Coding User Interfaces from Scratch

How Farbrausch created their Werkkzeugs

dierk 'chaos' ohlerich

farbrausch

Breakpoint '06

Hallo

dierk.chaos.ohlerich
farbrausch

Topics today

All kinds of stuff:

- User interfaces
- Getting started
- Managing large projects
- Garbage Collection

I've done enough talks about 3D...

how to code a user interface

Farbrausch approach:

- Ignore all the nice windows technology
- Use the lowest layer
- Code our own GUI-toolkit

Is that efficient?

- Code that uses toolkit is 22k lines
- Toolkit itself is 8k lines (standard functionality)
- Those 22k lines became a lot easier to write
- Not a single hour was wasted cursing Microsoft

Let's call it a draw...

Design decision: DX or GDI

Use DirectX (OpenGL) for drawing or GDI (Windows 2d)

- DirectX can be very fast when done properly
- GDI has a messy and complicated interface
- GDI is obsolete with Vista (software emulation)
- With DirectX you get notable input lag (better with GL)
- DirectX is not good for small apps (fixed with Vista)
- DirectX is good for switching to fullscreen

design decision: repaint or update

Paint whole screen with double buffering every frame or update only what needs update?

- Checking for updates can be a lot of work
- Flickering can be avoided with double buffering
- Repainting everything is natural and fast with DirectX
- Repainting everything sucks dry notebook batteries
  - Update when there was input
  - Update when there is animation

What makes a user interface

Lots of window classes

- Controls - numeric input, text input, slider, button
- Frames - layout, splitters
- Borders - decoration, toolbars
- Other windows - list, menus, dialog boxes
- App windows - spline editor, operator page

And a GUI manager.

How does a window work?

Every window has 4 virtual functions for IO

- void OnPaint()
  - Repaint the window
- sBool OnKey(sU32 key)
  - Get a key
  - Unhandled keys are passed to parent
  - 32 bit for key and shift, ctrl, repeat, break
- void OnDrag(sDragData &)
  - Mouse has moved
  - 3 modes: start - drag - stop
- sBool OnCommand(sInt cmd)
  - Receive commands from other windows
  - Misc information from GUI system
  - Unhandled messages are passed to parent

Layout

Simple two-pass layout process

- Calculate *requested* size, recursing from leaf to root
- Assign child window position according to request, recursing from root to leaf

So every window has two more virtual functions:

- void OnCalcSize(sInt &x,sInt &y)
  - Tell requested size
- void OnLayout()
  - Position childs

Borders

Borders are handled different:

- Borders of a window are stored in a different list than other child windows
- This makes the window the parent of the border
- During layout, the Borders *steal* screen space from their parent
- Messages are naturally passed from border to parent.
- Easy to use: AddBorder(new sFocusBorder);

Have a look at a simple window

```
void sFocusBorder::OnCalcSize(sInt &x,sInt &y)
{
  x = 2;
  y = 2;
}

void sFocusBorder::OnLayout()
{
  Parent->Inner.Extend(-1);
}

void sFocusBorder::OnPaint2D()
{
  if(Parent->Flags & sWF_CHILDFOCUS)
    sGui->RectHL(Client,sGC_LOW,sGC_HIGH);
  else
    sGui->RectHL(Client,sGC_HIGH,sGC_LOW);
}
```

Not so simple windows

Some windows are real hard work:

- Text/Number control
- Multi line text control

Most are simpler than you might think:

- Scrollbars
- All Frames
- Menus

But all require a lot of tweaking.

Getting started

demo: a typical hello world program

How does one start writing such a large program?

- Sketch
- Write document header
- Create dummy document
- Create dummy windows
- Write display code - OnPaint()
- Write navigation code - demo
- Write editing code
- Tweak, add more features, use it, ...

Look at the size of that thing

werkkzeug3 statistics:

- 150k lines of code
- 4MB of code
- 13 sub-projects
- Used in ca 10 productions
- Done by ryg & chaos (and kb for sound)

Keeping build times low.

What you should never do:

- One source for each class
  - Compile time gets dominated by processing includes
  - Precompiled headers help only with headers that never change
  - Relating classes belong in the same source
  - One or two files for document and one for each nontrivial window class
- Worst of all: windows.h
  - Avoid it at all costs
  - When following Microsofts style, it's in everywhere!
  - In werkkzeug, only 2 files include it
- Minimize header dependencies
  - Say "struct bla *", and "bla" may be unknown
  - Worst case: write all code in headers so that templates work

Memory Management

- C#/Java users have Garbage Collection
- Garbage Collection is often a bad way to do it
- But sometimes it's just the right thing.
- C#/Java users have no choice

Use GC only where you need it: for small control classes with no apperent ownership.

Simple GC in C++

Problems of GC

- You need to know all root pointers.
  - Global Variables
  - Locals / Registers
  - Stack
- You need to know all indirect Pointers

No way to get this in C++, except for Boehm's method.

Simple GC in C++

You need to know all root pointers.

- Just make sure the GC is performend only in the windows message loop.
- Then there will be no root pointers on stack and in registers
- Register global root pointers manually
  - sAddRoot()
  - sRemRoot()

You need to know all indirect Pointers

- Implement a Tag() function for each GC-aware class
- MyClass::Tag() calls ptr->Need() for every pointer it needs

Won't notice a managed pointer in an unmanaged class.

Collecting the Garbage

Now we need to know all Objects

- All managd objects have a common base class: sObject
- sObject constructor registers object in list
- sObject destructor may never be called directly
- Don't create local variables of GC-aware objects.

And how to perform the GC?

- Each object has a NeedFlag.
- Clear all NeedFlags for all known object
- Recurse through all registered Roots, setting the NeedFlags
- Delete all objects with cleared NeedFlag.

The object base class

```
class sObject
{
public:

  // implement these:

  virtual void Tag() {}

  // internal functions:

  sObject();                  // adds the object to GC
  virtual ~sObject() {}       // should never be called by anyone but GC
  sInt NeedFlag;              // flag needed objects
  void Need();                // this implements the GC recursion
};
```

The Garbage collector

```
void sAddRoot(sObject *);          // add an object as GC root
void sRemRoot(sObject *);          // add an object as GC root
void sCollect();                   // do the recursion
```

Implementing a managed class

```
class sWindow : public sObject{
public:
  sWindow *Parent;
  sArray<sWindow *> Childs;
  sArray<sWindow *> Borders;
  void Tag();
  /* ... */
};

void sWindow::Tag()
{
  sWindow *w;

  Parent->Need();

  sFORALL(Childs,w)
    w->Need();
  sFORALL(Borders,w)
    w->Need();
}
```

How Need() and Tag() interact

```
void sObject::Need()
{
  if(this && !NeedFlag)
  {
    NeedFlag=1;
    Tag();
  }
}
```

49Games

49Games is looking for
Graphicians and Coders
of different talents
for game development in Hamburg.

http://www.49Games.com/

Q&A

dierk.chaos.ohlerich

http://www.farbrausch.com/
http://www.xyzw.de/