding_box;
65         result : type_vector_model := point;
66     begin
67         move_by (result, invert (bounding_box.position));
68         return result;
69     end to_virtual;
70
71
72
73     function to_distance (
74         d : in type_distance_model_positive)
```

```
75          return type_logical_pixels_positive
76      is begin
77          return type_logical_pixels (d) * type_logical_pixels (S);
78      end to_distance;
79
80
81      function to_distance (
82          d : in type_logical_pixels_positive)
83          return type_distance_model_positive
84      is begin
85          return type_distance_model_positive (d / type_logical_pixels (S));
86      end to_distance;
87
88
89
90      function virtual_to_canvas (
91          V           : in type_vector_model;
92          zf          : in type_zoom_factor;
93          translate   : in boolean)
94          return type_logical_pixels_vector
95      is
96          Z : type_logical_pixels_vector;
97      begin
98          Z.x :=  (type_logical_pixels (V.x) * type_logical_pixels (zf)
99                      + F.x);
100
101         Z.y := -(type_logical_pixels (V.y) * type_logical_pixels (zf)
102                     + F.y);
103
104         -- If required: Move Z by the current translate-offset:
105         if translate then
106             Z.x := Z.x + T.x;
107             Z.y := Z.y + T.y;
108         end if;
109
110         return Z;
111     end virtual_to_canvas;
112
113
114
115     function canvas_to_virtual (
116         P           : in type_logical_pixels_vector;
117         zf          : in type_zoom_factor)
118         return type_vector_model
119     is
120         result : type_vector_model;
121         -- debug : boolean := false;
122     begin
123         result.x := type_distance_model
124             (( (P.x - T.x) - F.x) / type_logical_pixels (zf));
125
126         result.y := type_distance_model
127             ((-(P.y - T.y) - F.y) / type_logical_pixels (zf));
128
129         return result;
130     end canvas_to_virtual;
131
132
133
134
135     function real_to_canvas (
136         M   : in type_vector_model;
137         zf  : in type_zoom_factor)
138         return type_logical_pixels_vector
139     is
140         V : type_vector_model;
141         Z : type_logical_pixels_vector;
142     begin
143         -- Convert the given real model point
144         -- to a virtual model point:
145         V := to_virtual (M);
146
147         -- Convert the virtual model point V to a
148         -- canvas point and take the current translate-offset
149         -- into account:
150         Z := virtual_to_canvas (V, zf, translate => true);
```

```
151
152         return Z;
153     end real_to_canvas;
154
155
156     function canvas_to_real (
157         P   : in type_logical_pixels_vector;
158         zf  : in type_zoom_factor)
159         return type_vector_model
160     is
161         M : type_vector_model;
162         debug : boolean := false;
163     begin
164         if debug then
165             put_line ("canvas_to_real");
166             put_line ("T " & to_string (T));
167         end if;
168
169         -- Convert the given canvas point to a virtual
170         -- model point:
171         M := canvas_to_virtual (P, zf);
172
173         -- Convert the virtual model point to
174         -- a real model point:
175         return to_real (M);
176
177         exception
178             when constraint_error =>
179                 put_line ("ERROR: conversion from canvas point "
180                     & "to model point failed !");
181                 put_line (" point " & to_string (P));
182                 put_line (" zf     " & to_string (zf));
183                 put_line (" T      " & to_string (T));
184                 put_line (" F      " & to_string (F));
185                 raise;
186     end canvas_to_real;
187
188
189
190
191     function get_bounding_box_corners
192         return type_bounding_box_corners
193     is
194         result : type_bounding_box_corners;
195
196         -- The corners of the given area in model-coordinates:
197         BC : constant type_area_corners := get_corners (bounding_box);
198
199     begin
200         -- Convert the corners of the bounding-box to canvas coordinates:
201         result.TL := real_to_canvas (BC.TL, S);
202         result.TR := real_to_canvas (BC.TR, S);
203         result.BL := real_to_canvas (BC.BL, S);
204         result.BR := real_to_canvas (BC.BR, S);
205
206         return result;
207     end get_bounding_box_corners;
208
209 end demo_conversions;
```

9.16 Das Paket demo_cursor

Sofern ein Cursor verwendet wird, ist hier dessen Modellierung, Verschiebeoperationen und dessen Darstellung auf der Leinwand kodiert.

9.16.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                                CURSOR                                    --
6  --                                                                          --
7  --                                S p e c                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with demo_logical_pixels;       use demo_logical_pixels;
40 with demo_geometry;             use demo_geometry;
41
42
43 package demo_cursor is
44
45     -- The cursor is a crosshair that can be moved by the
46     -- cursor keys (arrow keys) about the canvas:
47     type type_cursor is record
48         position    : type_vector_model := origin;
49
50         -- For drawing the cursor:
51         linewidth_1 : type_logical_pixels_positive := 1.0;
52         linewidth_2 : type_logical_pixels_positive := 4.0;
53         length_1    : type_logical_pixels_positive := 20.0;
54         length_2    : type_logical_pixels_positive := 20.0;
55         radius      : type_logical_pixels_positive := 25.0;
56
57         -- CS: blink, color, ...
58     end record;
59
60
61     -- This is the instance of the cursor:
62     cursor : type_cursor;
63
64
65     -- This procedure moves the cursor to the given destination:
66     procedure move_cursor (
```

```
67          destination : type_vector_model);
68
69
70      -- This procedure moves the cursor into the given direction:
71      procedure move_cursor (
72          direction : type_direction);
73
74
75      -- This procedure draws the cursor at its current
76      -- position. To keep things simple, the cursor is
77      -- drawn always, regardless whether it is in the visible
78      -- area or not:
79      procedure draw_cursor;
80
81
82  end demo_cursor;
```

9.16.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                                CURSOR                                    --
6  --                                                                          --
7  --                                B o d y                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;               use ada.text_io;
40 with cairo;
41
42 with demo_zoom;
43 with demo_conversions;
44 with demo_visible_area;
45 with demo_grid;
46 with demo_canvas;
47 with demo_coordinates_display;  use demo_coordinates_display;
48
49
50
51 package body demo_cursor is
52
53     procedure move_cursor (
54         destination : type_vector_model)
55     is begin
56         cursor.position := destination;
57         update_cursor_coordinates;
58         update_distances_display;
59
60         -- Output the cursor position on the terminal:
61         put_line ("position " & to_string (cursor.position));
62     end move_cursor;
63
64
65
66     procedure move_cursor (
67         direction : type_direction)
68     is
69         use demo_visible_area;
70         use demo_grid;
71         use demo_canvas;
72     begin
73         -- Move the cursor by the grid spacing into the given direction:
74         put_line ("move cursor " & type_direction'image (direction));
```

```
75
76          case direction is
77              when DIR_RIGHT =>
78                  cursor.position.x := cursor.position.x + grid.spacing.x;
79
80              when DIR_LEFT =>
81                  cursor.position.x := cursor.position.x - grid.spacing.x;
82
83              when DIR_UP =>
84                  cursor.position.y := cursor.position.y + grid.spacing.y;
85
86              when DIR_DOWN =>
87                  cursor.position.y := cursor.position.y - grid.spacing.y;
88          end case;
89
90          -- CS Limit cursor position to range of type_distance_model
91          -- Exception handler ?
92
93          -- If the cursor is outside the visible area, then the
94          -- canvas must be shifted with the cursor:
95          if not in_area (cursor.position, visible_area) then
96              put_line ("cursor not in visible area");
97
98              case direction is
99                  when DIR_RIGHT =>
100                     -- If the cursor is right of the visible area,
101                     -- then shift the canvas to the right:
102                     if cursor.position.x >
103                         visible_area.position.x + visible_area.width then
104                         shift_swin (direction, grid.spacing.x);
105                     end if;
106
107                 when DIR_LEFT =>
108                     -- If the cursor is left of the visible area,
109                     -- then shift the canvas to the left:
110                     if cursor.position.x < visible_area.position.x then
111                         shift_swin (direction, grid.spacing.x);
112                     end if;
113
114                 when DIR_UP =>
115                     -- If the cursor is above of the visible area,
116                     -- then shift the canvas up:
117                     if cursor.position.y >
118                         visible_area.position.y + visible_area.height then
119                         shift_swin (direction, grid.spacing.y);
120                     end if;
121
122                 when DIR_DOWN =>
123                     -- If the cursor is below of the visible area,
124                     -- then shift the canvas down:
125                     if cursor.position.y < visible_area.position.y then
126                         shift_swin (direction, grid.spacing.y);
127                     end if;
128
129             end case;
130
131         end if;
132
133         refresh;
134
135         update_cursor_coordinates;
136         update_distances_display;
137
138         -- Output the cursor position on the terminal:
139         put_line ("cursor at " & to_string (cursor.position));
140
141         backup_visible_area (get_visible_area (canvas));
142     end move_cursor;
143
144
145
146     procedure draw_cursor is
147         use cairo;
148         use demo_canvas;
149         use demo_conversions;
150         use demo_zoom;
```

