Contents

1 Motivation

2 The concept of X-Windows

3 Concepts of Xgraphics

4 Xgraphics – Functions

4.1 Intialization and adminstrations

4.1.1 Initialize X and exit

`void InitX()`
Connection to the X-Server is established, X data structures like Graphics context and others are created and initialized with default values. Setting the font and a data structure within Xgraphics for administration is initialized.

`void ExitX()`
All remainig windows are closed, memory allocated by X and Xgraphics is freed, close connection to X-Server.

4.1.2 Handling of windows

`void Window CreateWindow( int width, int height, char* name )`
This function creates a window of the given size `width`×`height` in pixel and with name `name` (shown on top of the window). The window is not yet shown but must be made visible explicitly via `ShowWindow`. The function returns a data structure that describes the window, which must be used for every drawing activity.

`void DestroyWindow( Window win)`
The window `win` is closed. All drawing regions in this window are removed and their allocated memory is freed.

`void ShowWindow( Window win)`
The window `win` is shown on screen.

`void HideWindow( Window win)`
The window `win` is removed from screen, but the data structures are kept.

`void ClearWindow( Window win)`
The content of window `win` is removed.

4.1.3 Handling of drawing areas

```
World CreateWorld (
  Window win,
  int px, int py,
  int pwidth, int pheight,
  float wx1, float wy1,
  float wx2, float wy2,
  int scalable,
  int gravity
  )
```

The function `CreateWorld` creates a drawing area in the window `win`. The left, upper corner is at coordinates `(px,py)` relative to the window and the drawing area has the size `pwidth`×`pheight` in pixel. The "effective" coordinates of the upper left and lower right corners of the *drawing region* are `(wx1,wy1)` and `(wx2,wy2)`. The variable `scalable` determines the behavior of the drawin area when the corresponding window is resized. The following *constants* are defined:

0	The drawing area does not change its size.
`SCALABLE`	The drawing region can change its size.
`FREE_ASPECT`	The aspect ratio (height:width) can be changed. These constants can be com-
`FIXED_WIDTH`	The width stays fixed, implies `FREE_ASPECT`.
`FIXED_HEIGHT`	As `FIXED_WIDTH`, the height is fixed.

bined with logical or `|`. Examples:

- 0
  The drawing cannot change its size. If the window is enlarged, the drawing area moves according to the value of `gravity`. Such non-scalable drawing areas are the simplest ones. In this case *Xgraphics* can recover the content after an expose-event by itself, no `Redraw()` instruction is necessary.
- `SCALABLE` The size of the drawing area is fitted to the window. The asprect ratio is kept.
- `SCALABLE | FREE_ASPECT` The size of the drawing area is fitted to the size of the window, the margin size (relative to the window) is not changed.

For the variable `gravity` the following values are possible: `NorthGravity`, `NorthWestGravity`, `WestGravity`, ... The given cardinal point defines the corner or edge of the window with which the drawing area moves if the window is resized. So, things that are close to the lower border of the window should move with the lower border for not getting into conflict with upper laying, scalable areas.

`void DestroyWorld(World world)`
Removes the drawing area `world` from the memory. This instruction is necessary if one wants to overwrite a region in the window which is already occupied by another drawing area. If the previous one is not removed via `DestroyWorld()`, it can happen that the region at an expose-event is still erased and possibly after the content of the new drawing area has been restored.

`void ClearWorld(World world)`
The drawing area `world` is cleared.

4.1.4 Buttons

```
void InitButtons (
  Window win,
  const char* buttonstring,
  int width
  )
```

`InitButtons` creates at the right margin of the window `win` a column of buttons, which can be clicked with the mouse and then cause a `keypress` event. The number, function, and title of the buttons is set by the string `buttonstring`. For each button three specifications are made, separated by commas.

- distinction button or text line:
  a button is indicated by a `b`, a text by `t`
- title of the button / text
- key symbol that shall be returned

E.g., the string
`"t,commands:,,b,exit,e,b,menu,m,b,clear,c"`
creates the following buttons:

In a program with multiple windows there can only be one set of buttons. This corresponds to the philosophy to have a similar program flow as for a PC program. If control is handed over to another subprogram with its own window in which new buttons are initialized, the buttons from the main window are removed for this time. When the subprogram is exited the buttons of the main program must be re-initialized.

4.2 Event handling

```
int GetEvent( XEvent* event, int waitflag )
```

`GetEvent` reads in the next event from the event queue of the program and writes the information in the event variable, whose pointer is handed over via `event`. Usually, it is not necessary (except for `event.type`) to know the inner structure of the event variable. But as this variable is used directly the programmer familiar with X can get more information about the event. The variable `waitflag` sets the modus of `GetEvent`:

- 0 : If there is no event, the functions returns 0, creates an event with ID 0 (to prevent errors if event handling is tried nonetheless) and gives control back to the calling routine
- 1 : If there is no event, `GetEvent` will wait until an event occurs. This mode is only recomended, if the program has no further calculations to do, but is waiting for user input. In this mode the program will not consume CPU time (if one uses a waiting loop instead the program needs a lot of calculation time).

