A short introduction to GUI programming with the MVC concept using wxWindows

M. Bernreuther

Summer term 2000

Abstract

The article starts with an introduction to the C++ library wxWindows. The first small example wxHello, is the well known "Hello world" type example. Another example wxKoch will be extended to show the implementation of a Model-View-Controller concept.

Keywords wxWindows, C++ , GUI programming, MVC concept

Contents

1 Introduction to wxWindows

1.1 What is wxWindows

wxWindows is a C++ library which allows software development for several platforms. It's main target is the programming of the Graphical User Interface (GUI). To get more information look at the homepage of wxWindows `http://www.wxwindows.org/`. The examples are written for the windows version using Microsoft's Visual C++ compiler. It should also compile on all other supported platforms, like Unix (GTK+ or Motif) or Apple's Macintosh. For more information on the GTK+ based wxWindows version (called wxGTK), which is often used with GNU/Linux, look at `http://www.freiburg.linux.de/~wxxt/`.

Currently[1] version 2.2.0 is available. For downloads look at `http://www.wxwindows.org/dl_msw2.htm`, the main ftp site `ftp://www.remstar.com/pub/wxwin/` or a mirror site. We will use the setup executable `ftp://ftp.mapsy.nat.tu-bs.de/pub/mirror/wxwin/2.2.0/wxMSW-2.2.0-setup.zip`. You may also get `wxWindows-2.2.0-WinHelp.zip` and add the files after the installation procedure.

There is also a quite helpful mailing list. Look at `http://www.wxwindows.org/maillst2.htm` for information.

1.2 Installation

After unzipping and doing the common installation procedure using setup.exe, the library has to be build. There are two ways:

1. using makefiles
2. using VC project and workspace files (`%wxwin%\src\wxvc.ds?`)

Help can generally be found at `%wxwin%\docs`, where `%wxwin%` is the wxWindows root directory. `%wxwin%\docs\msw\install.txt` contains information about the installation procedure.

Do "rebuild all" for the "debug" and "release" configuration of "wxvc". Additional to `wxvc.ds?`, which generates static libraries, there's `wxvc_dll.ds?` for a dynamic link library (dll) version.

To find out about the features of wxWindows, the samples are very helpful. There are found at `%wxwin%\samples`. To build them all, `SamplesVC.dsw` can be used.

[1] July 2000

2 The first programm

The first example is the famous "hello world" application. It can be found at `http://www.wxwindows.org/hworld2.txt`. A similar program is explained at `http://www.wxwindows.org/hello.htm`. You might also have a look at the `minimal` sample.

It's important that after creating a new project with VC++, the project settings are adapted to needs of wxWindows, e. g. the search path have to include the wxWindows include and library files. One possibility is to (mis)use an existing sample. To simplify the process a tool is developed here called wxProjectGenerator to generate the *.ds? files, which can be opened with VC++. Let's generate a project called wxHello. There is a resource file called "wxHello.rc" with only one line

```
#include "wx/msw/wx.rc"
```

which will be generated automatically.

The source code is found in "wxHello.cpp". In this first example "wxHello.h" does not exist and all declarations will be made in the source file. Let's go through it:

```
// wxHello.cpp
// Robert Roebling, Martin Bernreuther

#include <wx/wx.h>
```

First the wxWindows library has to be included. There is a class called wxApp defined there, which represents the application.

```
class wxHelloApp : public wxApp
{
    virtual bool OnInit();
};
```

Now the wxHelloApp has to be derived from the generic wxApp. At the startup of the application a method "OnInit" is called automatically. Therefore "OnInit" has to be overloaded to plug in the initialization stuff.

```
IMPLEMENT_APP(wxHelloApp)
```

is a macro and (alternativly) stands for

```
wxApp *wxCreateApp() { return new wxHelloApp; }
wxAppInitializer wxTheAppInitializer((wxAppInitializerFunction)
                                     wxCreateApp);
wxHelloApp& wxGetApp() { return *(wxHelloApp*)wxTheApp; }
```

`wxTheAppInitializer` and `wxCreateApp` are used by the library itself. The global function `wxGetApp()` can be used to get a pointer to the (one and only)

wxApp (respectively wxHelloApp)-Object. Now a window with a menu is needed. To get this, the application has to create a kind of wxFrame-Object. The specialized version wxHelloFrame is derived:

```
class wxHelloFrame : public wxFrame
{
public:
    wxHelloFrame(const wxString& title, const wxPoint& pos
                 , const wxSize& size);
    void OnQuit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);
};
```

The wxHello-Application should understand two commands:

1. "Quit" to exit and
2. "About" to display a message box with the program name

Therefore there will be two methods "OnQuit" and "OnAbout", which will be executed in case the menu item is selected. But this will take two steps:

1. If you select the menu item, an event is initiated
2. The event is tied to a method

Therefore two event numbers have to be defined:

```
enum
{
    ID_Quit=1,
    ID_About
};
```

Now the methods have to be implemented:

```
// Implementation----------------------------------------------

bool wxHelloApp::OnInit()
{
    wxHelloFrame *frame = new wxHelloFrame("Hello World"
                                , wxPoint(50,50), wxSize(450,350));

    // dynamic events-----------------------------------------
    frame->Connect( ID_Quit, wxEVT_COMMAND_MENU_SELECTED,
        (wxObjectEventFunction) &wxHelloFrame::OnQuit );
    frame->Connect( ID_About, wxEVT_COMMAND_MENU_SELECTED,
        (wxObjectEventFunction) &wxHelloFrame::OnAbout );
    //------------------------------------------------------

    frame->Show(true);
```

```
    SetTopWindow(frame);
    return true;
}
```

As already mentioned "wxHelloApp::OnInit" is the first method being executed. Here the frame window with title "Hello World" will be created at a certain position and size. The "Connect" method dynamically connects an event to a method. If the event occurs, the method is executed. But something has to raise an event... This can be done with a button or through a menu item. If the frame is created, its constructor is executed:

```
wxHelloFrame::wxHelloFrame(const wxString& title
                , const wxPoint& pos, const wxSize& size)
    : wxFrame((wxFrame*)NULL,-1,title,pos,size)
{
    // create menubar
    wxMenuBar *menuBar = new wxMenuBar;
    // create menu
    wxMenu *menuFile = new wxMenu;
    // append menu entries
    menuFile->Append(ID_About,"&About...");
    menuFile->AppendSeparator();
    menuFile->Append(ID_Quit,"E&xit");
    // append menu to menubar
    menuBar->Append(menuFile,"&File");
    // set frame menubar
    SetMenuBar(menuBar);

    // create frame statusbar
    CreateStatusBar();
    // set statusbar text
    SetStatusText("Demo for wxWindows");
}
```