```
151
152         cp : type_logical_pixels_vector :=
153             real_to_canvas (cursor.position, S);
154
155         -- These are the start and stop positions for the
156         -- horizontal lines:
157         h1, h2, h3, h4 : type_logical_pixels;
158
159         -- These are the start and stop positions for the
160         -- vertical lines:
161         v1, v2, v3, v4 : type_logical_pixels;
162
163         -- This is the total length of an arm:
164         l : constant type_logical_pixels :=
165             cursor.length_1 + cursor.length_2;
166
167     begin
168         set_source_rgb (context, 0.5, 0.5, 0.5); -- gray
169
170         -- Compute the start and stop positions:
171         h1 := cp.x - l;
172         h2 := cp.x - cursor.length_1;
173         h3 := cp.x + cursor.length_1;
174         h4 := cp.x + l;
175
176         v1 := cp.y - l;
177         v2 := cp.y - cursor.length_1;
178         v3 := cp.y + cursor.length_1;
179         v4 := cp.y + l;
180
181         -- Draw the horizontal line from left to right:
182         -- thick
183         set_line_width (context, to_gdouble (cursor.linewidth_2));
184         move_to (context, to_gdouble (h1), to_gdouble (cp.y));
185         line_to (context, to_gdouble (h2), to_gdouble (cp.y));
186         stroke;
187
188         -- thin
189         set_line_width (context, to_gdouble (cursor.linewidth_1));
190         move_to (context, to_gdouble (h2), to_gdouble (cp.y));
191         line_to (context, to_gdouble (h3), to_gdouble (cp.y));
192         stroke;
193
194         -- thick
195         set_line_width (context, to_gdouble (cursor.linewidth_2));
196         move_to (context, to_gdouble (h3), to_gdouble (cp.y));
197         line_to (context, to_gdouble (h4), to_gdouble (cp.y));
198         stroke;
199
200         -- Draw the vertical line from top to bottom:
201         -- thick
202         move_to (context, to_gdouble (cp.x), to_gdouble (v1));
203         line_to (context, to_gdouble (cp.x), to_gdouble (v2));
204         stroke;
205
206         -- thin
207         set_line_width (context, to_gdouble (cursor.linewidth_1));
208         move_to (context, to_gdouble (cp.x), to_gdouble (v2));
209         line_to (context, to_gdouble (cp.x), to_gdouble (v3));
210         stroke;
211
212         -- thick
213         set_line_width (context, to_gdouble (cursor.linewidth_2));
214         move_to (context, to_gdouble (cp.x), to_gdouble (v3));
215         line_to (context, to_gdouble (cp.x), to_gdouble (v4));
216         stroke;
217
218         -- arc
219         set_line_width (context, to_gdouble (cursor.linewidth_1));
220         arc (context, to_gdouble (cp.x), to_gdouble (cp.y),
221                 radius => to_gdouble (cursor.radius),
222                 angle1 => 0.0, angle2 => 6.3);
223
224         stroke;
225
226         -- CS: To improve performance on drawing, it might help
```

```
227          -- to draw all objects which have a thin line first, then
228          -- all object with a thick line.
229      end draw_cursor;
230
231
232 end demo_cursor;
```

9.17 Das Paket demo_drawing_origin

In diesem Paket ist der Zeichnungsursprung modelliert. Es enthält die Prozedur zum Zeichnen des Ursprungs auf der Leinwand.

9.17.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                            DRAWING ORIGIN                                --
6  --                                                                          --
7  --                                S p e c                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with demo_logical_pixels;       use demo_logical_pixels;
40
41
42 package demo_drawing_origin is
43
44     -- The origin is a small cross at model position (0;0).
45
46     -- the arm-length:
47     origin_size         : constant type_logical_pixels_positive := 10.0;
48     origin_linewidth    : constant type_logical_pixels_positive := 1.0;
49
50
51     -- This procedure draws the origin. The origin is a small
52     -- cross at model position (0;0). This procedure does not
53     -- check whether the lines of the cross are inside the visible
54     -- area. Since it is about two simple lines we draw them
55     -- always:
56     procedure draw_origin;
57
58 end demo_drawing_origin;
```

9.17.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                             DRAWING ORIGIN                               --
6  --                                                                          --
7  --                                B o d y                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with cairo;
40 with demo_canvas;
41 with demo_zoom;
42 with demo_conversions;
43 with demo_geometry;              use demo_geometry;
44
45
46 package body demo_drawing_origin is
47
48
49     procedure draw_origin is
50         use cairo;
51         use demo_canvas;
52         use demo_zoom;
53         use demo_conversions;
54
55         cp : type_logical_pixels_vector := real_to_canvas (origin, S);
56     begin
57         set_source_rgb (context, 0.5, 0.5, 0.5); -- gray
58         set_line_width (context, to_gdouble (origin_linewidth));
59
60         -- Draw the horizontal line from left to right:
61         move_to (context,
62             to_gdouble (cp.x - origin_size), to_gdouble (cp.y));
63
64         line_to (context,
65             to_gdouble (cp.x + origin_size), to_gdouble (cp.y));
66
67         -- Draw the vertical line from top to bottom:
68         move_to (context,
69             to_gdouble (cp.x), to_gdouble (cp.y - origin_size));
70
71         line_to (context,
72                 to_gdouble (cp.x), to_gdouble (cp.y + origin_size));
73
74         stroke;
```

