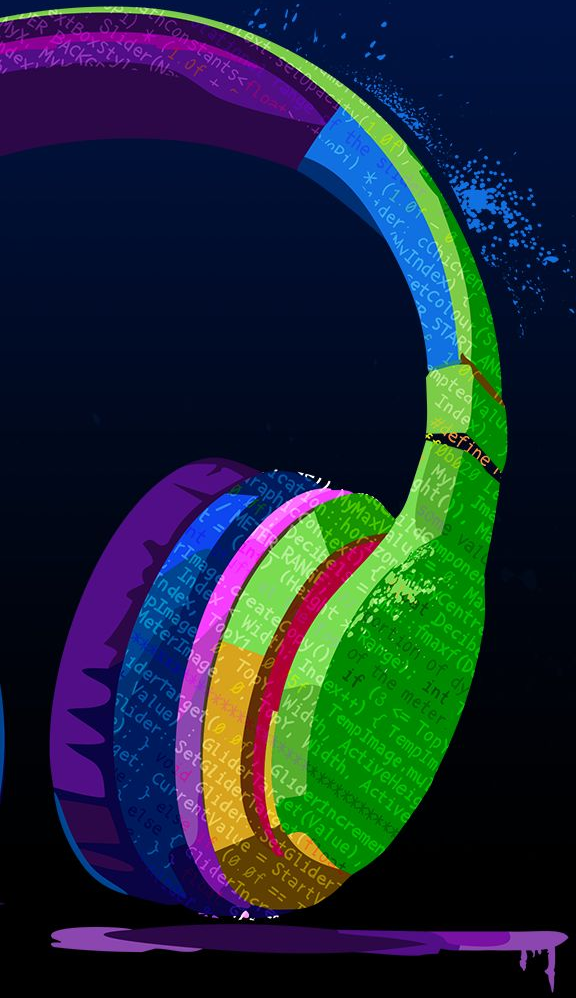


ADC24 Bristol

REINVENTING THE PLUGIN EDITOR

IMMEDIATE MODE GUIS FOR AUDIO PLUGINS

GUSTAV ANDERSSON

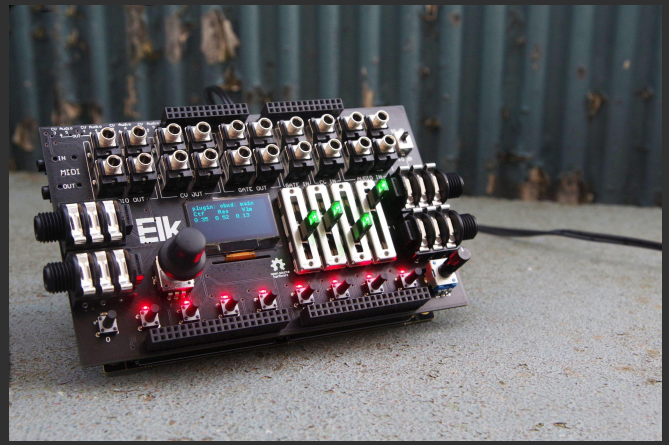
Introduction

- Introduction
- Background
- Immediate mode vs retained mode GUIs
- Drawing, window managers, message loops & threads
- Plugin GUI specifics
- Integrating a specific library (Dear ImGui)
- Customising Dear ImGui
- Summary and conclusions

▶Introduction

Elk Audio OS

- Linux based operating system
- Off-the-shelf SOCs (ARM and x86)
- Less than 1ms roundtrip latency
- Hard realtime performance
- Open Source (*github.com/elk-audio*)



Things that interest me

- Plugin host development
- C++ and performance
- Guitars
- 80s/90s grainy rack effects

Code examples in this talk

- C++
- Windows examples when OS specific
- I recommend using a library to abstract away OS specifics (glfw, SDL2, SFML, JUCE, etc)

▶Background

Not invented here

“Not invented here (NIH) is the tendency to avoid using or buying products, research, standards, or knowledge from external origins. It is usually adopted by social, corporate, or institutional cultures. Research illustrates a strong bias against ideas from the outside.”