Here the menubar is created and menus are appended. A menu can have several menu items, which are associated to an event number. If a menu item is selected, the specified event occurs. The event method will then be executed. The & char marks the key shortcut. Here Alt–F will open the File menu and x will afterwards exit the application. The status bar is found at the bottom of the frame window and the message "Demo for wxWindows" is written there.

```
void wxHelloFrame::OnQuit(wxCommandEvent& event)
{
    Close(true);
}

void wxHelloFrame::OnAbout(wxCommandEvent& event)
{
```

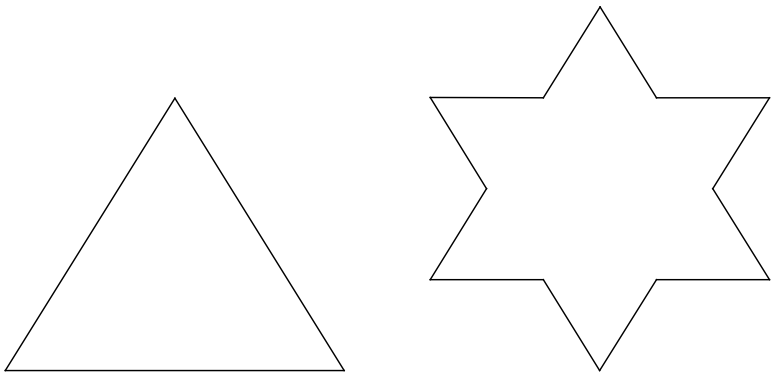
```
    wxMessageBox("wxWindows Hello World example."
                 ,"About Hello World", wxOK|wxICON_INFORMATION
                 , this);
}
```

"OnQuit" just closes the application and "OnAbout" shows a messagebox titled "About Hello World" with the text "wxWindows Hello World example.".

3 The Koch snowflake

3.1 What is a Koch snowflake?

The Koch snowflake is a nice recursive figure. It starts with a equal sided triangle. This triangle shown in figure 1(a) will be referred to as a Koch snowflake of level 0. Every edge is divided in three parts of equal length.



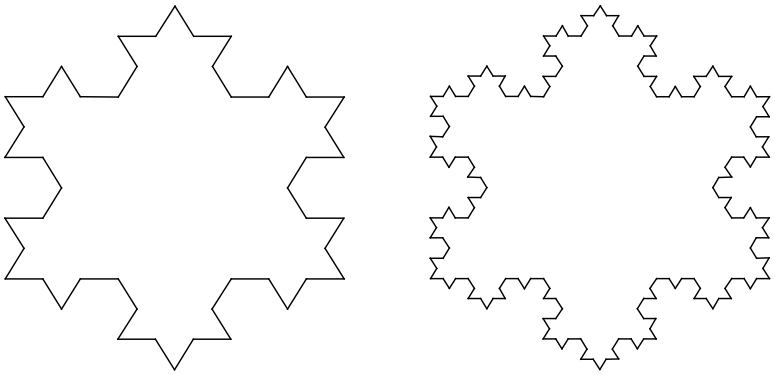
(a) level 0 (b) level 1

Figure 1: Koch snowflakes level 0 & 1

The middle one is replaced with the two other edges of another equal sided triangle. This Koch snowflake of level 1 is shown in figure 1(b). The procedure is repeated and Koch snowflakes of level n are obtained recursivly (see 2).

3.2 wxKoch0: The first approach

A program showing a Koch snowflake for a given level will be developed. In contrast to the program wxHello in section 2, a separate header file contains the declarations:



(a) level 2 (b) level 3

Figure 2: Koch snowflakes level 2 & 3

```
// wxKoch0.h
// Martin Bernreuther, Jan. 2000
// wxWindows 2
// draw a Koch snowflake recursivly
// wxKoch0: no model and no repaint

#ifndef WXKOCHH
#define WXKOCHH

#include <wx/wx.h>

class wxKochFrame : public wxFrame
{
public:
    wxKochFrame(const wxString& title, const wxPoint& pos
                , const wxSize& size);
    void OnQuit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);
    void OnInpLevel(wxCommandEvent& event);
};

class wxKochApp : public wxApp
{
    wxKochFrame *m_pframe;
    virtual bool OnInit();
public:
    void Draw(unsigned int n);
    void DrawEdge(wxDC& dc,unsigned int n,int x1,int y1
                                          ,int x2,int y2);
};
```

```
#endif
```

The wxKochApp stores the pointer to its frame m_pframe, which is created in the OnInit method. "Draw" will draw a Koch snowflake of level n and makes use of the DrawEdge method, which calls itself recursively.

Looking at the source file, the first part is similar to wxHello:

```
// wxKoch0.cpp
// Martin Bernreuther, Jan. 2000
// wxWindows 2

#include <wx/wx.h>
#include <math.h>

#include "wxKoch0.h"

IMPLEMENT_APP(wxKochApp)

enum
{
    ID_Quit=1,
    ID_About,
    ID_InpLevel
};

bool wxKochApp::OnInit()
{
    m_pframe = new wxKochFrame("Koch_Snowflake", wxPoint(50,50)
                               , wxSize(450,350));
    m_pframe->Show(true);

    SetTopWindow(m_pframe);
    return true;
}

wxKochFrame::wxKochFrame(const wxString& title, const wxPoint& pos
                         , const wxSize& size)
    : wxFrame((wxFrame*)NULL,-1,title,pos,size)
{
    SetIcon(wxICON(KochIcon));

    // create menubar
    wxMenuBar *menuBar = new wxMenuBar;
    // create menu
    wxMenu *menuFile = new wxMenu;
    wxMenu *menuHelp = new wxMenu;
    // append menu entries
    menuFile->Append(ID_InpLevel,"Input_&Level...\tCtrl-D");
```

```
    menuFile->AppendSeparator();
    menuFile->Append(ID_Quit,"E&xit");
    menuHelp->Append(ID_About,"&About...");
    // append menu to menubar
    menuBar->Append(menuFile,"&File");
    menuBar->Append(menuHelp,"&Help");
    // set frame menubar
    SetMenuBar(menuBar);

    // Connect event to callback function
    Connect( ID_Quit, wxEVT_COMMAND_MENU_SELECTED,
        (wxObjectEventFunction) &wxKochFrame::OnQuit );
    Connect( ID_About, wxEVT_COMMAND_MENU_SELECTED,
        (wxObjectEventFunction) &wxKochFrame::OnAbout );
    Connect( ID_InpLevel, wxEVT_COMMAND_MENU_SELECTED,
        (wxObjectEventFunction) &wxKochFrame::OnInpLevel );

    // create frame statusbar
    CreateStatusBar();
    // set statusbar text
    SetStatusText("Generating Koch snowflakes");
}

void wxKochFrame::OnQuit(wxCommandEvent& event)
{
    Close(true);
}

void wxKochFrame::OnAbout(wxCommandEvent& event)
{
    wxMessageBox("Generating Koch snowflakes\nby M. Bernreuther"
                 ,"About wxKoch", wxOK|wxICON_INFORMATION, this);
}
```