```
75      end draw_origin;
76
77
78
79 end demo_drawing_origin;
```

9.18 Das Paket demo_geometry

Dieses Paket enthält die Typdeklarationen und Spezifikationen von Entfernungen, Punkten, Vektoren und Flächen in der Modelldomäne (CS1). Darüber hinaus sind hier Routinen für Operationen mit Flächen zu finden. Die Typdeklaration von Winkeln ist hier vorbereitet, wird aber im gegenwärtigen Stand nicht verwendet.

9.18.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                                DEMO CANVAS                               --
4  --                                                                          --
5  --                                 GEOMETRY                                 --
6  --                                                                          --
7  --                                 S p e c                                  --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --    For correct displaying set tab width in your editor to 4.
27
28 --    The two letters "CS" indicate a "construction site" where things are not
29 --    finished yet or intended for the future.
30
31 --    Please send your questions and comments to:
32 --
33 --    info@blunk-electronic.de
34 --    or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --    history of changes:
37 --
38
39 with ada.numerics;
40 with ada.numerics.generic_elementary_functions;
41
42
43 package demo_geometry is
44
45     -- The directions into which the an object can be moved
46     -- by means of the cursor keys (arrow keys):
47     type type_direction is (DIR_RIGHT, DIR_LEFT, DIR_UP, DIR_DOWN);
48
49
50
51
52 -- INTERNAL FLOAT TYPE:
53
54     -- This float type is used for internal computations only:
55     type type_float is new float; -- CS refinement required
56
57
58     package pac_float_numbers_functions is new
59         ada.numerics.generic_elementary_functions (type_float);
60
61
62
63 -- DISTANCE:
```

```
64
65      -- The model coordinates system uses so called
66      -- ordinary binary fixed point numbers for distances and positions:
67      type type_distance_model is delta 0.01
68          range -100_000.00 .. 100_000.00;
69      for type_distance_model'small use 0.01;
70
71      -- Use this type for distances, lengths, ...
72      -- Because those things require positive numbers:
73      subtype type_distance_model_positive is type_distance_model
74          range 0.0 .. type_distance_model'last;
75
76
77      -- This function returns the given distance as string:
78      function to_string (
79          distance : in type_distance_model)
80          return string;
81
82
83
84  -- ROTATION / ANGLE:
85
86      -- The model coordinates system uses so called
87      -- ordinary binary fixed point numbers for angles and rotations:
88      rotation_smallest : constant := 0.01;
89      type type_rotation_model is delta rotation_smallest
90          range -360.0 + rotation_smallest .. 360.0 - rotation_smallest;
91      for type_rotation_model'small use rotation_smallest;
92
93      -- Converts the given rotation/angle to a string:
94      function to_string (
95          rotation : in type_rotation_model)
96          return string;
97
98
99
100
101 -- POINT / POSITION / LOCATION / LOCATION VECTOR / DISTANCE VECTOR:
102
103     type type_vector_model is record
104         x, y : type_distance_model := 0.0;
105     end record;
106
107
108     -- This function returns the given vector
109     -- as string:
110     function to_string (
111         v : in type_vector_model)
112         return string;
113
114
115     -- This function inverts a vector by multiplying
116     -- its components by -1:
117     function invert (
118         point   : in type_vector_model)
119         return type_vector_model;
120
121
122     -- Moves a model point by the given offset:
123     procedure move_by (
124         point   : in out type_vector_model;
125         offset  : in type_vector_model);
126
127
128
129
130     -- Returns the absolute distance between the given
131     -- model points. Uses internally a float type:
132     function get_distance (
133         p1, p2 : in type_vector_model)
134         return type_distance_model_positive;
135
136
137     -- Returns the angle of direection from the given
138     -- point p1 to the point p2. Uses internally a float type:
139     function get_angle (
```

```
140         p1, p2 : in type_vector_model)
141         return type_rotation_model;
142
143
144
145 -- ORIGIN:
146
147     -- The origin is a small cross at model position (0;0).
148     origin : constant type_vector_model := (0.0, 0.0);
149
150
151
152
153
154 -- AREA:
155
156     type type_area is record
157         width       : type_distance_model_positive := 0.0;
158         height      : type_distance_model_positive := 0.0;
159         position    : type_vector_model; -- lower left corner
160     end record;
161
162
163     -- Returns the position and dimensions of the given area as string:
164     function to_string (
165         box : in type_area)
166         return string;
167
168
169     -- In order to handle the four corners of an
170     -- area this type is required:
171     type type_area_corners is record
172         BL, BR, TL, TR : type_vector_model;
173     end record;
174
175
176     -- Returns the four corners of the given area.
177     -- The area is given in model coordinates:
178     function get_corners (
179         area    : in type_area)
180         return type_area_corners;
181
182
183     -- Returns the center of the given area:
184     function get_center (
185         area    : in type_area)
186         return type_vector_model;
187
188
189
190     -- Returns true if the given point lies inside the given
191     -- area or on its border.
192     function in_area (
193         point   : in type_vector_model;
194         area    : in type_area)
195         return boolean;
196
197
198     -- Returns true if the given areas overlap each other:
199     function areas_overlap (
200         A, B : in type_area)
201         return boolean;
202
203
204     -- Merges the given area B into area A:
205     procedure merge_areas (
206         A : in out type_area;
207         B : in type_area);
208
209
210 end demo_geometry;
```

9.18.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                               GEOMETRY                                   --
6  --                                                                          --
7  --                               B o d y                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;               use ada.text_io;
40
41
42 package body demo_geometry is
43
44
45     function to_string (
46         distance : in type_distance_model)
47         return string
48     is begin
49         return type_distance_model'image (distance);
50     end to_string;
51
52
53     function to_string (
54         rotation : in type_rotation_model)
55         return string
56     is begin
57         return type_rotation_model'image (rotation);
58     end to_string;
59
60
61
62     function to_string (
63         v : in type_vector_model)
64         return string
65     is begin
66         return "x/y: "
67             & to_string (v.x) & "/" & to_string (v.y);
68     end to_string;
69
70
71     function invert (
72         point   : in type_vector_model)
73         return type_vector_model
74     is begin
```

```
75          return (- point.x, - point.y);
76      end invert;
77
78
79
80      procedure move_by (
81          point   : in out type_vector_model;
82          offset  : in type_vector_model)
83      is begin
84          point.x := point.x + offset.x;
85          point.y := point.y + offset.y;
86      end move_by;
87
88
89
90      function get_distance (
91          p1, p2 : in type_vector_model)
92          return type_distance_model_positive
93      is
94          use pac_float_numbers_functions;
95
96          dx : type_float := abs (type_float (p2.x - p1.x));
97          dy : type_float := abs (type_float (p2.y - p1.y));
98          d : type_float;
99      begin
100         d := sqrt (dx**2.0 + dy**2.0);
101         return type_distance_model_positive (d);
102     end get_distance;
103
104
105     function get_angle (
106         p1, p2 : in type_vector_model)
107         return type_rotation_model
108     is
109         use pac_float_numbers_functions;
110
111         dx : type_float := type_float (p2.x - p1.x);
112         dy : type_float := type_float (p2.y - p1.y);
113         a : type_float;
114     begin
115         -- For a tangens operations, dx must not
116         -- be zero. If it is zero, then dy determines
117         -- whether the result is 90 or -90 degree:
118         if dx /= 0.0 then
119             a := arctan (dy, dx, 360.0);
120         else
121             if dy > 0.0 then
122                 a := 90.0;
123             else
124                 a := -90.0;
125             end if;
126         end if;
127
128         return type_rotation_model (a);
129
130         exception
131             when ADA.NUMERICS.ARGUMENT_ERROR =>
132                 put_line ("tangens␣error");
133                 raise;
134
135     end get_angle;
136
137
138
139
140     function to_string (
141         box : in type_area)
142         return string
143     is begin
144         return "(x/y/w/h):␣"
145             & to_string (box.position.x) & "/"
146             & to_string (box.position.y) & "/"
147             & to_string (box.width) & "/"
148             & to_string (box.height);
149     end to_string;
150
```

```
151
152     function get_corners (
153         area    : in type_area)
154         return type_area_corners
155     is
156         result : type_area_corners;
157     begin
158         result.BL := (area.position.x, area.position.y);
159         result.BR := (area.position.x + area.width, area.position.y);
160
161         result.TL := (area.position.x, area.position.y + area.height);
162         result.TR := (area.position.x + area.width,
163                       area.position.y + area.height);
164         return result;
165     end get_corners;
166
167
168
169     function get_center (
170         area    : in type_area)
171         return type_vector_model
172     is
173         result : type_vector_model;
174     begin
175         result.x := area.position.x + area.width  * 0.5;
176         result.y := area.position.y + area.height * 0.5;
177         return result;
178     end get_center;
179
180
181
182     function in_area (
183         point   : type_vector_model;
184         area    : type_area)
185         return boolean
186     is
187         result : boolean := false;
188     begin
189         -- text x-axis:
190         if point.x >= area.position.x then
191             if point.x <= area.position.x + area.width then
192
193                 -- test y-axis:
194                 if point.y >= area.position.y then
195                     if point.y <= area.position.y + area.height then
196                         result := true;
197                     end if;
198                 end if;
199
200             end if;
201         end if;
202
203         return result;
204     end in_area;
205
206
207
208
209
210
211     function areas_overlap (
212         A, B : in type_area)
213         return boolean
214     is
215         -- CS: Optimization required. Compiler options ?
216         -- CS: rename lx, gx, ly, gy to x1, x2, y1, y2
217
218         -- AREA A:
219         -- This is the lowest x used by area A
220         A_lx : type_distance_model renames A.position.x;
221
222         -- This is the greatest x used by area A
223         A_gx : constant type_distance_model := A_lx + A.width;
224
225
226         -- This is the lowest y used by area A
```