The IKEA effect

“The IKEA effect is a cognitive bias in which consumers place a disproportionately high value on products they partially created. The name refers to Swedish manufacturer and furniture retailer IKEA, which sells many items of furniture that require assembly.”

en.wikipedia.org/wiki/IKEA_effect
en.wikipedia.org/wiki/Not_invented_here

KVR Developer Challenge - the deadline I needed

- Build a new plugin
- Every 2-3 year
- Community driven
- Must be released for free

KVR Developer Challenge » 2021

KVR Developer Challenge 2021

KVR DC21 DEVELOPER CHALLENGE 2021

KVR Developer Challenge 2021

Voting is CLOSED! The winners have been announced here (FYI they are listed below in the order they finished - the winner at the top). Thank you and congratulations to everyone!

» Skip To The Downloads «

KVRDC21 Prize Fund

$4,000.00

Donations Closed
See You Next Time!

Welcome to the **KVR Developer Challenge 2021**, the eighth free-for-all audio plugin / audio application / soundware design event.

The **"KVR Developer Challenge"** is for anyone who develops Audio Plugins or Applications and Soundware. The challenge is to create and release a *brand new* free audio plugin, application or sound library / pack / set that will benefit the community at large.

Creativity is key, it can be as simple or as complex as you want - KVR members will vote on the entries and pick the eventual winner using whatever criteria they choose to.

Five cash prizes will be awarded to the top entries and a wildcard pick. Prize moneys are sponsored by the community-funded KVR Developer Challenge 2021 donation pool.

The KVR Developer Challenge began in 2006 and has occurred every 2-3 years ever since. It's delivered gems such as **ProF.E.T. by Ignite Amps**, **Deducktion by Dead Duck Software** and **MPS by Full Bucket Music** back in 2018, **Youlean Loudness Meter**, **Lagrange by Ursa DSP** and **Spaceship Delay from Musical Entropy** in 2016, **Multiply by Acon Digital**, **Nova-67P by vladg/sound** and **Emissary by Ignite Amps** in 2014, and so it goes on...

Roland AG 5 Funny Cat

- Weird envelope filter + compressor / overdrive hybrid
- Model only Soft Distortion Sustainer
- Crude envelope follower + FET based VCA
- No explicit diode clipper