To get the level an input of the user is required. The method wxKochApp::OnInpLevel is executed through the menu.

```
void wxKochFrame::OnInpLevel(wxCommandEvent& event)
{
    // long Result=wxGetNumberFromUser(Descriptionline,Label
    //              ,Init,Min,Max,ParentWindow,Defaultposition)
    long level = wxGetNumberFromUser( "","Level:", "Input Level"
                                      , 4, 0, 10, this );
    wxString msg;
    if ( level == -1 )
        msg = "Invalid number entered or dialog cancelled.";
    else
    {
        msg.Printf("Level: %lu",level);
        wxGetApp().Draw(level);
```

```
    }
    SetStatusText(msg);
}
```

For the input of an positive integer wxWindows offers the dialog wxGetNumberFromUser, which also can be found in the Dialogs sample. If a valid input exists, the Koch snowflake is drawn.

```
void wxKochApp::Draw(unsigned int n)
{
    int width,height;
    unsigned int d,x1,y1,x2,y3,x3;

    // determine size of Window
    m_pframe->GetClientSize(&width, &height);
    d=height;
    // calculate coordinates of initial triangle
    if(width<height) d=width;
    y1=.5*height+.25*d;
    y3=.5*height-.5*d;
    x1=.5*width-.25*d*sqrt(3.);
    x2=.5*width+.25*d*sqrt(3.);
    x3=.5*width;

    // Initialize device context
    wxClientDC dc(m_pframe);
    dc.Clear();
    dc.SetPen(wxPen(wxColour(0,0,0), 1, wxSOLID));
    dc.BeginDrawing();
    // draw the edges
    DrawEdge(dc,n,x1,y1,x2,y1);
    DrawEdge(dc,n,x2,y1,x3,y3);
    DrawEdge(dc,n,x3,y3,x1,y1);
    dc.EndDrawing();
}

void wxKochApp::DrawEdge(wxDC& dc,unsigned int n,int x1,int y1
                                                 ,int x2,int y2)
{
    if(n>0)
    {
        // calculate new coordinates
        int x3,y3,x4,y4,x5,y5;
        x3=2.*x1/3.+x2/3.;
        y3=2.*y1/3.+y2/3.;
        DrawEdge(dc,n-1,x1,y1,x3,y3);
        x4=x1/3.+2.*x2/3.;
        y4=y1/3.+2.*y2/3.;
        x5=.5*(x1+x2)-(y2-y1)*sqrt(3.)/6.;
        y5=.5*(y1+y2)+(x2-x1)*sqrt(3.)/6.;
```

```
        // recursive calls
        DrawEdge(dc,n-1,x3,y3,x5,y5);
        DrawEdge(dc,n-1,x5,y5,x4,y4);
        DrawEdge(dc,n-1,x4,y4,x2,y2);
    }
    else
        // just draw a line
        dc.DrawLine(x1,y1,x2,y2);
}
```

The application will draw to the frame client window. The size of the window and the coordinates of the corners of the initial triangle are determined. Drawing operations need a device context. Here a wxClientDC is created and used. "Clear" first will result in an empty area and SetPen will specify the pen, which will be used for drawing. A black solid pen with width 1 is created here for this purpose. DrawEdge will receive the drawing context by reference and will do the recursion and drawing using DrawLine.

4 Model-View-Controller Concept

4.1 Overview

Following the Model-View-Controller (MVC) concept, an application consists of three parts (see 3), which communicate through messages.

The View and the Controller part are grown together in most applications. Important is the model part, which can be modelled as own class in C++. The properties of the model class represent the data used in the application. This can be drawing objects in a CAD like application or text and formatting information in a word processor. Since the MFC[2] was written for applications like MS-Word in the first place, a model is called "Document" there. wxWindows offers a MVC like environment, where the model also is called "document". This won't be used here. More information can be obtained looking at the help and the doc* samples.

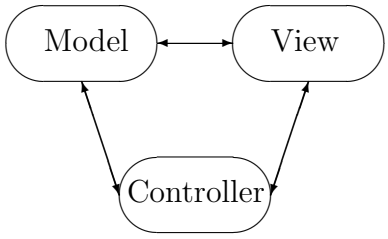


Figure 3: MVC concept

An application can have multiple instances of model data or just one at a time. If there are multiple[3] allowed, usually there is an active one, which

[2]Microsoft Foundation Classes: a commercial GUI/Class library
[3]this is often referred to as MDI (Multiple Document Interface)

receives the changes.

Regarding the views, the window manager of the Windows operating system doesn't store the areas overdrawn through other windows in the front. The application has to take care of restore the invalid areas itself. There is an repaint event which is release from the window manager to tell the view (through the application) to restore the area.

4.2 wxKoch1: Introducing a model and view class

Taking wxKoch0 (see 3.2), a model and a view class should be added like described in 4. The view class is a very specialized one and is able to draw Koch snowflakes. It might be better to have a general view and move the functionality to the model.

```
class KochView;

class KochModel
{
private:
    // level:
    unsigned int m_n;
    // the one and only view:
    KochView *m_pview;

public:
    KochModel();
    ~KochModel();

    void Setn(unsigned int n);
    unsigned int Getn() { return m_n; }

    void AddView(KochView *pview);
    void RemoveView(KochView *pview);

    void Draw();
};

class KochView : public wxWindow
{
private:
    // the model the view belongs to:
    KochModel *m_pmodel;
    wxPen m_pen;
    unsigned int m_x1,m_y1,m_x2,m_x3,m_y3;

    void CalcStartCoordinates(unsigned int width
                              ,unsigned int height
```

```
                                        ,int& x1,int& y1
                                        ,int& x2,int& x3,int& y3);

    void Draw(wxDC& dc,int x1,int y1,int x2,int x3,int y3);
    void DrawEdge(wxDC& dc,unsigned int n
                  ,int x1,int y1,int x2,int y2);

public:
    KochView(wxWindow *pparent, KochModel *pmodel);
    virtual ~KochView();
    void SetStatusText(wxString text)
        { ((wxFrame*)GetParent())->SetStatusText(text); }
    void Draw() { Refresh(); }
    void OnPaint(wxPaintEvent& event);
};
```