```
227         A_ly : type_distance_model renames A.position.y;
228
229         -- This is the greatest y used by area A
230         A_gy : constant type_distance_model := A_ly + A.height;
231
232
233         -- AREA B:
234         -- This is the lowest x used by area B
235         B_lx : type_distance_model renames B.position.x;
236
237         -- This is the greatest x used by area B
238         B_gx : constant type_distance_model := B_lx + B.width;
239
240
241         -- This is the lowest y used by area B
242         B_ly : type_distance_model renames B.position.y;
243
244         -- This is the greatest y used by area B
245         B_gy : constant type_distance_model := B_ly + B.height;
246
247     begin
248         -- If all of the four criteria are true then the two
249         -- areas DO overlap:
250         if  B_lx < A_gx
251         and B_gx > A_lx
252         and B_ly < A_gy
253         and B_gy > A_ly then
254             return true;
255         else
256             return false;
257         end if;
258     end areas_overlap;
259
260
261     procedure merge_areas (
262         A : in out type_area;
263         B : in type_area)
264     is
265         -- CS: Optimization required. Compiler options ?
266
267         -- AREA A:
268         -- This is the lowest x used by area A
269         A_lx : type_distance_model renames A.position.x;
270
271         -- This is the greatest x used by area A
272         A_gx : type_distance_model := A_lx + A.width;
273
274
275         -- This is the lowest y used by area A
276         A_ly : type_distance_model renames A.position.y;
277
278         -- This is the greatest y used by area A
279         A_gy : type_distance_model := A_ly + A.height;
280
281
282         -- AREA B:
283         -- This is the lowest x used by area B
284         B_lx : type_distance_model renames B.position.x;
285
286         -- This is the greatest x used by area B
287         B_gx : type_distance_model := B_lx + B.width;
288
289
290         -- This is the lowest y used by area B
291         B_ly : type_distance_model renames B.position.y;
292
293         -- This is the greatest y used by area B
294         B_gy : type_distance_model := B_ly + B.height;
295
296     begin
297         -- x-axis:
298         if B_lx < A_lx then
299             A_lx := B_lx;
300         end if;
301
302         if B_gx > A_gx then
```

```
303             A_gx := B_gx;
304         end if;
305
306         -- y-axis:
307         if B_ly < A_ly then
308             A_ly := B_ly;
309         end if;
310
311         if B_gy > A_gy then
312             A_gy := B_gy;
313         end if;
314
315         A.width  := A_gx - A_lx;
316         A.height := A_gy - A_ly;
317     end merge_areas;
318
319 end demo_geometry;
```

9.19 Das Paket demo_logical_pixels

GTK3 verwendet von Haus aus für die logischen Pixel den Zahlentyp `gdouble`. Von diesem Typ werden in diesem Paket die Zahlentypen `type_logical_pixels` und `type_logical_pixels_positive` abgeleitet und entsprechende Routinen zum Konvertieren angeboten. Sämtliche Zeichenoperationen auf der Leinwand und Einstellungen von Rollbalken erfolgen mit dem Typ `type_logical_pixels_positive` weil die Leinwand und die Rollbalken nur positive Zahlen kennen. Ebenso basieren Vektoren (oder Punkte) auf der Leinwand auf dem Typ `type_logical_pixels_positive`.

9.19.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                                DEMO CANVAS                               --
4  --                                                                          --
5  --                              LOGICAL PIXELS                              --
6  --                                                                          --
7  --                                 S p e c                                  --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with glib;
40
41 package demo_logical_pixels is
42
43     -- GTK3 uses the type gdouble for primitive draw operations
44     -- on the canvas. It also uses gdouble for scrollbar settings.
45     -- A point, a vector or a distance is expressed in
46     -- so called "logical pixels".
47
48     -- We derive a new type from glib.gdouble in order to
49     -- get a clear separation from things defined in glib:
50     type type_logical_pixels is new glib.gdouble;
51
52
53     -- Converts logical pixels to a human readable string:
54     function to_string (
55         lp : in type_logical_pixels)
56         return string;
57
58
```

```
59      -- This function converts a gdouble number
60      -- to logical pixels:
61      function to_lp (
62          gd : in glib.gdouble)
63          return type_logical_pixels;
64
65
66      -- This function converts logical pixels
67      -- to a gdouble number:
68      function to_gdouble (
69          lp : in type_logical_pixels)
70          return glib.gdouble;
71
72
73      -- Use this type for distances, lengths, scrollbar settings,
74      -- primitive draw operations, ...
75      -- because such things are always positive numbers:
76      subtype type_logical_pixels_positive is type_logical_pixels
77          range 0.0 .. type_logical_pixels'last;
78
79
80      -- This function converts positive logical pixels
81      -- to a positive gdouble number:
82      function to_gdouble_positive (
83          lp : in type_logical_pixels_positive)
84          return glib.gdouble;
85
86
87      -- A point, a location vector or a distance vector is
88      -- defined by this type:
89      type type_logical_pixels_vector is record
90          x, y : type_logical_pixels := 0.0;
91      end record;
92
93
94      -- This function outputs the x and y component of a vector
95      -- on the console:
96      function to_string (
97          v : in type_logical_pixels_vector)
98          return string;
99
100
101
102
103     -- Clips the given value by the given limit.
104     -- If the given value is less or equal the limit,
105     -- then value remains unchanged:
106     procedure clip_max (
107         value   : in out type_logical_pixels;
108         limit   : in type_logical_pixels);
109
110     -- Clips the given value by the given limit.
111     -- If the given value is greater or equal the limit,
112     -- then value remains unchanged:
113     procedure clip_min (
114         value   : in out type_logical_pixels;
115         limit   : in type_logical_pixels);
116
117
118 end demo_logical_pixels;
```

9.19.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                            LOGICAL PIXELS                                --
6  --                                                                          --
7  --                               B o d y                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;                use ada.text_io;
40
41
42 package body demo_logical_pixels is
43
44     function to_string (
45         lp : in type_logical_pixels)
46         return string
47     is begin
48         return type_logical_pixels'image (lp);
49     end to_string;
50
51
52
53     function to_lp (
54         gd : in glib.gdouble)
55         return type_logical_pixels
56     is begin
57         return type_logical_pixels (gd);
58     end to_lp;
59
60
61     function to_gdouble (
62         lp : in type_logical_pixels)
63         return glib.gdouble
64     is begin
65         return glib.gdouble (lp);
66     end to_gdouble;
67
68
69     function to_gdouble_positive (
70         lp : in type_logical_pixels_positive)
71         return glib.gdouble
72     is begin
73         return glib.gdouble (lp);
74     end to_gdouble_positive;
```