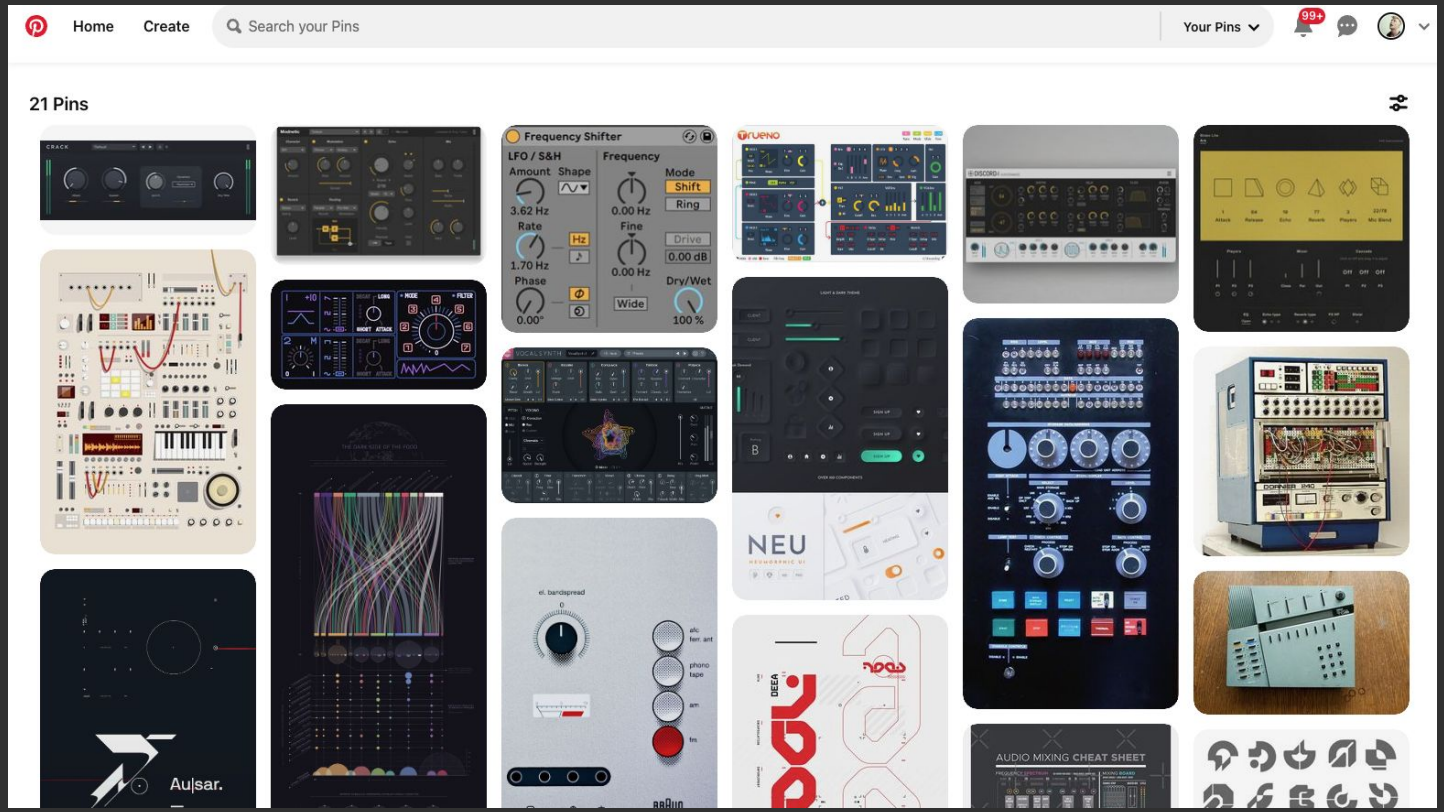
Starting point

- Connect bits and pieces
- Know a lot about plugin APIs from the host’s perspective
- Started work on a plugin wrapper
- Missing a GUI framework (writing my own was out of scope)

Things I didn’t want to do

- Bundling React native / web browser
- Bitmaps and filmstrips knobs
- 100+ MB binaries

Inspiration



GUI framework requirements

- Reasonably minimal / Not too bloated
- Vector based
- Easy to draw and make your own widgets / look and feel
- Suitable for inclusion in an audio plugin
- C/C++, cross platform and no weird build system

A new class of GUI libraries

- Nuklear, NanoVG/NanoGUI, zgui, Dear ImGui, OnGui(from Unity)
- Originates from the gaming industry
- Minimalistic and HW accelerated
- Many use a API paradigm called Immediate Mode GUI

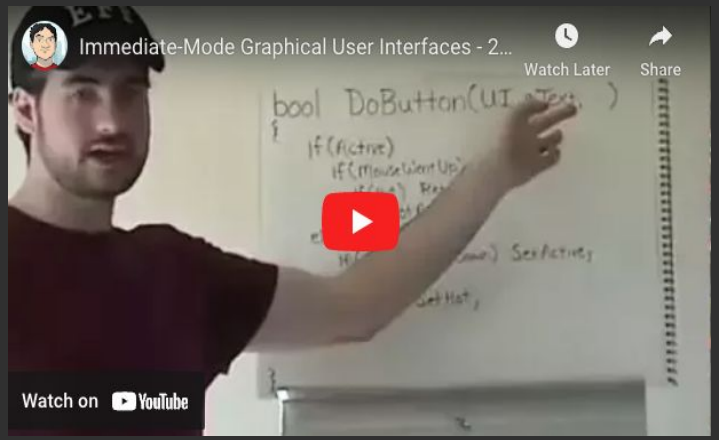
Things I liked

- Different take on GUI APIs
- Minimal in code/binary size and in system memory
- Procedural API - very expressive and to the point
- Looked nice and customisable

- Immediate mode ?