If the event was detected, there should be in general a reaction on it. This can be best done by using a `switch` block. The different cases are selected on basis of the type of the event. For every event the type is indicated by `event.typ` (or `event->type`, if `event` is a pointer to an event variable). The most important events that can occur and on which a reaction should happen:

`Expose:` The window has become visible again for some reason The content of the window must drawn again. For drawing areas that are non-scalable, i.e. `scalable=0`, this is done by `GetEvent` internally. Only for scalable drawing areas the program must provide a routine to recover the them. This routine is then called for an Expose event

`KeyPress:` A key or a mouse button is pressed. The character code of the pressed key can be obtained by `ExtractChar`

`ButtonPress:` A mouse button is pressed. Information on this event, like the coordinates and the number of the mouse button can be obtained by `WGetMousePos`.

`char ExtractChar(XEvent event)`
returns the ASCII code of the pressed key in case of a `KeyPress` event.

`int WGetMousePos(World world, XEvent event, float* x, float* y )`
gives the coordinates of the `ButtonPress` event relative to the coordinates of the drawing area `world`. The function returns 0, if the mouse was not within the drawing area. Otherwise, the number of the mouse button that was pressed is returned.

`int GetNumber(Window win, int x, int y, float *value)`
can be used to read in a number from the keyboard. At the position `x, y` the number appears. `GetNumber` allows to use backspace for editting this number. The number may have a sign or a decimal point. The result is written to the variable that is passed by the pointer `*value`

`int WGetNumber(World world, float x, float y, float *value)`
Similar to `GetNumber`, but coordinates refer to the drawing area `world`.

4.3 Drawing commands

4.3.1 Colors and drawing functions

The color of an object to be drawn and the function used to draw it are combined in the variable `int c` as follows. The four lowest bits encode the color. The represent an entry in the internal list `mycolors`, that is created during initialization with `InitX()`. The bits 4-7 then encode the function that is used to for the representation. As the normal function `GXcopy` in X does not correspond to the value `0`, there is another flag in bit 8, which defines whether a function other than `GXcopy` should be used. For example:

- Normal representation:
  `c = 2`. E.g., a point is shown with the color `mycolors[2]`, i.e., the pixel is overwritten with this color.
- Representation with Xor: `c = 4096 + 256*GXor + 2`. A point is connected by the function Xor with the previous value of the corresponding pixel. For simplification, the representation with Xor uses the constant `Xor = 4096 + 356*GXor`. If the point had originally the color white, after the command the point will have the color `mycolors[2]`. If the point had another color before the result is in general not predictable.

4.3.2 Drawing in a window

`void ClearArea (Window win, int x, int y, int width, int height)`
clears the given area.

`void DrawPoint (Window win, int x, int y, int c)`
draws a point of color `c` in the Window `win`.

`void DrawPoints (Window win, XPoint *points, int NofPoints, int c)`
draws `NofPoints` Points. They are passed through the structure `XPoint *points`, an array of `struct { int x,y } XPoints`.

`void DrawLine (Window win, int x1, int y1, int x2, int y2, int c)`
draws a line from (`x1,y1`) to (`x2,y2`).

`void DrawLines (Window win, XPoint *points, int NofPoints, int c)`
draws an open polygon with corners defined by `points`.

`void DrawCircle (Window win, int x, int y, int r, int c)`
draws a circle with radius `r` and midpoint (`x,y`).

`void FillCircle (Window win, int x, int y, int r, int c)`
draws a filled circle.

`void DrawString (Window win, int x, int y, const char *text, int c`
draws text `text` at position (`x,y`).

`void DrawRectangle (Window win, int x1, int y1, int x2, int y2, int c`
draws a rectangle with left upper corner (`x1,y1`) and right lower corner (`x2,y2`).

`void FillRectangle (Window win, int x1, int y1, int x2, int y2, int c`
draws a filled rectangle.

`void DrawPoly (Window win, XPoint *points, int NofPoints, int c`
draws a closed polygon. The polygonal line given by `points` does not need to be closed.

`void FillPoly (Window win, XPoint *points, int NofPoints, int c, int cfill`
draws a polygon with color `c` and fills it with color `cfill`.