```
75
76
77      function to_string (
78          v : in type_logical_pixels_vector)
79          return string
80      is begin
81          --return "vector logical pixels x/y: "
82          return to_string (v.x) & "/"
83              & to_string (v.y);
84      end to_string;
85
86
87
88      procedure clip_max (
89          value   : in out type_logical_pixels;
90          limit   : in type_logical_pixels)
91      is begin
92          if value > limit then
93              value := limit;
94          end if;
95      end clip_max;
96
97
98      procedure clip_min (
99          value   : in out type_logical_pixels;
100         limit   : in type_logical_pixels)
101     is begin
102         if value < limit then
103             value := limit;
104         end if;
105     end clip_min;
106
107
108 end demo_logical_pixels;
```

9.20 Das Paket demo_primitive_draw_ops

Sämtliche Routinen zum Zeichnen von Linien und Kreisen (primitive Objekte) sind hier umgesetzt.

9.20.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                                DEMO CANVAS                               --
4  --                                                                          --
5  --                           PRIMITIVE DRAW OPERATIONS                      --
6  --                                                                          --
7  --                                  S p e c                                 --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with demo_geometry;          use demo_geometry;
40 with demo_objects;           use demo_objects;
41
42
43 package demo_primitive_draw_ops is
44
45     -- This is a primitive draw operation that draws a line.
46     -- The argument pos contains the position
47     -- of the parent complex object.
48     -- Regarding the argument do_stroke there are two modes:
49     -- 1. If the argument do_stroke is false (default) then
50     --  no setting of linewidth and no stroking will be done. In this
51     --  case it is assumed that the caller has already set a linewidth
52     --  and that the caller will later care for a stroke command.
53     --  The given linewidth has no meaning in this case. This mode
54     --  requires less time for drawing the line than with do_stroke enabled
55     --  and should be used when many lines of same linewidth have to be drawn.
56     --  An explicit call of the stroke procedure is required finally.
57     -- 2. If do_stroke is true, then the given linewidth is applied,
58     --  the line drawn and finally a stroke command executed.
59     procedure draw_line (
60         line        : in type_line;
61         pos         : in type_vector_model;
62         -- CS: default origin (0;0) in case there is no parent object ?
63         do_stroke   : in boolean := false);
64
65
66     -- This is a primitive draw operation that draws a circle.
```

```
67      -- For arguments see draw_line:
68      procedure draw_circle (
69          circle      : in type_circle;
70          pos         : in type_vector_model;
71          -- CS: default origin (0;0) in case there is no parent object ?
72          do_stroke   : in boolean := false);
73
74
75  end demo_primitive_draw_ops;
```

9.20.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                         PRIMITIVE DRAW OPERATIONS                        --
6  --                                                                          --
7  --                                 B o d y                                  --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;                use ada.text_io;
40
41 -- with glib;                        use glib;
42 with cairo;                       use cairo;
43
44 with demo_logical_pixels;         use demo_logical_pixels;
45 with demo_zoom;                   use demo_zoom;
46 with demo_conversions;            use demo_conversions;
47 with demo_visible_area;           use demo_visible_area;
48 with demo_visibility;             use demo_visibility;
49 with demo_canvas;                 use demo_canvas;
50
51
52 package body demo_primitive_draw_ops is
53
54     procedure draw_line (
55         line         : in type_line;
56         pos          : in type_vector_model;
57         do_stroke    : in boolean := false)
58     is
59         -- Make a copy of the given line:
60         l : type_line := line;
61
62         -- When the line is drawn, we need canvas points
63         -- for start and end:
64         c1, c2 : type_logical_pixels_vector; -- start and end of the line
65
66         -- The bounding-box of the line. It is required
67         -- for the area and size check:
68         b : type_area;
69
70     begin
71         -- CS: If the line is to be rotated about its origin,
72         -- then do the rotation here.
73
74         -- Move the line to the given position:
```

```
75          move_line (l, pos);
76
77          -- Get the bounding-box of line:
78          b := get_bounding_box (l);
79          -- put_line ("b " & to_string (b));
80
81          -- Do the area check. If the bounding-box of the line
82          -- is inside the visible area then draw the line. Otherwise
83          -- nothing will be drawn:
84          if areas_overlap (visible_area, b) and then
85
86              -- Do the size check. If the bounding-box is greater
87              -- (either in width or heigth) than the visiblity threshold
88              -- then draw the line. Otherwise nothing will be drawn:
89              above_visibility_threshold (b) then
90
91              -- If an individual stroke is requested for
92              -- the given line, then set the linewidth:
93              if do_stroke then
94                  set_line_width (context,
95                                  to_gdouble_positive (to_distance (line.w)));
96              end if;
97
98              c1 := real_to_canvas (l.s, S);
99              c2 := real_to_canvas (l.e, S);
100
101             -- THESE DRAW OPERATIONS CONSUME THE MOST TIME:
102             move_to (context,
103                      to_gdouble_positive (c1.x), to_gdouble_positive (c1.y));
104
105             line_to (context,
106                      to_gdouble_positive (c2.x), to_gdouble_positive (c2.y));
107
108             -- Direct conversion to gdouble does not improve performance:
109             -- move_to (context, gdouble (c1.x), gdouble (c1.y));
110             -- line_to (context, gdouble (c2.x), gdouble (c2.y));
111
112             -- If an individual stroke is requested for
113             -- the given line, then do it now:
114             if do_stroke then
115                 stroke (context);
116             end if;
117
118
119             -- CS: use OpenGL ?
120         end if;
121     end draw_line;
122
123
124     procedure draw_circle (
125         circle      : in type_circle;
126         pos         : in type_vector_model;
127         do_stroke   : in boolean := false)
128     is
129         -- Make a copy of the given circle:
130         c : type_circle := circle;
131
132         -- When the circle is drawn, we need a canvas point
133         -- for the center:
134         m : type_logical_pixels_vector;
135
136         r : type_logical_pixels_positive;
137
138         -- The bounding-box of the circle. It is required
139         -- for the area and size check:
140         b : type_area;
141
142     begin
143         -- CS: Since this is a circle, a rotation about
144         -- its origin can be omitted here.
145
146         -- Move the circle to the given position:
147         move_circle (c, pos);
148
149         -- Get the bounding-box of circle:
150         b := get_bounding_box (c);
```

```
151         -- put_line ("b " & to_string (b));
152
153         -- Do the area check. If the bounding-box of the circle
154         -- is inside the visible area then draw the circle. Otherwise
155         -- nothing will be drawn:
156         if areas_overlap (visible_area, b) and then
157
158             -- Do the size check. If the bounding-box is greater
159             -- (either in width or heigth) than the visiblity threshold
160             -- then draw the line. Otherwise nothing will be drawn:
161             above_visibility_threshold (b) then
162
163             -- put_line ("draw circle");
164
165             -- If an individual stroke is requested for
166             -- the given circle, then set the linewidth of the
167             -- circumfence:
168             if do_stroke then
169                 set_line_width (context,
170                                 to_gdouble_positive (to_distance (circle.w)));
171             end if;
172
173             m := real_to_canvas (c.c, S);
174             r := to_distance (c.r);
175
176             -- required to suppress an initial line:
177             -- CS new_sub_path (context);
178
179
180             -- THIS DRAW OPERATION CONSUMES THE MOST TIME:
181             arc (context,
182                  to_gdouble_positive (m.x),
183                  to_gdouble_positive (m.y),
184                  to_gdouble_positive (r),
185                  0.0, 6.3 ); -- start and end angle in radians
186
187
188             -- If an individual stroke is requested for
189             -- the given circle, then do it now:
190             if do_stroke then
191                 stroke (context);
192             end if;
193
194
195             -- CS: use OpenGL ?
196         end if;
197     end draw_circle;
198
199
200 end demo_primitive_draw_ops;
```

9.21 Das Paket demo_scale

Grundfunktionen betreffend des Maßstabes sind in diesem Paket zu finden:

1. Typdeklaration des Maßstabes
2. globale Variable M
3. Umrechnung zwischen Original (oder Realität) und Modell

9.21.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                                SCALE                                     --
6  --                                                                          --
7  --                               S p e c                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.text_io;               use ada.text_io;
40
41 with demo_geometry;             use demo_geometry;
42
43
44 package demo_scale is
45
46     -- The scale is a floating point type.
47     -- Its ranges are defined here:
48     type type_scale is digits 3 range 0.01 .. 100.0;
49     -- If you intend to change this declaration, please see the
50     -- comments in function to_string.
51
52     -- This is the global scale:
53     M : type_scale := 1.0;
54     -- use it for the rectangle, triangle and circle
55
56     -- M : type_scale := 50.0;
57     -- use it for the bridge example
58
59     -- M : type_scale := 0.1;
60     -- M : type_scale := 0.04;
```

```
61      --use it for the screw example
62
63      -- Examples for usage:
64      -- 1)
65      -- If M is set to 10 then the scale is 1:10 (in words one to ten).
66      -- This means: A distance of 1mm in the model represents
67      -- a distance of 10mm in the real world.
68      -- The drawing shows the reality downsized.
69
70      -- 2)
71      -- If M is set to 0.1 then the scale is 10:1 (in words ten to one).
72      -- This means: A distance of 10mm in the model represents
73      -- a distance of 1mm in the real world.
74      -- The drawing shows the reality enlarged.
75
76
77      package pac_scale_io is new ada.text_io.float_io (type_scale);
78
79
80      -- Converts the given scale factor to a string like 1:100
81      -- or 100:1:
82      function to_string (
83          scale : in type_scale)
84          return string;
85
86
87      -- Converts a distance of the model to a distance
88      -- in reality:
89      function to_reality (
90          d : in type_distance_model)
91          return type_distance_model;
92
93      procedure to_reality (
94          d : in out type_distance_model);
95
96
97
98      -- Converts a distance of the reality to
99      -- a distance in the model:
100     function to_model (
101         d : in type_distance_model)
102         return type_distance_model;
103
104     procedure to_model (
105         d : in out type_distance_model);
106
107
108
109     -- Converts a vector from model to reality:
110     function to_reality (
111         v : in type_vector_model)
112         return type_vector_model;
113
114
115     -- Converts a vector from reality to model:
116     function to_model (
117         v : in type_vector_model)
118         return type_vector_model;
119
120 end demo_scale;
```