Immediate mode GUI

- Coined by Casey Muratori in 2002 as *“Single-path Immediate Mode Graphical User Interface,”*
- *Zero Memory Widget* by Thierry Excoffier
- Immediate mode drawing (Open GL, etc) applied to GUIs
- More procedural way to approach GUIs than traditional architecture based on OOP hierarchy of widget classes
- Settled on **Dear ImGui** (*github.com/ocornut/imgui*) by Omar Cornut



youtube.com/watch?v=Z1qyvQsjK5Y

Retained mode GUI (traditional GUI)

- Based on widget objects and callbacks
- Mimics how OS window managers work (event driven)
- Everything is a widget, often implemented through polymorphism
- Widgets hold their state
- Tree structure of parents and children

Clickable button with JUCE

```
Class Editor : public juce::AudioProcessorEditor,
               private juce::Button::Listener
{
public:
     Editor::Editor()
     {
          _button.setText("Distortion");
          _button.addListener(this);
          addAndMakeVisible(_button);
     }
      ...
private:
     void buttonClicked(juce::Button* button) override;
     juce::TextButton _button;
}

void Editor::buttonClicked(juce::Button* button)
{
     _audioProcessor->enableDistortion(true);
}
```

Clickable button with JUCE

```
Class Editor : public juce::AudioProcessorEditor,
               private juce::Button::Listener
{
public:
     Editor::Editor()
     {
          _button.setText("Distortion");
          _button.addListener(this);
          addAndMakeVisible(_button);
     }
      ...
private:
     void buttonClicked(juce::Button* button) override;
     juce::TextButton _button;
}

void Editor::buttonClicked(juce::Button* button)
{
     _audioProcessor->enableDistortion(true);
}
```

Clickable button with JUCE

```
Class Editor : public juce::AudioProcessorEditor,
               private juce::Button::Listener
{
public:
     Editor::Editor()
     {
          _button.setText("Distortion");
          _button.addListener(this);
          addAndMakeVisible(_button);
     }
      ...
private:
     void buttonClicked(juce::Button* button) override;
     juce::TextButton _button;
}

void Editor::buttonClicked(juce::Button* button)
{
     _audioProcessor->enableDistortion(true);
}
```

Clickable button with JUCE

```
Class Editor : public juce::AudioProcessorEditor,
               private juce::Button::Listener
{
public:
     Editor::Editor()
     {
          _button.setText("Distortion");
          _button.addListener(this);
          addAndMakeVisible(_button);
     }
      ...
private:
     void buttonClicked(juce::Button* button) override;
     juce::TextButton _button;
}

void Editor::buttonClicked(juce::Button* button)
{
     _audioProcessor->enableDistortion(true);
}
```

Clickable button with JUCE

```
Class Editor : public juce::AudioProcessorEditor,
               private juce::Button::Listener
{
public:
     Editor::Editor()
     {
          _button.setText("Distortion");
          _button.addListener(this);
          addAndMakeVisible(_button);
     }
      ...
private:
     void buttonClicked(juce::Button* button) override;
     juce::TextButton _button;
}

void Editor::buttonClicked(juce::Button* button)
{
     _audioProcessor->enableDistortion(true);
}
```