4.3.3 Drawing into drawing areas

The drawing functions for drawing areas have the same syntax as the drawing functions for windows. Instead of the window the drawing area is given. The coordinates are now floating point numbers.

```
void WDrawPoint (World world, float x, float y, int c)

void WDrawPoints (World world, WPoint *points, int NofPoints, int c)

void WDrawLine (World world, float x1, float y1, float x2, float y2, int c)

void WDrawLines (World world, WPoint *points, int NofPoints, int c)

void WDrawCircle (World world, float x, float y, float r, int c)

void WFillCircle (World world, float x, float y, float r, int c)

void WDrawString (World world, float x, float y, const char *text, int c

void WDrawRectangle (World world, float x1, float y1, float x2, float y2, int c

void WFillRectangle (World world, float x1, float y1, float x2, float y2, int c

void WDrawPoly (World world, WPoint *points, int NofPoints, int c

void WFillPoly (World world, WPoint *points, int NofPoints, int c, int cfill
```

5 Demo program

This chapter is intended to demonstrate the functionality of Xgraphics with the help of some examples for programs. Moreover, the examples should demonstrate "good" coding, e.g., giving suggestions for the setting of drawing areas, usage of colors, and for the treatment of waiting loops.

Let's start with a "minimum program", which only opens a window, draws some Lissajous figures in it, and closes the window after pressing the mouse button.

```
/*****************************************************************************
*                    Xgraphics demo program                                  *
*****************************************************************************/

#include <math.h>
#include "Xgraphics.h"

//Main program
int main () {

//Variables for window handling
 Window mywindow ;
 World myworld ;
 XEvent myevent ;

 int i, done = 0 ; // control variables

//1st: Connect to X-server
 InitX() ;

//Set window and drawing area
 mywindow = CreateWindow(
               400, 400, // width, height
               (char*)"demo1" // name of window
                        ) ;
 myworld  = CreateWorld(
               mywindow, // window to draw in
               5,5,390,390, // x,y,width,height
               -1,1,1,-1, // wx1,wy1,wx2,wy2
               0,        // scaling property
               0         // "gravity"
                       ) ;

//Presenting the window and drawing a Lissajous figure
 ShowWindow(mywindow) ;

 for(i=1;i<620;++i) WDrawCircle(myworld,cos(i),sin(2*i),0.1,1);

//Event handling: wait for mouse click
 while(!done) {
   GetEvent (&myevent,1) ;
   if (myevent.type == ButtonPress) done = 1 ;
 }
```

```
//Clean exit of program
 ExitX() ; // clean up memory
 return 0 ;
}
```