9.21.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                                SCALE                                     --
6  --                                                                          --
7  --                                B o d y                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with ada.strings.bounded;
40 with ada.strings;
41 with ada.strings.fixed;
42
43
44 package body demo_scale is
45
46     function to_string (
47         scale : in type_scale)
48         return string
49     is
50         use pac_scale_io;
51         use ada.strings.bounded;
52         use ada.strings;
53         use ada.strings.fixed;
54
55         package pac_scale_bounded is new generic_bounded_length (10);
56         use pac_scale_bounded;
57
58         m_bounded : pac_scale_bounded.bounded_string;
59
60         -- This string holds temporarily the given scale.
61         -- The length of the string should be set in advance
62         -- here in order to take the longest possible combination
63         -- of charecters according to the declaration of type_scale.
64         -- Mind, the comma/point. It must also taken into account here:
65         m_fixed : string (1 .. type_scale'digits + 3);
66         -- CS find something more elegantly here.
67
68         m_reciprocal : type_scale;
69     begin
70         --put_line ("scale" & type_scale'image (scale));
71
72         -- Since we want an output like 1:100 or 100:1 the given scale
73         -- must be checked whether it is greater or less than 1.0:
74         if scale >= 1.0 then
```

```
75
76              -- Output the given scale to a fixed string
77              -- without an exponent:
78              put (
79                  to      => m_fixed, -- like 100.00
80                  item    => scale,
81                  exp     => 0); -- no exponent
82
83              -- Trim the string on both ends and store it in m_bounded:
84              m_bounded := trim (to_bounded_string (m_fixed), both);
85              -- CS remove leading zeroes after the comma.
86
87              -- Return a nicely formatted expression like 1:100
88              return "1:" & to_string (m_bounded);
89          else
90              -- The scale is smaller than 1.0. So we first
91              -- calculate the reciprocal of scale.
92              -- For example: scale 0.01 turns to 100.0:
93              m_reciprocal := 1.0 / scale;
94
95              -- Output the given scale to a fixed string
96              -- without an exponent:
97              put (
98                  to      => m_fixed, -- like 100.0
99                  item    => m_reciprocal,
100                 exp     => 0); -- no exponent
101
102             -- Trim the string on both ends and store it in m_bounded:
103             m_bounded := trim (to_bounded_string (m_fixed), both);
104             -- CS remove leading zeroes after the comma.
105
106             -- Return a nicely formatted expression like 100:1
107             return to_string (m_bounded) & ":1";
108         end if;
109     end to_string;
110
111
112     function to_reality (
113         d : in type_distance_model)
114         return type_distance_model
115     is begin
116         return type_distance_model_positive (M) * d;
117     end to_reality;
118
119
120     procedure to_reality (
121         d : in out type_distance_model)
122     is begin
123         d := type_distance_model_positive (M) * d;
124     end to_reality;
125
126
127
128     function to_model (
129         d : in type_distance_model)
130         return type_distance_model
131     is begin
132         return type_distance_model_positive (1.0 / M) * d;
133     end to_model;
134
135     procedure to_model (
136         d : in out type_distance_model)
137     is begin
138         d := type_distance_model_positive (1.0 / M) * d;
139     end to_model;
140
141
142
143     function to_reality (
144         v : in type_vector_model)
145         return type_vector_model
146     is begin
147         return (x => to_reality (v.x), y => to_reality (v.y));
148     end to_reality;
149
150
```

```
151     function to_model (
152         v : in type_vector_model)
153         return type_vector_model
154     is begin
155         return (x => to_model (v.x), y => to_model (v.y));
156     end to_model;
157 
158 
159 end demo_scale;
```

9.22 Das Paket demo_translate_offset

Dieses kleine Paket enthält nur eine Spezifikation. Es gibt also keine Datei für den Körper. Es enthält die globale Variable für den Translation-Offset T.

9.22.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                                DEMO CANVAS                               --
4  --                                                                          --
5  --                              TRANSLATE OFFSET                            --
6  --                                                                          --
7  --                                 S p e c                                  --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --    For correct displaying set tab width in your editor to 4.
27
28 --    The two letters "CS" indicate a "construction site" where things are not
29 --    finished yet or intended for the future.
30
31 --    Please send your questions and comments to:
32 --
33 --    info@blunk-electronic.de
34 --    or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --    history of changes:
37 --
38
39 with demo_logical_pixels;           use demo_logical_pixels;
40
41
42 package demo_translate_offset is
43
44     -- The global translate-offset by which all draw operations on the canvas
45     -- are translated when the operator zooms on the pointer or the cursor:
46     T : type_logical_pixels_vector := (0.0, 0.0);
47
48 end demo_translate_offset;
```

9.23 Das Paket demo_visibility

Zur Festlegung der Sichtbarkeitsschwelle und zur Größenprüfung von Flächen ist dieses Paket vorgesehen.

9.23.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                              VISIBILITY                                  --
6  --                                                                          --
7  --                               S p e c                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with demo_logical_pixels;       use demo_logical_pixels;
40 with demo_geometry;             use demo_geometry;
41
42
43 package demo_visibility is
44
45     -- If an object occupies a space that is wider or
46     -- higher than this constant, then it will be drawn on the screen:
47     visibility_threshold : constant type_logical_pixels_positive := 5.0;
48
49
50     -- Returns true if the given area is large enough
51     -- to display objects therein:
52     function above_visibility_threshold (
53         a : in type_area)
54         return boolean;
55
56 end demo_visibility;
```

9.23.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                              VISIBILITY                                  --
6  --                                                                          --
7  --                               B o d y                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 with glib;                          use glib;
40
41 with demo_logical_pixels;           use demo_logical_pixels;
42 with demo_geometry;                 use demo_geometry;
43 with demo_conversions;              use demo_conversions;
44
45
46 package body demo_visibility is
47
48
49     function above_visibility_threshold (
50         a : in type_area)
51         return boolean
52     is
53         -- CS: Optimization required. Compiler options ?
54         w : constant type_logical_pixels := to_distance (a.width);
55         h : constant type_logical_pixels := to_distance (a.height);
56         l : type_logical_pixels;
57     begin
58         -- Get the greatest of w and h:
59         l := type_logical_pixels'max (w, h);
60
61         if l > visibility_threshold then
62             return true;
63         else
64             return false;
65         end if;
66
67     end above_visibility_threshold;
68
69
70 end demo_visibility;
```

9.24 Das Paket window_dimensions

Von Haus aus verwendet *GTK3* für die Abmessungen von Fenstern den Zahlentyp `gint`. Die Abmessungen von Fenstern werden hier neu definiert, indem natürliche Zahlen verwendet werden.