The application class stores a pointer to the actual model:

```
class KochApp : public wxApp
{
    KochFrame *m_pframe;
    KochModel *m_pmodel;
    virtual bool OnInit();
    virtual int OnExit();
public:
    KochFrame* GetpFrame() { return m_pframe; }
    KochModel* GetpModel() { return m_pmodel; }
    void SetStatusText(wxString text)
        { if(m_pframe) m_pframe->SetStatusText(text); }
};
```

which is created in KochApp::OnInit through:

```
    m_pmodel=new KochModel();
```

and deleted in KochApp::OnExit, which is called from wxWindows at the program termination:

```
int KochApp::OnExit()
{
    if(m_pmodel) delete m_pmodel;
    // application return code
    return 0;
}
```

The model creates its own view:

```
KochModel::KochModel()
{
    m_pview=NULL;
    m_n=4;
    new KochView(wxGetApp().GetpFrame(),this);
```

```
}

KochModel::~KochModel()
{
    if(m_pview)
        m_pview->Destroy();
}
```

Instead of deleting the view, the Destroy() or Close() method should be used instead. The KochView automatically registers and unregisters itself at KochModel in the constructor resp. destructor

```
KochView::KochView(wxWindow *pparent, KochModel *pmodel):
 wxWindow(pparent, -1), m_pmodel(pmodel)
{
    wxColour col(0,0,0);
    m_pen = wxPen(col, 1, wxSOLID);

    if(m_pmodel) m_pmodel->AddView(this);

    Connect( -1, wxEVT_PAINT
             , (wxObjectEventFunction) &KochView::OnPaint );
}

KochView::~KochView()
{
    if(m_pmodel) m_pmodel->RemoveView(this);
}
```

using KochModel::AddView resp. KochModel::RemoveView

```
void KochModel::AddView(KochView *pview)
{
    if(m_pview) delete m_pview;
    m_pview=pview;
}

void KochModel::RemoveView(KochView *pview)
{
    if(pview==m_pview) m_pview=NULL;
}
```

KochModel::Setn just sets a new level, which is the data the model has to store:

```
void KochModel::Setn(unsigned int n)
{
    if(n!=m_n)
    {
        m_n=n;
        if(m_pview)
```

```
        {
            wxString msg;
            msg.Printf("Level changed to %lu", m_n );
            wxGetApp().SetStatusText(msg);
            m_pview->Draw();
        }
    }
}
```

It is called from KochFrame::OnInpLevel. Since the view is responsible for the visualization, KochModel::Draw just calls the KochView::Draw method:

```
void KochModel::Draw()
{
    if(m_pview) m_pview->Draw();
}
```

The view receives the repaint event to update the drawing area through KochView::OnPaint. A pen is needed for the drawing procedure. The data comes from the model.

```
KochView::KochView(wxWindow *pparent, KochModel *pmodel):
 wxWindow(pparent, -1), m_pmodel(pmodel)
{
    wxColour col(0,0,0);
    m_pen = wxPen(col, 1, wxSOLID);

    if(m_pmodel) m_pmodel->SetView(this);

    Connect( -1, wxEVT_PAINT
            , (wxObjectEventFunction) &KochView::OnPaint );
}
```

The calculation of the first three vertices coordinates of the triangle is done in KochView::CalcStartCoordinates depending on the size of the view. The drawing is started with KochView::Draw and done in the recursive KochView::OnPaint:

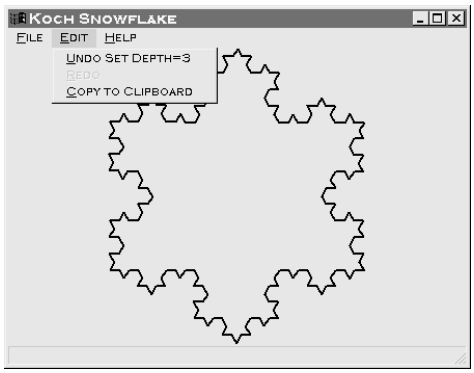
```
void KochView::CalcStartCoordinates(unsigned int width
                                    ,unsigned int height
                                    ,int& x1,int& y1
                                    ,int& x2,int& x3,int& y3)
{
    if(width<height)
    {
        y1=.5*height+.25*width;
        y3=.5*(height-width);
        x1=(.5-.25*sqrt(3.))*width;
        x2=(.5+.25*sqrt(3.))*width;
    }
    else
    {
```

```
            y1=.75*height;
            y3=0.;
            x1=.5*width-.25*height*sqrt(3.);
            x2=.5*width+.25*height*sqrt(3.);
        }
        x3=.5*width;
}

void KochView::Draw(wxDC& dc,int x1,int y1
                    ,int x2,int x3,int y3)
{
        if (!m_pmodel) return;

        int n=m_pmodel->Getn();

        DrawEdge(dc,n,x1,y1,x2,y1);
        DrawEdge(dc,n,x2,y1,x3,y3);
        DrawEdge(dc,n,x3,y3,x1,y1);
}

void KochView::DrawEdge(wxDC& dc,unsigned int n
                        ,int x1,int y1,int x2,int y2)
{
        if(n>0)
        {
            int x3,y3,x4,y4,x5,y5;
            x3=2.*x1/3.+x2/3.;
            y3=2.*y1/3.+y2/3.;
            DrawEdge(dc,n-1,x1,y1,x3,y3);
            x4=x1/3.+2.*x2/3.;
            y4=y1/3.+2.*y2/3.;
            x5=.5*(x1+x2)-(y2-y1)*sqrt(3.)/6.;
            y5=.5*(y1+y2)+(x2-x1)*sqrt(3.)/6.;
            DrawEdge(dc,n-1,x3,y3,x5,y5);
            DrawEdge(dc,n-1,x5,y5,x4,y4);
            DrawEdge(dc,n-1,x4,y4,x2,y2);
        }
        else
            dc.DrawLine(x1,y1,x2,y2);
}

void KochView::OnPaint(wxPaintEvent& event)
{
        if(m_pmodel)
        {
            int width,height;
            GetClientSize(&width, &height);
            int x1,y1,x2,x3,y3;
            CalcStartCoordinates(width,height,x1,y1,x2,x3,y3);
```

```
        wxPaintDC dc(this);
        dc.Clear();
        dc.SetPen(m_pen);
        Draw(dc,x1,y1,x2,x3,y3);
    }
}
```

KochView::OnPaint is called every time a repaint event is released.