Clickable button with JUCE

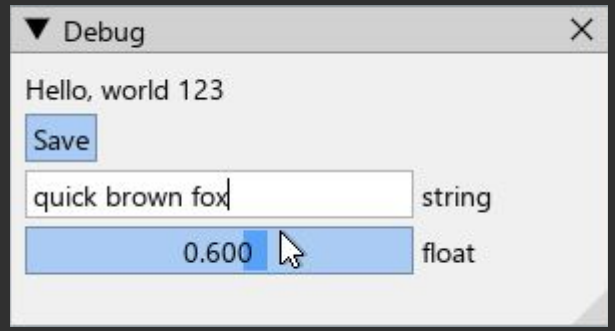
```
Class Editor : public juce::AudioProcessorEditor,
               private juce::Button::Listener
{
public:
     Editor::Editor()
     {
          _button.setText("Distortion");
          _button.addListener(this);
          addAndMakeVisible(_button);
     }
      ...
private:
     void buttonClicked(juce::Button* button) override;
     juce::TextButton _button;
}
```

```
void Editor::buttonClicked(juce::Button* button)
{
     _audioProcessor->enableDistortion(true);
}
```

Clickable button with Dear ImGui

```
Editor::drawUi()
{
    ...
    if (ImGui::Button("Distortion"))
    {
        _audioProcessor->enableDistortion(true);
    }
    ...
}
```

Window example




```
ImGui::Begin("Debug");
ImGui::Text("Hello, world %d", 123);
if (ImGui::Button("Save"))
{
   MySaveFunction();
}
ImGui::InputText("string", buf, IM_ARRAYSIZE(buf));
ImGui::SliderFloat("float", &f, 0.0f, 1.0f);
ImGui::End();
```

Immediate mode GUI

- Procedural - No classes
- Less boilerplate code - All in one place
- Optimised for dynamic UIs instead of static
- Minimises state duplication - Draws current model state
- Drawing is tied with logic - full redraw required to do work
- Stack based instead of parent - child tree structure

Stack example

```
ImGui::PushFont(_large_font):
ImGui::PushStyleColor(ImGuiCol_Text, BLUE_COLOR);
ImGui::Text("Large and Blue headline!");
ImGui::PopStyleColor();
ImGui::PopFont();

if (ImGui::IsItemHovered())
{
  ShowTooltip();
}
```

Lets hook it up to an audio plugin editor

But first a digression…

- Drawing
  window managers
  message loops
  threads

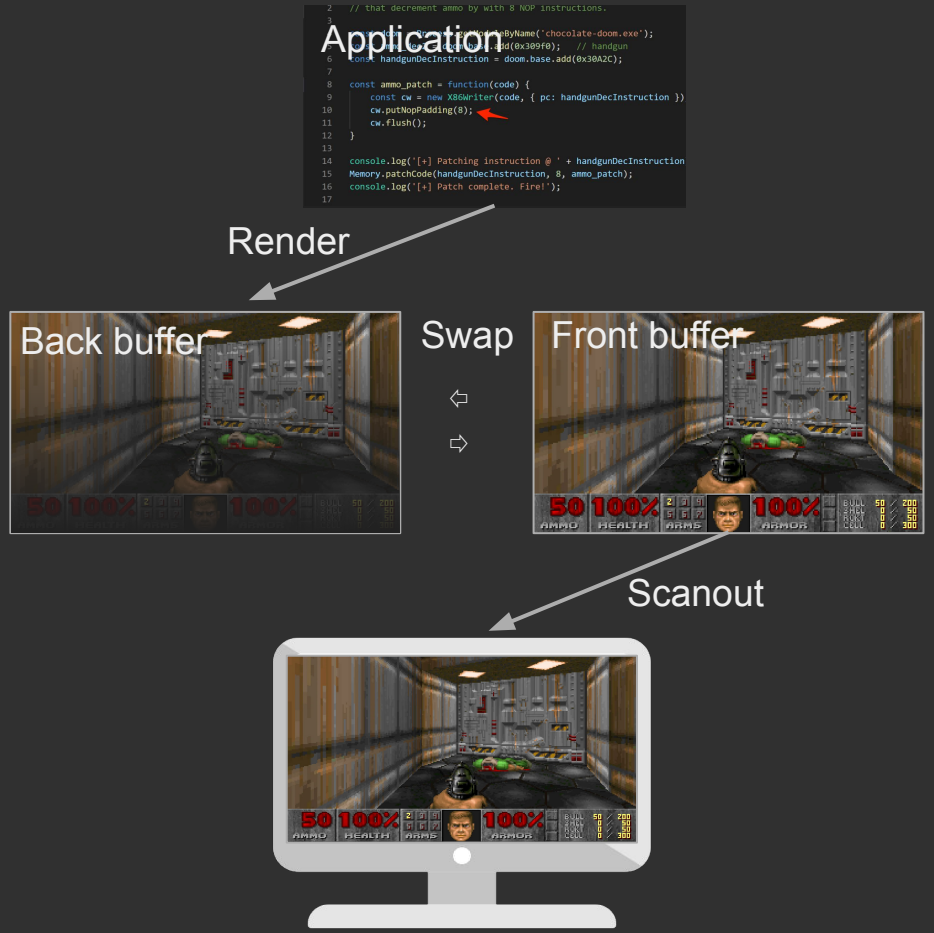
How do computers draw on the screen?