9.24.1 Spezifikation

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                           WINDOW DIMENSIONS                              --
6  --                                                                          --
7  --                               S p e c                                    --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --    For correct displaying set tab width in your editor to 4.
27
28 --    The two letters "CS" indicate a "construction site" where things are not
29 --    finished yet or intended for the future.
30
31 --    Please send your questions and comments to:
32 --
33 --    info@blunk-electronic.de
34 --    or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --    history of changes:
37 --
38
39 -- with glib;
40
41 package demo_window_dimensions is
42
43     -- CS derive a type from gint like
44     -- type type_device_pixels is new glib.gint;
45     -- subtype type_device_pixels_positive is type_device_pixels
46     --  range 0 .. 1_000_000; -- CS use a reasonable limit
47
48
49     type type_window_size is record
50         width, height : positive := 1; -- CS subtype for reasonable limits
51     end record;
52
53
54     function to_string (
55         size : in type_window_size)
56         return string;
57
58
59 end demo_window_dimensions;
```

9.24.2 Körper

```
1  ------------------------------------------------------------------------------
2  --                                                                          --
3  --                              DEMO CANVAS                                 --
4  --                                                                          --
5  --                           WINDOW DIMENSIONS                              --
6  --                                                                          --
7  --                                B o d y                                   --
8  --                                                                          --
9  -- Copyright (C) 2024                                                       --
10 -- Mario Blunk / Blunk electronic                                           --
11 -- Buchfinkenweg 3 / 99097 Erfurt / Germany                                 --
12 --                                                                          --
13 -- This library is free software;  you can redistribute it and/or modify it --
14 -- under terms of the  GNU General Public License  as published by the Free --
15 -- Software  Foundation;  either version 3,  or (at your  option) any later --
16 -- version. This library is distributed in the hope that it will be useful, --
17 -- but WITHOUT ANY WARRANTY;  without even the implied warranty of MERCHAN- --
18 -- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                            --
19 --                                                                          --
20 -- You should have received a copy of the GNU General Public License and    --
21 -- a copy of the GCC Runtime Library Exception along with this program;     --
22 -- see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see    --
23 -- <http://www.gnu.org/licenses/>.                                          --
24 ------------------------------------------------------------------------------
25
26 --   For correct displaying set tab width in your editor to 4.
27
28 --   The two letters "CS" indicate a "construction site" where things are not
29 --   finished yet or intended for the future.
30
31 --   Please send your questions and comments to:
32 --
33 --   info@blunk-electronic.de
34 --   or visit <http://www.blunk-electronic.de> for more contact data
35 --
36 --   history of changes:
37 --
38
39 package body demo_window_dimensions is
40
41
42     function to_string (
43         size : in type_window_size)
44         return string
45     is begin
46         return "w/h " & positive'image (size.width)
47             & "/" & positive'image (size.height);
48     end to_string;
49
50
51 end demo_window_dimensions;
```

Kapitel 10

Kompilieren und Starten des Demoprogramms

Das Demoprogramm wurde unter *OpenSuse* Linux entwickelt. Es wurde auf *OpenSuse* Linux und *Ubuntu* Linux getestet. Der Anwender hat zwei Möglichkeiten, das Programm zu testen:

1. Herunterladen der fertigen direkt ausführbaren Binärdatei
2. Herunterladen des Quellcodes und Kompilieren. Quelle: `https://github.com/Blunk-electronic/ada_training/tree/master/src/gtk/canvas`

10.1 Herunterladen der Binärdatei

Wer sich mit dem Kompilieren nicht aufhalten möchte, kann die fertige Binärdatei ausprobieren. Sie entspricht dem aktuellen Stand des Quellcodes. Die gepackte Binärdatei ist zum Herunterladen hier verfügbar:

`http://www.blunk-electronic.de/eda/demo.zip`

Beachte: Die Binärdatei ist für eine festgelegte Betriebsart (Modus 3) der Ansicht, einen festen Rasterabstand und eine festgelegte Sammlung darzustellender Objekte (Dreieck, Rechteck und Kreis) kompiliert. Wer daran etwas ändern will, muß den Quellcode entsprechend ändern und kompilieren.

10.1.1 Systemvoraussetzungen

Die Binärdatei ist so kompiliert, daß sie fast ohne externe Systembibliotheken[1]. ausführbar ist. Einzig die Standardbibliothek `GLIBC` muß in einer gewissen Aktualität vorhanden sein. `GLIBC` enthält für Linux-Systeme wichtige Kernfunktionen und ist in jeder Distribution enthalten. Um zu erfahren, ob diese Bibliothek vorhanden ist, eignet sich dieses Kommando im Terminal:

```
user@machine1:~> ldd --version
ldd (GNU libc) 2.39 ...
```

Ausführliche Informationen zu Dateinamen und Verknüpfungen erhält man auf diese Weise:

```
user@machine1:~> ldd `which ls` | grep libc
libcap.so.2 => /lib64/libcap.so.2 (0x00007f95ad250000)
libc.so.6 => /lib64/libc.so.6 (0x00007f95ad000000)
```

[1] Unter Linux enden diese Dateien auf `.so`. Windows® verwendet die Endung `.dll`.

Unter *OpenSuse* ist diese Bibliothek im Paket `glibc-devel-static` enthalten und kann einfach installiert werden.

Somit kann die Binärdatei auf jedem anderen Linux-System wie z.B. *Ubuntu* gestartet werden.

10.2 Kompilieren des Quellcodes

Um die Auswirkungen eigener Änderungen, Verbesserungen oder Erweiterungen zu testen oder um auch auf anderen Betriebssystemen zu experimentieren, muß kompiliert werden.

10.2.1 Systemvoraussetzungen

1. das Versionskontroll-System *Git*[8],[9]
2. der Ada-Kompiler `gnatmake`
3. die Werkzeugsammlung *GTK3*[5]
4. Das Kompilierwerkzeug `GPRbuild`[6]
5. Die GTK3-Bindungen an Ada `GtkAda`[6]

Gängige Linux-Distributionen wie *OpenSuse* oder *Ubuntu* behinhalten *Git*, `gnatmake` und *GTK3*. Etwas schwierig wird es mit der Installation von `GPRbuild` und `GtkAda` sodaß der Anwender hier leider etwas Eigenarbeit leisten muß. Eine Anleitung für die Installation von `gprbuild` und `GtkAda` unter *OpenSuse* ist hier zu finden:

`https://github.com/Blunk-electronic/ada_training/blob/master/gtkada-installation.md`

Zusätzlich ist dort auch ein entsprechendes Installationsskript verfügbar, welches die Installation deutlich vereinfacht.

Darüber hinaus sei auf die Anleitungen und Quellen von *AdaCore*® verwiesen:

- `https://github.com/AdaCore/gtkada`
- `https://github.com/AdaCore/gprbuild`

10.2.2 Kompilieren

Es ist ratsam, ein temporäres Verzeichnis anzulegen, in welchem der Quellcode abgelegt und kompiliert werden soll:

```
user@machine1:~> mkdir tmp
```

In dieses Verzeichnis wird sodann gewechselt:

```
user@machine1:~> cd tmp
```

Dann wird mittels des Befehls `git clone` der Quellcode in das gegenwärtige Arbeitsverzeichnis kopiert:

```
user@machine1:~/tmp> git clone https://github.com/Blunk-electronic/ada_training
Cloning into 'ada_training'...
remote: Enumerating objects: 5996, done.
remote: Counting objects: 100% (819/819), done.
remote: Compressing objects: 100% (361/361), done.
remote: Total 5996 (delta 680), reused 596 (delta 458), pack-reused 5177
Receiving objects: 100% (5996/5996), 2.30 MiB | 758.00 KiB/s, done.
Resolving deltas: 100% (4352/4352), done.
```

Terminal 10.1: Herunterladen des Quellcodes

Das heruntergeladene Paket enthält noch zahlreiche andere Demoprogramme, die uns aber nicht interessieren. Wir wechseln direkt in das Verzeichnis, in welchem sich der Quellcode für unser Demoprogramm sich befindet.