5 wxKoch: The final version



5.1 Added features

5.1.1 Change line thickness and color

The line thickness is described with an integer number. The input is therefore analog to the input of the level. To choose a color, wxWindows offers a color dialog, which is used here. For common dialogs look at the "dialogs" sample.

5.1.2 Printing and Clipboard copy

wxWindows offers special device contexts for printing and the clipboard. Instead of drawing to the screen, KochView::Draw draws to the printer or in the clipboard.

5.1.3 Undo/Redo

Undo/Redo is implemented through classes for each command. The class CmdSetLevel for example sets and restores the level with its Do and Undo method. The command processor object manages all command objects.

5.2 The sources

5.2.1 wxKoch.h

```
1  // wxKoch.h
2  // Martin Bernreuther, July 2000
3  // using wxWindows 2.2.0
4  // draws & prints the recursive Koch snowflake structure
5
6  // wxKoch1 -> wxKoch: added the possibility of
7  //       setting Pen width & color
8  //       printing capabilities
9  //       the possibility of copying to the clipboard
10 //       Undo/Redo
11
12
13 #ifndef WXKOCHH
14 #define WXKOCHH
15
16 #include <wx/wx.h>
17 // define wxCommand:
18 #include <wx/docview.h>
19
20 class KochView;
21
22 class KochModel
23 {
24 private:
25     // level:
26     unsigned int m_n;
27     // the one and only view:
28     KochView *m_pview;
29
30 public:
31     KochModel();
32     ~KochModel();
33
34     void Setn(unsigned int n);
35     unsigned int Getn() { return m_n; }
36
37     void AddView(KochView *pview);
38     void RemoveView(KochView *pview);
39
40     void Draw();
41
42     unsigned int GetPenWidth();
43     wxColor GetPenColor();
44     void SetPenWidth(int w);
45     void SetPenColor(wxColor c);
```

```
46      void Print();
47      void Copy2Clipboard();
48  };
49
50  class KochView : public wxWindow
51  {
52  private:
53      // the model the view belongs to:
54      KochModel *m_pmodel;
55      wxPen m_pen;
56      wxFont m_font;
57      unsigned int m_x1,m_y1,m_x2,m_x3,m_y3;
58
59      void CalcStartCoordinates(unsigned int width
60                                ,unsigned int height
61                                ,int& x1,int& y1
62                                ,int& x2,int& x3,int& y3);
63
64      void Draw(wxDC& dc,int x1,int y1,int x2,int x3,int y3);
65      void DrawEdge(wxDC& dc,unsigned int n
66                    ,int x1,int y1,int x2,int y2);
67
68  public:
69      KochView(wxWindow *pparent, KochModel *pmodel);
70      virtual ~KochView();
71      void SetStatusText(wxString text)
72          { ((wxFrame*)GetParent())->SetStatusText(text); }
73      void Draw() { Refresh(); }
74      void OnPaint(wxPaintEvent& event);
75
76      unsigned int GetPenWidth();
77      wxColour GetPenColor();
78      void SetPenWidth(int w);
79      void SetPenColor(wxColor c);
80      void Print();
81      void Copy2Clipboard();
82  };
83
84
85  class KochFrame : public wxFrame
86  {
87  private:
88      wxCommandProcessor m_cmdprocessor;
89  public:
90      KochFrame(const wxString& title, const wxPoint& pos
91                                     , const wxSize& size);
92      void OnQuit(wxCommandEvent& event);
93      void OnAbout(wxCommandEvent& event);
94      void OnInpLevel(wxCommandEvent& event);
```

```
95
96      void OnSetPenWidth(wxCommandEvent& event);
97      void OnSetPenColor(wxCommandEvent& event);
98      void OnPrint(wxCommandEvent& event);
99      void OnCopyClipboard(wxCommandEvent& event);
100     void OnUndo(wxCommandEvent& event);
101     void OnRedo(wxCommandEvent& event);
102 };
103
104
105 class KochApp : public wxApp
106 {
107 private:
108     KochFrame *m_pframe;
109     KochModel *m_pmodel;
110     wxColourData m_coldata;
111     wxPrintDialogData m_prndata;
112     virtual bool OnInit();
113     virtual int OnExit();
114 public:
115     KochFrame* GetpFrame() { return m_pframe; }
116     KochModel* GetpModel() { return m_pmodel; }
117     void SetStatusText(wxString text)
118         { if(m_pframe) m_pframe->SetStatusText(text); }
119     wxColourData* GetpColdata() { return &m_coldata; }
120     wxPrintDialogData* GetpPrndata() { return &m_prndata; }
121 };
122
123 class CmdSetLevel : public wxCommand
124 {
125 private:
126     unsigned int m_level, m_oldlevel;
127 public:
128     CmdSetLevel(unsigned int d)
129         : wxCommand(TRUE,"Input Level"),m_level(d) {}
130     virtual bool Do();
131     virtual bool Undo();
132 };
133
134 class CmdSetPenWidth : public wxCommand
135 {
136 private:
137     unsigned int m_penwidth, m_oldpenwidth;
138 public:
139     CmdSetPenWidth(unsigned int w)
140         : wxCommand(TRUE,"Set Pen Width"),m_penwidth(w) {}
141     virtual bool Do();
142     virtual bool Undo();
143 };
```

```
144
145 class CmdSetPenColor : public wxCommand
146 {
147 private:
148     wxColourData m_coldata, m_oldcoldata;
149 public:
150     CmdSetPenColor(const wxColourData& c)
151         : wxCommand(TRUE,"Set Pen Color"),m_coldata(c) {}
152     virtual bool Do();
153     virtual bool Undo();
154 };
155
156 #endif
```