The early days

- No buffering of full image - write directly to display output
- Hardware accelerated sprites
- “Race the beam” for effects and optimisations
- Unparalleled latency

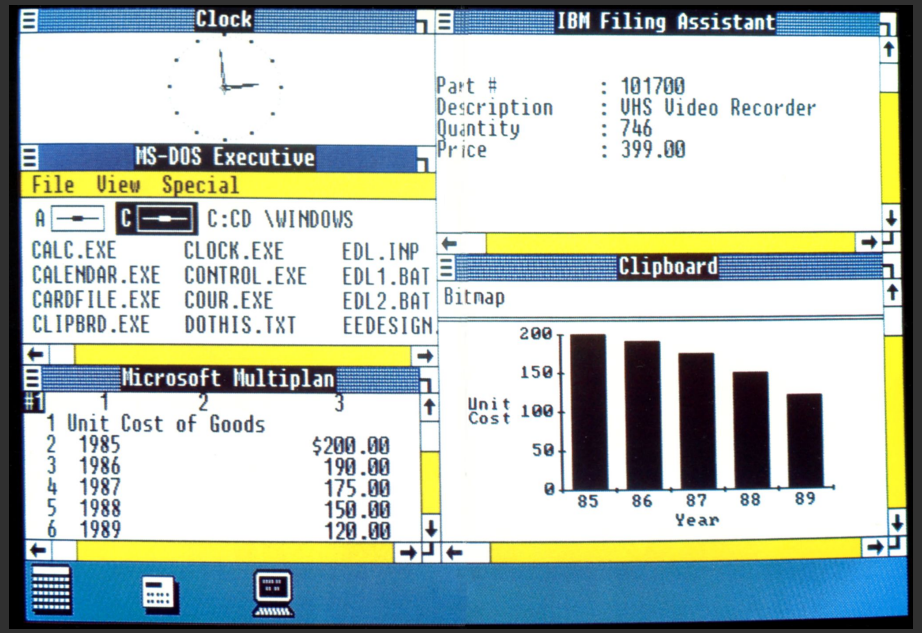
Draw to a framebuffer

- Required for 3D games
- Double buffering - render to an off screen buffer
- Vertical blank - vsync