```
user@machine1:~/tmp> cd ada_training/src/gtk/canvas/
```

Nun wird mit `gprbuild` der Ada-Kompiler und der Linker gestartet:

```
user@machine1:~/tmp/ada_training/src/gtk/canvas> gprbuild
using project file demo.gpr
Compile
   [Ada]          demo.adb
   [Ada]          demo_base_offset.adb
   [Ada]          demo_bounding_box.adb
   [Ada]          demo_callbacks.adb
   [Ada]          demo_canvas.adb
   [Ada]          demo_coordinates_display.adb
   [Ada]          demo_frame.adb
   [Ada]          demo_grid.adb
   [Ada]          demo_main_window.adb
   [Ada]          demo_objects.adb
   [Ada]          demo_scrolled_window.adb
   [Ada]          demo_visible_area.adb
   [Ada]          demo_zoom.adb
   [Ada]          demo_logical_pixels.adb
   [Ada]          demo_geometry.adb
   [Ada]          demo_conversions.adb
   [Ada]          demo_cursor.adb
   [Ada]          demo_drawing_origin.adb
   [Ada]          demo_scale.adb
   [Ada]          demo_translate_offset.ads
   [Ada]          demo_buttons.adb
   [Ada]          demo_window_dimensions.adb
   [Ada]          demo_primitive_draw_ops.adb
   [Ada]          demo_visibility.adb
Bind
   [gprbind]      demo.bexch
   [Ada]          demo.ali
Link
   [link]         demo.adb
```

Das Ergebnis ist die ausführbare Datei `demo`:

```
user@machine1:~/tmp/ada_training/src/gtk/canvas> ls demo
demo
```

10.3 Starten der Binärdatei

Nach dem Herunterladen oder Kompilieren erhält der Anwender die ausführbare Binärdatei `demo`. Diese kann direkt aus der graphischen Oberfläche per Mausklick gestartet werden oder aber bevorzugt aus dem Terminal. Im Terminal sind Statusmeldungen, Fehlermeldungen und sonstige Nachrichten zu sehen, die das Demoprogramm ausgibt.

Meistens wird beim Herunterladen im Verzeichnis `Downloads` gespeichert. Wenn dort entpackt wurde und aus dem Terminal gestartet werden soll, dann wird die Binärdatei auf diese Weise ausgeführt:

```
user@machine1:~/Downloads> ./demo
```

Wenn nach obiger Anleitung (Abschnitt 10.2) kompiliert wurde, wird die Binärdatei hier gestartet:

```
user@machine1:~/tmp/ada_training/src/gtk/canvas> ./demo
```

Unmittelbar nach dem Starten, werden Statusmeldungen auf dem Terminal ausgegeben. Die folgende Auflistung zeigt ein Beispiel. In Zeile 1 erfolgte der Start durch den Bediener. Alle nachfolgenden Zeilen sind Nachrichten zur Laufzeit des Demoprogrammes:

```
 1 user@machine1:~/Downloads> ./demo
 2 make_drawing_frame
 3 frame position:x/y: -150.00/-105.00
 4 make_database_1
 5 scale_objects
 6 M: 1:1.00
 7 compute_canvas_size
 8 S_max :  1.00000000000000E+02
 9 Bw_max:  2.00000000000000E+03
10 Bh_max:  1.00000000000000E+03
11 F_max :  1.98000000000000E+05/-1.00000000000000E+05
12 Cw    :  398000
13 Ch    :  199000
14 compute_bounding_box
15 bounding-box: (x/y/w/h): -150.50/-105.50/ 298.00/ 211.00
16 has changed
17 set_up_main_window
18 create_window
19 set_up_command_buttons
20 create_buttons
21 set_up_swin_and_scrollbars
22 create_scrolled_window
23 scrolled window zoom mode: MODE_3_ZOOM_FIT
24 set_up_canvas
25 canvas size allocated (w/h): 1 / 1
26 canvas size minimum   (w/h): 398000 / 199000
27 add canvas to scrolled window
28 add scrolled window to box_h
29 show all widgets
30 set initial scrollbar settings
31 zoom_to_fit
32 start gtk main loop
33 cb_terminate
```

Terminal 10.2: Diagnose und Nachrichten

Kapitel 11

Sonstiges

11.1 Warum ausgerechnet Ada ?

An dieser Stelle möchte ich betonen, daß ich niemanden zur Verwendung von *Ada* bekehren will. Der Quellcode verwendet keine Ada-spezifischen Dinge, wie z.B. parametrierte oder gesteuerte Datentypen. Es sollte also jedem einigermaßen erfahrenen C/C++ oder Python-Programmierer möglich sein, den Code in seine Lieblingsprogrammiersprache zu portieren.

Mein Grundgedanke bei der Entscheidung für *Ada* war dieser: Wenn *Ada* für sicherheitsrelevante Anwendungen in der Medizin, Transport, Luftfahrt, Raumfahrt und im Militär verwendet wird, dann ist *Ada* für ein CAD-System ganz bestimmt geeignet. Bisher hat sich diese Entscheidung, nach 10 Jahren Programmierung in *Ada*, als richtig erwiesen.

Ada ist zertifiziert nach ISO/IEC 8652:1995(E) und 8652/2012(E).
Siehe `https://www.ada2012.org/`.

Eine große Sammlung von kleinen und großen Beispieldateien zum Erlernen der Programmiersprache *Ada* ist in [7] zu finden.

11.2 Warum GTK3 ?

Die Entscheidung für *GTK3* fiel einfach, weil es bereits die entsprechenden Bindungen[1] zwischen *Ada* und *GTK3* gibt.

Die Firma *AdaCore*®pflegt das quelloffene Paket *GtkAda*[6]. Siehe Kapitel 10 für Hinweise zur Installation.

11.3 Warum Versionskontrolle ?

Softwareentwicklung ist ohne Versionskontrolle kaum denkbar. Eine wichtige Voraussetzung für Versionskontrolle ist, daß Quellcode in Klartext vorliegt, was naturgemäß immer der Fall ist.

Entscheidende Gründe für die Anwendung von Versionskontrolle sind:

- Rückverfolgung von Fehlern im Quellcode
- Verfolgung von Änderungen
- Versionierung von Quellcode

Für den Einstieg in die Thematik sei auf [8] verwiesen. Ausführliche Literatur ist in [9] zu finden.

[1] engl. bindings

Literaturverzeichnis

[1] Nell Dale, Chip Weems, John McCormick. *Programming and Problem Solving with Ada 95*. Jones and Bartlett Publishers, Inc.

[2] John W. McCormick, Peter C. Chapin. *Building High Integrity Applications with SPARK*. Cambridge University Press

[3] John Barnes. *Programming in Ada 2005*. Addison Wesley.

[4] Andrew Krause. *Foundations of GTK+ Development*. Apress.

[5] GTK3. *The OpenSource Graphical Toolkit 3*. `https://www.gtk.org`

[6] AdaCore. *For portable, efficient GUI-driven applications in Ada*. `https://www.adacore.com/gtkada`

[7] Mario Blunk. *A collection of examples to learn programming in Ada*. `https://github.com/Blunk-electronic/ada_training`

[8] Mario Blunk. *Einführung in Versionskontrolle mit Git*. `http://www.blunk-electronic.de/pdf/git_einfuehrung.pdf`

[9] *A free and open source distributed version control system*. `https://git-scm.com/`

[10] *The Industry's Foundation for High Performance Graphics*. `http://www.opengl.org/`