5.2.2 wxKoch.cpp

```
1  // wxKoch.cpp
2  // Martin Bernreuther, July 2000
3  // using wxWindows 2.2.0
4
5  #include <wx/wx.h>
6  #include <math.h>
7
8  #include <wx/colordlg.h>
9  #include <wx/printdlg.h>
10 #include <wx/metafile.h>
11 #include <wx/clipbrd.h>
12
13
14 #include "wxKoch.h"
15
16 IMPLEMENT_APP(KochApp)
17
18 enum
19 {
20     ID_Quit=1,
21     ID_About,
22     ID_InpLevel,
23
24     ID_SetPenWidth,
25     ID_SetPenColor,
26     ID_CopyClipboard,
27     ID_Print
28 };
29
30 //KochApp----------------------------------------------------------
31
32 bool KochApp::OnInit()
33 {
```

```
34      m_pframe = new KochFrame("Koch Snowflake", wxPoint(50,50)
35                               , wxSize(450,350));
36      m_pframe->Show(true);
37      SetTopWindow(m_pframe);
38      m_pmodel = new KochModel();
39
40      // initialize ColourDialogData
41      m_coldata.SetChooseFull(TRUE);
42      for (int i = 0; i < 16; i++)
43      {
44          wxColour col(i*16, i*16, i*16);
45          m_coldata.SetCustomColour(i, col);
46      }
47
48      // initialize PrintDialogData
49      m_prndata.EnableSelection(FALSE);
50      m_prndata.EnablePageNumbers(FALSE);
51
52      return true;
53  }
54
55
56  int KochApp::OnExit()
57  {
58      if(m_pmodel) delete m_pmodel;
59      // application return code
60      return 0;
61  }
62
63  //KochFrame-----------------------------------------------------------
64
65  KochFrame::KochFrame(const wxString& title
66                       , const wxPoint& pos, const wxSize& size)
67          : wxFrame((wxFrame*)NULL,-1,title,pos,size)
68  {
69      SetIcon(wxICON(KochIcon));
70
71      // create menubar
72      wxMenuBar *menuBar = new wxMenuBar;
73      // create menu
74      wxMenu *menuFile = new wxMenu;
75      wxMenu *menuEdit = new wxMenu;
76      wxMenu *menuHelp = new wxMenu;
77      // append menu entries
78      menuFile->Append(ID_InpLevel,"Input &Level...\tCtrl-D");
79      menuFile->Append(ID_SetPenWidth,"Set Pen &Width...");
80      menuFile->Append(ID_SetPenColor,"Set Pen &Color...");
81      menuFile->Append(ID_Print,"&Print...\tCtrl-P");
82      menuFile->AppendSeparator();
```

```
83      menuFile->Append(ID_Quit,"E&xit\tAlt-F4");
84      menuEdit->Append(wxID_UNDO,"&Undo");
85      menuEdit->Append(wxID_REDO,"&Redo");
86      menuEdit->Append(ID_CopyClipboard,"&Copy to Clipboard");
87      menuHelp->Append(ID_About,"&About...");
88      // append menu to menubar
89      menuBar->Append(menuFile,"&File");
90      menuBar->Append(menuEdit,"&Edit");
91      menuBar->Append(menuHelp,"&Help");
92      // add Undo/Redo entries to menuEdit & Initialize
93      m_cmdprocessor.SetEditMenu(menuEdit);
94      m_cmdprocessor.Initialize();
95      // set frame menubar
96      SetMenuBar(menuBar);
97
98      Connect( ID_Quit, wxEVT_COMMAND_MENU_SELECTED
99               ,(wxObjectEventFunction) &KochFrame::OnQuit );
100     Connect( ID_About, wxEVT_COMMAND_MENU_SELECTED
101              ,(wxObjectEventFunction) &KochFrame::OnAbout );
102     Connect( ID_InpLevel, wxEVT_COMMAND_MENU_SELECTED
103              ,(wxObjectEventFunction) &KochFrame::OnInpLevel );
104     Connect( ID_SetPenWidth, wxEVT_COMMAND_MENU_SELECTED
105              ,(wxObjectEventFunction) &KochFrame::OnSetPenWidth );
106     Connect( ID_SetPenColor, wxEVT_COMMAND_MENU_SELECTED
107              ,(wxObjectEventFunction) &KochFrame::OnSetPenColor );
108     Connect( ID_Print, wxEVT_COMMAND_MENU_SELECTED
109              ,(wxObjectEventFunction) &KochFrame::OnPrint );
110     Connect( ID_CopyClipboard, wxEVT_COMMAND_MENU_SELECTED
111              ,(wxObjectEventFunction) &KochFrame::OnCopyClipboard );
112     Connect( wxID_UNDO, wxEVT_COMMAND_MENU_SELECTED
113              ,(wxObjectEventFunction) &KochFrame::OnUndo );
114     Connect( wxID_REDO, wxEVT_COMMAND_MENU_SELECTED
115              ,(wxObjectEventFunction) &KochFrame::OnRedo );
116
117     // create frame statusbar
118     CreateStatusBar();
119     // set statusbar text
120     SetStatusText("Generating Koch snowflakes");
121 }
122
123 void KochFrame::OnQuit(wxCommandEvent& event)
124 {
125     Close(true);
126 }
127
128 void KochFrame::OnAbout(wxCommandEvent& event)
129 {
130     wxMessageBox("Generating Koch snowflakes\nby M. Bernreuther"
131                  ,"About Koch", wxOK|wxICON_INFORMATION, this);
```

```
132 }
133
134 void KochFrame::OnInpLevel(wxCommandEvent& event)
135 {
136     // long Result=wxGetNumberFromUser(Descriptionline,Label
137     //                 ,Init,Min,Max,ParentWindow,Defaultposition)
138     long res = wxGetNumberFromUser( "","Level:", "Input Level"
139                   , wxGetApp().GetpModel()->Getn() , 0, 10, this );
140     wxString msg;
141     if ( res == -1 )
142         msg = "Invalid number entered or dialog cancelled.";
143     else
144     {
145         msg.Printf("Level: %lu", res );
146         m_cmdprocessor.Submit(new CmdSetLevel(res));
147         wxGetApp().GetpModel()->Draw();
148     }
149     SetStatusText(msg);
150 }
151
152 void KochFrame::OnSetPenWidth(wxCommandEvent& event)
153 {
154     long res = wxGetNumberFromUser( "","Pen Width:", "Input Pen Width"
155                   , wxGetApp().GetpModel()->GetPenWidth() , 0, 10, this );
156     wxString msg;
157     if ( res == -1 )
158         msg = "Invalid number entered or dialog cancelled.";
159     else
160     {
161         msg.Printf("Pen width: %lu", res );
162         m_cmdprocessor.Submit(new CmdSetPenWidth(res));
163         wxGetApp().GetpModel()->Draw();
164     }
165     SetStatusText(msg);
166 }
167
168 void KochFrame::OnSetPenColor(wxCommandEvent& event)
169 {
170     wxColourDialog coldialog(this, wxGetApp().GetpColdata());
171     wxString msg;
172     if (coldialog.ShowModal() == wxID_OK)
173     {
174         msg="Pen Color set";
175         m_cmdprocessor.Submit(new CmdSetPenColor(coldialog.GetColourData()));
176         wxGetApp().GetpModel()->Draw();
177     }
178     else
179         msg="Color Dialog canceled";
180     SetStatusText(msg);
```

```
181 }
182
183 void KochFrame::OnPrint(wxCommandEvent& event)
184 {
185     wxGetApp().GetpModel()->Print();
186 }
187
188 void KochFrame::OnCopyClipboard(wxCommandEvent& event)
189 {
190     wxGetApp().GetpModel()->Copy2Clipboard();
191 }
192
193 void KochFrame::OnUndo(wxCommandEvent& event)
194 {
195     m_cmdprocessor.Undo();
196 }
197
198 void KochFrame::OnRedo(wxCommandEvent& event)
199 {
200     m_cmdprocessor.Redo();
201 }
202
203 //KochModel-----------------------------------------------------------
204
205 KochModel::KochModel()
206 {
207     m_pview=NULL;
208     m_n=4;
209     /*m_pview=*/
210     new KochView(wxGetApp().GetpFrame(),this);
211 }
212
213
214 KochModel::~KochModel()
215 {
216     if(m_pview)
217         //delete m_pview;
218         m_pview->Destroy();
219         //m_pview->Close();
220 }
221
222 void KochModel::Setn(unsigned int n)
223 {
224     if(n!=m_n)
225     {
226         m_n=n;
227         if(m_pview)
228         {
229             wxString msg;
```

```
230                 msg.Printf("Level changed to %lu", m_n );
231                 wxGetApp().SetStatusText(msg);
232                 m_pview->Draw();
233             }
234         }
235 }
236
237 void KochModel::AddView(KochView *pview)
238 {
239     if(m_pview) delete m_pview;
240     m_pview=pview;
241 }
242
243 void KochModel::RemoveView(KochView *pview)
244 {
245     if(pview==m_pview) m_pview=NULL;
246 }
247
248 void KochModel::SetPenWidth(int w)
249 {
250     if(m_pview) m_pview->SetPenWidth(w);
251 }
252
253 unsigned int KochModel::GetPenWidth()
254 {
255     unsigned int w=0;