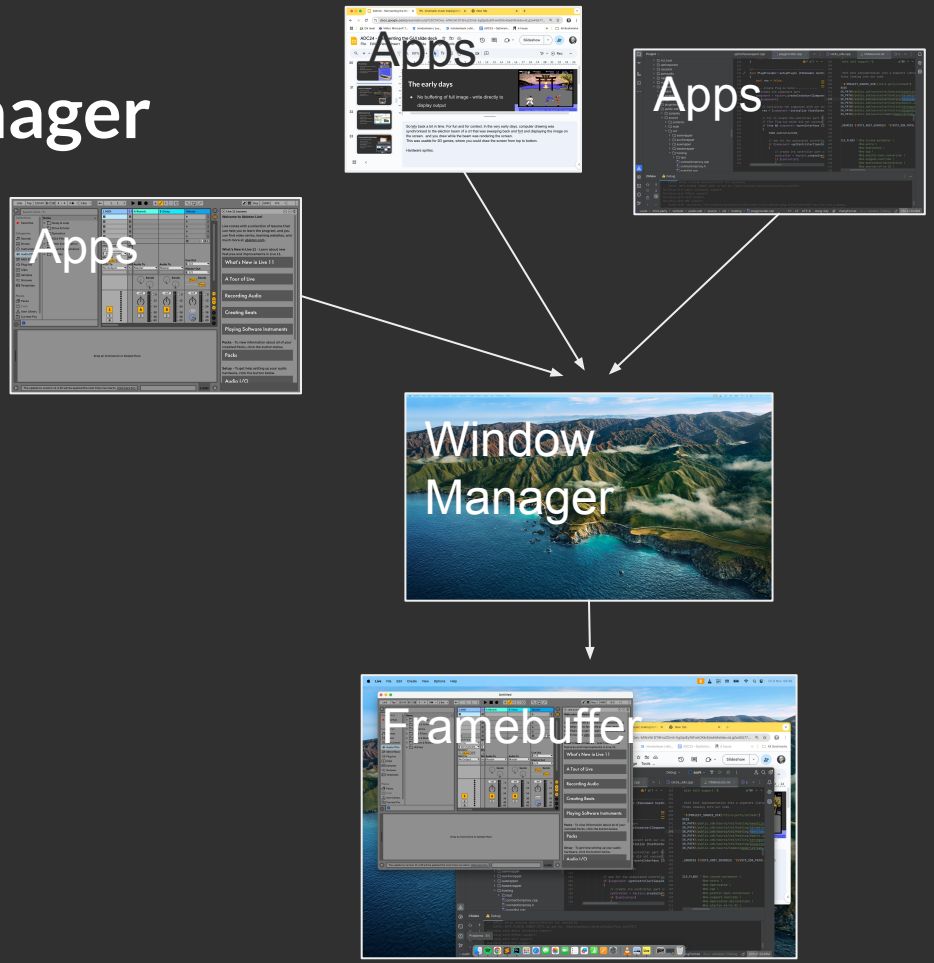
Multitasking is hard

- Sharing the screen is harder
- Window Manager
- Multiple processes need write to the framebuffer
- Windows 1.0 (1985) with a tiling window manager



Compositing Window Manager

- Since Windows Vista (2006) & Mac OS 10.0 (2001)
- Per window off-screen-buffers (or 2 with double buffering)
- Allows for effects, transparency, miniatures, etc



Basic windowed application (event driven)

- Create window
- Attach Window callback to handle events/messages
- Call *GetMessage()* in a loop
    - or *XNextEvent()* (linux) *nextEventMatchingMask()* (Cocoa).

```
MSG message;
while (GetMessage(&message, nullptr, 0, 0) > 0)
{
   TranslateMessage(&message);
   DispatchMessage(&message);
}
```

Window callback

- Called by your message loop
- OS posts messages to the queue whenever something happens
  - Mouse & keyboard input
  - Window moved, resized, closed
  - Clipboard / drag & drop

```
LRESULT CALLBACK windowProc(HWND hWnd,
                            UINT uMsg,
                            WPARAM wParam,
                            LPARAM lParam)
switch (uMsg)
{
   case WM_SETFOCUS:
   {
      ...
   }
   case WM_KEYDOWN:
   {
      ...
   }

   case WM_MOUSEMOVE:
   {
      ...
   }

   case WM_MOUSEWHEEL:
   {
      ...
   }

   case WM_MOVE:
   {
      ...
   }
```

This is your message loop

- Often hidden by the GUI library
- At the bottom of *QGuiApplication::exec()*
- Plugin hosts call *AEffEditor.open()* in this thread

```
MSG message = { };
while (GetMessage(&message, nullptr, 0, 0) > 0)
{
    TranslateMessage(&message);
    DispatchMessage(&message);
}
```

Cross-platform example

- *glfwWaitEvents()* does the equivalent of

  GetMessage() -> TranslateMessage() -> DispatchMessage()

  on every platform

```
...
while (running)
{
    glfwWaitEvents();
}
```

Painting

- WM_PAINT message sent to windows when updates required
- Painting separate from handling input
- “Damaged” parts accessed through *InvalidateRect()* and *GetUpdateRect()*
- Painting happens in your thread, but when the OS decides.

