Immediate Mode Graphical User Interface (IMGUI)

Why do we need a new way of making GUIs?

- Common sentiment - “GUIs are hard”
- Win32 / MFC is powerful, but is not trivial to use:
  - Widget layout tools
  - Decentralized linkages
  - Callbacks
  - IDs

MFC for tools

- Massive Entertainment used MFC for tools/editors:
  - Ground Control mission editor (GenEd)
  - Drömjobbet data editor (JuiceMaker)
  - Ground Control II mission editor (XED)
  - In each case dedicated programmer with large investment in MFC skillz...
- Motivation:
  - “Guis are hard!”
  - We need complex widgets, i.e. tree controls
  - We feel serious when we use MFC :)

Proprietary Game GUI systems at Massive Entertainment

- Required hardware accelerated GUI (DirectX) that worked well with real time rendering
- Ground Control had 2 UI systems
  - IGComponent system for in-game
    - Code driven
  - Management system for front-end
    - Code and data driven
- Drömjobbet i Rosemond Valley
  - Entirely new system (nothing reused from GC)
    - Used semi-standard data description language (Juice)
    - Better tools for editing layout (JuiceMaker)
    - Code and data driven

Proprietary Game UI systems at Massive Entertainment

- Ground Control II, yet another system!
  - MGui (Massive Gui)
    - Part of company framework (MFramework)
    - Re-used Juice/JuiceMaker as design tool
    - Code and data driven
- All systems were:
  - More or less based on MFC design
  - Big - lots of classes
  - Complex - required dedicated coders
  - Fragile – we never really felt we got it right

Proprietary Game UI systems at Massive Entertainment

- World In Conflict
  - New tools
  - Code extension of GCII system
- Future projects
  - Yet another system!

Proprietary Game UI systems at MindArk

- System written ca 1999
  - Mix av software (including asm) and hardware acceleration
- No coder fully understood it
  - Extremely large system, many layers, very decentralized
- No coder dared to replace it
  - Several incomplete replacement attempts
  - Entropia Universe has at least 3 styles of GUI at once
- No senior coder wanted to maintain it
  - Delegate maintenance tasks to new programmers!

Why so much trouble?

“Guis are hard!”

Someone finally questioned this!

- Casey Muratori (www.mollyrocket.com) stumbled upon / “invented” IMGUI while working at Rad Game Tools
  - hObbE at www.spellofplay.com showed me this
  - The original IMGUI presentation .avi changed my life!
    - http://www.mollyrocket.com/video/imgui.avi
    - Sadly this link is broken, but I have a copy!

Why so much trouble?

“Guis are hard!”

or perhaps...

“Guis are in Retained Mode!”

Retained Mode GUIs (RMGUI), i.e. MFC

- Application steps
  - Init
    - Create widgets using framework classes
      - CButton, CList, CCombo, CEdit, CTreeCtrl
    - Resource editors / code generators
    - Subclass framework classes for application specific windows, dialogs and menus
  - Update
    - ???
    - Framework “does stuff”, calls back / messages your app
  - Callbacks / Messages
    - i.e. OnButtonClicked(ID, stuff...)
    - Delete / create / enable / disable widgets dynamically
    - Move state back and forth between framework objects (widgets) and your app
    - Widget ID linkage is central!

Immediate Mode GUIs (IMGUI)

- Application steps
  - Update

```
if(doButton())              //returns true on click
{
    //do something
}

//only draw button/do interaction if appValue == true
if(appValue == true && doButton())
{
    //do something else
}
```

Immediate Mode GUIs (IMGUI)

- "Framework" implementation
  - Button "widget"

```
bool doButton(const Rect& aLayout, const char* aText)
{
    drawRect(aLayout, BUTTON_COLOR);
    drawText(aLayout, aText);

    return mouseCursorInside(aLayout) && mouseButtonClicked();
}
```

Immediate Mode GUIs (IMGUI)

- Application code (i.e. Controller)
  - Radio Button “widget”

```
Rect layout;
int i;

layout.width = 40;
layout.height = 10;
layout.x = 0;
layout.y = 0;

for(i = 0; i < NUM_ITEMS; i++)
{
    if(doRadio(myItem == i, layout, ITEM_NAMES[i])
    {
        myItem = i;
    }

    layout.x += layout.width;
}
```

Immediate Mode GUIs (IMGUI)

- "Framework" implementation
  - Radio Button "widget"

```
bool doRadio(const bool anActiveFlag,
             const Rect& aLayout,
             const char* aText)
{
    drawRect(aLayout,
        anActiveFlag ? ACTIVE_RADIO_COLOR : INACTIVE_RADIO_COLOR);
    drawText(aLayout, aText);

    return mouseCursorInside(aLayout) && mouseButtonClicked();
}
```

Rethink some widgets (IMGUI)

- List controls not needed
  - Just loop app items and doRadio() / doButton() / doText() for each item

```
for(i = 0; i < NUM_ITEMS; i++)
{
    //selection is visible due to use of radio button "widgets"
    if(doRadio(mySelection == i, x, y, ITEM_NAMES[c]))
    {
        mySelection = i;
    }
}
```

Rethink some widgets (IMGUI)