256     if(m_pview) w=m_pview->GetPenWidth();
257     return w;
258 }
259
260 void KochModel::SetPenColor(wxColor c)
261 {
262     if(m_pview) m_pview->SetPenColor(c);
263 }
264
265 wxColor KochModel::GetPenColor()
266 {
267     if(m_pview)
268         return m_pview->GetPenColor();
269     else
270         return wxNullColour;
271 }
272
273 void KochModel::Draw()
274 {
275     if(m_pview) m_pview->Draw();
276 }
277
278 void KochModel::Print()
```

```
279 {
280     if(m_pview) m_pview->Print();
281 }
282
283 void KochModel::Copy2Clipboard()
284 {
285     if(m_pview) m_pview->Copy2Clipboard();
286 }
287
288 //KochView-------------------------------------------------------------
289
290 KochView::KochView(wxWindow *pparent, KochModel *pmodel):
291  wxWindow(pparent, -1), m_pmodel(pmodel)
292 {
293     SetPenColor(wxGetApp().GetpColdata()->GetColour());
294
295     if(m_pmodel) m_pmodel->AddView(this);
296
297     Connect( -1, wxEVT_PAINT
298              , (wxObjectEventFunction) &KochView::OnPaint );
299 }
300
301
302 KochView::~KochView()
303 {
304     if(m_pmodel) m_pmodel->RemoveView(this);
305 }
306
307 void KochView::SetPenWidth(int w)
308 {
309     m_pen.SetWidth(w);
310     Refresh();
311 }
312
313 unsigned int KochView::GetPenWidth()
314 {
315     return m_pen.GetWidth();
316 }
317
318 void KochView::SetPenColor(wxColor c)
319 {
320     m_pen.SetColour(c);
321     Refresh();
322 }
323
324 wxColour KochView::GetPenColor()
325 {
326     return m_pen.GetColour();
327 }
```

```
328
329 void KochView::CalcStartCoordinates(unsigned int width
330                                      , unsigned int height
331                                      , int& x1, int& y1
332                                      , int& x2, int& x3, int& y3)
333 {
334     if(width<height)
335     {
336         y1=.5*height+.25*width;
337         y3=.5*(height-width);
338         x1=(.5-.25*sqrt(3.))*width;
339         x2=(.5+.25*sqrt(3.))*width;
340     }
341     else
342     {
343         y1=.75*height;
344         y3=0.;
345         x1=.5*width-.25*height*sqrt(3.);
346         x2=.5*width+.25*height*sqrt(3.);
347     }
348     x3=.5*width;
349 }
350
351 void KochView::Draw(wxDC& dc,int x1,int y1
352                                 ,int x2,int x3,int y3)
353 {
354     if (!m_pmodel) return;
355
356     int n=m_pmodel->Getn();
357
358     DrawEdge(dc,n,x1,y1,x2,y1);
359     DrawEdge(dc,n,x2,y1,x3,y3);
360     DrawEdge(dc,n,x3,y3,x1,y1);
361 }
362
363 void KochView::DrawEdge(wxDC& dc,unsigned int n
364                         ,int x1,int y1,int x2,int y2)
365 {
366     if(n>0)
367     {
368         int x3,y3,x4,y4,x5,y5;
369         x3=2.*x1/3.+x2/3.;
370         y3=2.*y1/3.+y2/3.;
371         DrawEdge(dc,n-1,x1,y1,x3,y3);
372         x4=x1/3.+2.*x2/3.;
373         y4=y1/3.+2.*y2/3.;
374         x5=.5*(x1+x2)-(y2-y1)*sqrt(3.)/6.;
375         y5=.5*(y1+y2)+(x2-x1)*sqrt(3.)/6.;
376         DrawEdge(dc,n-1,x3,y3,x5,y5);
```

```
377             DrawEdge(dc,n-1,x5,y5,x4,y4);
378             DrawEdge(dc,n-1,x4,y4,x2,y2);
379         }
380         else
381             dc.DrawLine(x1,y1,x2,y2);
382 }
383
384 void KochView::OnPaint(wxPaintEvent& event)
385 {
386     if(m_pmodel)
387     {
388         int width,height;
389         GetClientSize(&width, &height);
390         int x1,y1,x2,x3,y3;
391         CalcStartCoordinates(width,height,x1,y1,x2,x3,y3);
392
393         wxPaintDC dc(this);
394         dc.Clear();
395         dc.SetPen(m_pen);
396         Draw(dc,x1,y1,x2,x3,y3);
397     }
398 }
399
400 void KochView::Print()
401 {
402     wxWindow *pparent=wxGetApp().GetpFrame();
403     wxPrintDialogData *pprndata=wxGetApp().GetpPrndata();
404     wxPrintDialog prndialog(pparent,pprndata);
405     if(prndialog.ShowModal()!=wxID_OK)
406     {
407         wxGetApp().SetStatusText("Printer Dialog cancelled");
408         return;
409     }
410     *pprndata = prndialog.GetPrintDialogData();
411     wxDC *pdc = prndialog.GetPrintDC();
412     if(!pdc->Ok())
413     {
414         wxGetApp().SetStatusText(
415                     "Error creating device context");
416         return;
417     }
418
419     int width,height;
420     pdc->GetSize(&width, &height);
421     int x1,y1,x2,x3,y3;
422     CalcStartCoordinates(width,height,x1,y1,x2,x3,y3);
423
424     pdc->SetPen(m_pen);
425     m_font.SetPointSize(int(height/100));
```

```
426     pdc->SetFont(m_font);
427     pdc->SetTextForeground(wxColour(0,0,0));
428
429     int xpos,ypos,twidth,theight;
430     xpos=width/20;
431     ypos=height/15;
432     wxString msg="Koch snowflake";
433     pdc->GetTextExtent(msg,&twidth,&theight);
434
435     pdc->StartDoc("wxKoch printout");
436     pdc->StartPage();
437
438     pdc->DrawText(msg,xpos,ypos);
439
440     ypos+=theight+5;
441     msg.Printf("Depth %lu",wxGetApp().GetpModel()->Getn());
442     //pdc->GetTextExtent(msg,&twidth,&theight);
443     pdc->DrawText(msg,xpos,ypos);
444
445     pdc->GetFont().SetPointSize(int(height/200));
446     msg="by M. Bernreuther";
447     pdc->GetTextExtent(msg,&twidth,&theight);
448     xpos=.99*width-twidth;
449     ypos=.99*height-theight;
450     pdc->DrawText(msg,xpos,ypos);
451
452     Draw(*pdc,x1,y1,x2,x3,y3);
453
454     pdc->EndPage();
455     pdc->EndDoc();
456 }
457
458
459 void KochView::Copy2Clipboard()
460 {
461     int x1,y1,x2,x3,y3;
462     CalcStartCoordinates(1000,1000,x1,y1,x2,x3,y3);
463
464     wxEnhMetaFileDC dc;
465     if(!dc.Ok())
466     {
467         wxGetApp().SetStatusText("wxMetafileDC not Ok");
468         return;
469     }
470
471     dc.SetPen(m_pen);
472     Draw(dc,x1,y1,x2,x3,y3);
473
474     wxEnhMetaFile *pmf = dc.Close();
```

```
475     if (!pmf)
476     {
477         wxGetApp().SetStatusText("Could not create wxMetafile");
478         return;
479     }
480     if(pmf->Ok())
481     {
482         if(wxTheClipboard->Open())
483         {
484             if(wxTheClipboard->SetData(new wxEnhMetaFileDataObject(*pmf)))
485                 wxGetApp().SetStatusText("Clipboard data set");
486             else
487                 wxGetApp().SetStatusText("Error setting Clipboard data");
488             wxTheClipboard->Close();
489         }
490         else
491             wxGetApp().SetStatusText(
492                 "Error opening Clipboard");
493     }
494     else
495         wxGetApp().SetStatusText("wxMetafile not Ok");
496
497     delete pmf;
498 }
499
500 //CmdSetDepth----------------------------------------------------------
501
502 bool CmdSetLevel::Do()
503 {
504     m_oldlevel=wxGetApp().GetpModel()->Getn();
505     wxGetApp().GetpModel()->Setn(m_level);
506     m_commandName.Printf("Set Depth=%lu",m_level);
507     return TRUE;
508 }
509
510 bool CmdSetLevel::Undo()
511 {
512     wxGetApp().GetpModel()->Setn(m_oldlevel);
513     return TRUE;
514 }
515
516 bool CmdSetPenWidth::Do()
517 {
518     m_oldpenwidth=wxGetApp().GetpModel()->GetPenWidth();
519     wxGetApp().GetpModel()->SetPenWidth(m_penwidth);
520     m_commandName.Printf("Set Pen Width=%lu",m_penwidth);
521     return TRUE;
522 }
523
```

```
524 bool CmdSetPenWidth::Undo()
525 {
526     wxGetApp().GetpModel()->SetPenWidth(m_oldpenwidth);
527     return TRUE;
528 }
529
530 bool CmdSetPenColor::Do()
531 {
532     wxColourData *pcoldata=wxGetApp().GetpColdata();
533     m_oldcoldata=*pcoldata;
534     *pcoldata = m_coldata;
535     wxGetApp().GetpModel()->SetPenColor(pcoldata->GetColour());
536     return TRUE;
537 }
538
539 bool CmdSetPenColor::Undo()
540 {
541     wxColourData *pcoldata=wxGetApp().GetpColdata();
542     *pcoldata = m_oldcoldata;
543     wxGetApp().GetpModel()->SetPenColor(pcoldata->GetColour());
544     return TRUE;
545 }
```