The following program demonstrates the usage of drawing areas with different scaling properties, the usage of buttons when branching into a subprogram with its own buttons, the read out of mouse pointer coordinates, and the treatment of an `Expose` event for scalable drawing areas.

```
/*****************************************************************************
*                     Xgraphics demo program                                 *
*****************************************************************************/

#include <math.h>
#include "Xgraphics.h"

void redraw (World &world1, World &world2, World &world3) ;
void submenu (Window &subwindow, Window &mywindow, World &world1, World &world2,
              World &world3, XEvent &myevent) ;

// Main program:

int main () {

  Window mywindow, subwindow ;
  World world1, world2, world3, world4, textworld ;
  XEvent myevent ;

  int done = 0 ; // control variable
  double x, y ;  // coordinates

  InitX() ;      // connect to X-server

  mywindow = CreateWindow (
                           600, 400, // width, height
                           (char *)"demo2" // name of the window
                          ) ;

  subwindow = CreateWindow (
                            300, 200,
                            (char *)"submenu"
                            ) ;

  /* This program contains five drawing areas. Three of them can
     be scaled. To prevent them from cutting through each other when scaling
     the window, one has to specify at which margin another scalable
     drawing area is attached. This is done via the FLOAT_xxx
     options. */

  world1 = CreateWorld ( mywindow,
                         5, 5, 245, 122,
                         0, 0, 1, 1,
                         FLOAT_SOUTH +
                         SCALABLE,
                         0
                       ) ;
```

```
world2 = CreateWorld ( mywindow,
                       5, 128, 122, 122,
                       0, 0, 1, 1,
                       FLOAT_NORTH +
                       FLOAT_EAST +
                       SCALABLE,
                       0
                     ) ;

world3 = CreateWorld ( mywindow,
                       128, 128, 122, 122,
                       0, 0, 10, 10,
                       FREE_ASPECT +
                       FLOAT_NORTH +
                       FLOAT_WEST +
                       SCALABLE,
                       0
                       ) ;

/* For the worlds that cannot be scaled, one has to specify, how they
  should move realtive to the window margins if the window is shrinked/enlarged.
  The corner or margin of the window to which the world should stay attached
  is specified via the gravity constant.*/


world4 = CreateWorld ( mywindow,
                       255, 5, 245, 245,
                       0, 0, 10, 10,
                       0,
                       NorthEastGravity
                     ) ;

textworld = CreateWorld ( mywindow,
                          5, 300, 490, 95,
                          0, 0, 489, 94,
                          0,
                          SouthWestGravity
                        ) ;

// definition of the buttons:
InitButtons ( mywindow,
              "t, commands:,,b,exit,e, b,menu,m, b,clear,c",
              100
            ) ;

// the window must be made visible and then one can draw into it
ShowWindow (mywindow) ;

// startup things
WDrawString (textworld, 10, 15,
     "Press any mouse button in the right field", 1) ;
WDrawString (textworld, 10, 30,
```

```
     "and resize the Window.", 1) ;

WDrawRectangle (world4, 0, 0, 10, 10, 1) ;

redraw (world1, world2, world3) ;
/* In redraw( ) the scalable drawing areas are used.
  The main function contains in most cases only the event loop, from
  there the program calls different functions depending on the event. */

// main loop
while (!done) {
  if ( GetEvent(&myevent,1) ) { // event handling

  /* In the GetEvent( ) function the wait_flag is set to 1,
     so the program waits for events without consuming CPU time. */

    switch (myevent.type) {

    /*Treatment of the KeyPress events, issued by a key or
       a mouse click in one of the fields defined by InitButtons( ). */

    case KeyPress:
      switch ( ExtractChar(myevent) ) {
      case 'e': done = 1 ; break ;
      case 'c': ClearWorld(world4) ;
        WDrawRectangle(world4, 0, 0, 10, 10, 1) ;
        break ;
      case 'm': submenu (subwindow, mywindow, world1, world2, world3, myevent) ;
        break ;
      default: break ;
      }
      break ;

    // Mouse actions: read out mouse pointer coordinates

    case ButtonPress:                          // mouse click
      WGetMousePos (world4, myevent, &x, &y) ; // get coordinates
      switch (myevent.xbutton.button) {
      case 1:
        WDrawCircle (world4, x, y, 0.2, 1) ; break ;
      case 2:
        WDrawCircle (world4, x, y, 0.4, 4096+256*GXor+1) ; break ;
      case 3:
        WFillCircle (world4, x, y, 0.2, 1) ; break ;
      default: break ;
      }
      break ;

    /* The window is redrawn. The non-scalable drawing areas are restored
      automatically. One has to take care by oneself for the other ones. It is
      recommended to define a function redraw( ) to do this, and which is called
      in the case of an Expose event. */
```

```
      case Expose: redraw (world1, world2, world3) ; break ;
      default: break ;
      }
    }
  }

  ExitX () ; //Cleans up everything, closes windows, and frees memory

  return 0 ;

}

/* ------------------------------------------------------------------- */
/* In redraw( ) all scalable worlds are drawn, moreover two different
   ways of selecting colors are demonstrated. */

void redraw (World &world1, World &world2, World &world3) {

  WPoint points[100] ;

  for (int i = 0 ; i < 100 ; ++i) {
    points[i].x = i / 100. ;
    points[i].y = ( sin( i * 2. * 0.031415) + 1. ) / 2. ;
  }
  WDrawRectangle (world2, 0, 0, 1, 1, 1) ;
  WDrawLines (world2, points, 100, 1) ;
  WDrawRectangle (world1, 0, 0, 1, 1, 1) ;

  for (int i = 1 ; i < 20 ; ++i)
    WDrawLine (world1, 0, float(i) / 20., 1, 1. - float(i) / 20.,
       i%(NofColors-1)+1) ;
 //These colors should be always different from the background color.

  WDrawRectangle (world3, 0, 0, 10, 10, 1) ;

  for (int i = 1 ; i <= 10 ; ++i)
    for ( int j = i ; j <= 10 ; ++j)
      WFillRectangle (world3, i-1, j-1, i, j, 2+(i+j)%2) ;
  //These colors should be different from each other.

  return ;
}
/* ------------------------------------------------------------------- */
/* Example of a subprogram with its own window and own menu. It might be easier
   to define this window and the drawing areas _globally_.

void submenu (Window &subwindow, Window &mywindow, World &world1, World &world2,
              World &world3, XEvent &myevent) {

  int done = 0 ;
  ShowWindow (subwindow) ;
```

```
  InitButtons (subwindow,
               "t,submenu:,,b,main,m",
               100) ;

  while (!done) {
    if ( GetEvent(&myevent, 1) ) { // event handling
      switch (myevent.type) {

      case KeyPress:
        switch ( ExtractChar(myevent) ) {
        case 'm':
          done = 1 ; break ;
        default: break ;
        }
        break ;
      case Expose:
        redraw(world1, world2, world3) ; break ;
      default: break ;
      }
    }
  }

  /* Before returning to the main program, one has to restore the
     buttons of the main program. It is reasonable to do this already in the
     subprogram, so that the main program main program is independent of any
     subprogram defining its own buttons. */

  InitButtons (mywindow,
       "t,commands:,,b,exit,e, b,menu,m, b,clear,c",
       100) ;
  HideWindow(subwindow);

  return ;

}
```