- Tree controls (application)
  - Can be reduced to a TreeNode widget that can be expanded and collapsed
  - Requires a “handle” from the application

```
for(c = 0; c < NUM_CATEGORIES; c++)
{
    //returns true if node is expanded
    //application passes a const void* to identify the node
    //across frames (first param, can be anything unique)
    //actual expand/collapse state is stored inside the gui
    //in a map (const void* <-> bool)
    if(doTreeNode(CATEGORY_NAMES[c], x, y, CATEGORY_NAMES[c]))
    {
        //do gui for expanded node
    }
}
```

Rethink some widgets (IMGUI)

- Tree controls (framework)

```
bool doTreeNode(const void* aHandle,
                const int aX, const int aY,
                const WCHAR* aLabel)
{
    //hardcoded limit to number of const void* <-> bool mappings
    if(myNumHandles < MAX_HANDLES)
    {
        bool& h(handleState(aHandle));
        String s;

        s.format(h ? "-%s" : "+%s", aLabel);

        if(doRadio(h, aX, aY, s))
            h = !h;

        return h;
    }

    return false;
}
```

Rethink some widgets (IMGUI)

- Combo boxes (application)
  - Similar to TreeNode (uses same handle concept)
  - Framework handles expand / collapse gui
  - Framework can turn off all other widget interaction while combo is expanded (to handle overlap)

```
static const char* COMBO_CHOICES[] =
{
    "orange",
    "pink",
    "red",
    "blue",
    NULL,
};

doCombo(myChoice, x, y, COMBO_CHOICES);
```

Rethink some widgets (IMGUI)

- Combo boxes (framework)

```
void doCombo(unsigned int& aChoice, const int aX, const int aY, const char** someChoices)
{
	if(myNumHandles < MAX_HANDLES)
	{
		bool& h(handleState(&aChoice));

		//expanded
		if(h)
		{
			//current choice
			if(doButton(aX, aY, someChoices[aChoice]))
				h = false;	//same choice

			//list
			unsigned int c(0);
			int y(aY);
			while(someChoices[c])		//terminate on NULL
			{
				if(doRadio(c == aChoice, aX, y += buttonHeight(), someChoices[c]))
				{
					aChoice = c;
					h = false;
				}
				c++;
			}
		}
		//collapsed
		else
		{
			if(doRadio(h, aX, aY, someChoices[aChoice]))
				h = true;
		}
	}
}
```

Other advanced widgets in IMGUI

- Edit boxes
- Sliders
- Drag-n-drop
- Color pickers
- Multiple windows / focus control

- All of the above are possible!
  - Examples: http://www.johno.se/software/IMGUI.zip

Implementation tips

- Do key/mouse interaction at time of “widget” call
  - i.e. Controller::doInput()
- Postpone / cache rendering until time to draw
  - i.e. Controller::doOutput()
  - This helps when sorting / batching primitives in i.e. DirectX
  - Also useful when supporting multiple overlapping windows / focus control

Implementation tricks

```
//code from johno's DirectX9 IMGUI
const int Gui::button(const Style& aStyle,
                      const int aX, const int aY,
                      const int aWidth, const bool anEdgesFlag,
                      const WCHAR* aText,
                      Font& aFont, const int aKey)
{
    //add to batches
    addRect(aStyle, aX, aY, aWidth, anEdgesFlag);
    if(aText)
    {
        aFont.addText(aX + BUTTON_TEXT_OFFSET, aY - 1, aText, false,
          aStyle.myFont);
    }

    //sample input state directly
    return buttonClicked(aX, aY, aWidth, buttonHeight(), aKey);
}

void Gui::draw(IDirect3DDevice9& aDevice)
{
    //draw calls all clear batches after drawing
    myRects.draw(aDevice);
    myFont.draw(*mySprite);
    myBoldFont.draw(*mySprite);
    myMouseInsideFlag = false;
}
```

RMGUI traits

- RMGUIs tend to cache lots of application state internally
  - Sync requirements lead to lots of tedious code that “doesn't do anything valuable”
- Systems become decentralized
  - Define gui widgets here...
  - Handle callbacks over there...
  - Need centralized IDs in yet another place...
- Systems become data-driven
- The capabilities of procedural logic are lost
  - Code is more powerful than pure data / code is a superset of data

IMGUI traits

- IMGUIs cache little / no application state
- No sync of data
- Centralized flow control
  - Single code path controls all gui
    - Code flow controls what is on the screen (procedural), as opposed to init-time pre-defined setup (data driven)
- “Widgets” are procedural style function calls
  - “Don't call the method, don't have a widget!”
    - Enables very dynamic guis
    - Dynamic widget creation / destruction / enable / disable is not an issue in IMGUI
- Geared towards real-time rendering, i.e. games

Re-evaluate “classic” widgets?

- Because IMGUI is so dynamic, maybe we don't need all those classic widgets and windows...
  - I personally don't like:
    - Overlapping (i.e windows, drop-down menus)
      - would rather have “dedicated screen space”
    - Free scroll bars
      - implications of clipping widgets / viewports
      - would rather have “snapped pages”
- Questioning these “standards” is risky due to customer expectations
- Games are in a better position to experiment than traditional apps

Links

- https://mollyrocket.com/forums/viewforum.php?f=10
- http://sol.gfxile.net/files/Assembly07_IMGUI.pdf
- http://www.johno.se/software/IMGUI.zip