6 Other C++ GUI-/Class-libraries

Qt In contrast to wxWindows the Qt library (`http://www.trolltech.com/products/qt/qt.html`) is a commercial product. For the Unix version there is a free edition (`http://www.trolltech.com/products/download/freelicense/`), which the K Desktop Environment `http://www.kde.org/` is based on.

GTK+ The Gimp ToolKit `http://www.gtk.org/` is a GUI toolkit based on C. The gnome desktop environment (`http://www.gnome.org/`) uses gtk+. A Windows port exists (`http://www.gimp.org/~tml/gimp/win32/`), but is under development. Gtk– `http://gtkmm.sourceforge.net/` is a C++ interface for gtk+.

FLTK The Fast Light Tool Kit Home Page can be found at `http://www.fltk.org/`.

V The V Homepage can be found at `http://www.objectcentral.com/vgui/vgui.htm`

Amulet The Amulet Homepage can be found at `http://www.cs.cmu.edu/afs/cs/project/amulet/www/amulet-home.html`

YACL Yet Another Class Library (`http://www.cs.sc.edu/~sridhar/yacl/`) is not developed any longer.

An overview of GUI libraries can be found at `http://www.geocities.com/SiliconValley/Vista/7184/guitool.html`.

You can also do the GUI programming in Java (`http://www.sun.com/java/`)...