Vsync-driven event handling and painting

- *glfwPollEvents()* instead of *glfwWaitEvents()*
- Drawing synchronised to screen refresh rate instead of paint messages

```
glfwSwapInterval(1)
...
while (running)
{
   glfwPollEvents();
   // Painting, etc..
   ...
   glfwSwapBuffers();
}
```

Hardware acceleration

- OpenGL, Vulcan, Metal, DirectX.
- Sends draw commands to the GPU which draws asynchronously
- Call *glSwapBuffers()* to sync
- OpenGL contexts are not thread safe
- Render backends provided by Dear ImGui

- Plugin GUI specifics

Adapt a GUI framework to an audio plugin

Main issues

1. You don’t own the message loop
2. Managing multiple instances
3. Window creation

You don’t own the message loop

- Don’t create your own - Use the host’s message loop
- Window callback will be called by the host’s message loop
- Easy with Dear ImGui because of frontend / core / renderer split
- Plugin APIs may request special behaviour - Read keyboard events through *IPlugView::onKeyDown()* (VST3), not through WM_KEYDOWN, WM_CHAR

Multiple instances

- Static data and initialising it (last one out turns out the lights)
- Docking branch of Dear ImGui supports multiple viewports
- The more subtle case of multiple plugins built with same framework
- Prefer static linking
- Unique Window ClassName (Windows)

Window creation

- Cross platform libraries wrap the native OS window
- VST API gives you a native OS window
- Create your own window and reparent it to the OS window
- SDL has *SDL_CreateWindowFrom()*
- Window resize can be tricky

- Integrating Dear ImGui in a plugin

Dear ImGui architecture

- Core has no dependencies apart from STL
- Example control frontends and renderers included for most platforms/apis
- Full state contained in 1 context struct. Usually set up as a static variable

Dear ImGui on embedded systems

Control events

Vertex buffers

- Embedded systems usually have no WM (X11) - Direct framebuffer access
- Need a custom control frontend
- Moderate, but predictable CPU load.

My first attempt (too smart for my own good)

- glfw for window handling with static counter for (de)initialising
- Separate render thread for each editor window
- Thread local Dear ImGui contexts
- Worked decently on Linux and Windows
- Failed on Mac due to permissions

I think I have it figured out

Do everything on the message thread, including drawing

Or:

Set up a render thread that draws synced to the frame rate and pump messages to it from the message thread

Single threaded drawing

- Easy to reason about
- Less locking
- Set up timer to call draw function at 20-60Hz on message thread
  - On Linux you need to rely on plugin api timers (CLAP, VST3, LV2)
- Animations could lag

Separate render thread

- Allows for smooth animations synched to screen refresh rate
- Events still need to be handled on the message thread
- Needs sync for window size changes / open / close / minimise, etc
- Included glfw frontend not thread safe

Plugin state - GUI State

- *ImGui::SliderFloat(const char* label, float* v, float v_min, float v_max …)*
- Can point directly into data model - not ideal for an audio plugin
  - Bypasses the host
  - Preserve values across callbacks and atomic updates
  - Performance hit through cache invalidation
- Solution - keep some state
- Dirty flag per parameter

Redraw strategies

- If no interaction and no animations - no need to draw
- Still need more than WM_PAINT, all user input require a redraw
- Built in event queue to handle low fps
- No support for partial redraws

- Customising Dear ImGui

Layout

- Demo window
- By default geared towards a vertical layout
- Can use *ImGui::SameLine()*
- Went for completely fixed layout

Custom horizontal layout

- Fixed widget placement using *ImGui::SetCursor()*
- Useful pattern to divide in high level blocks and return position + width
- ~500 LOC

Useful layout pattern

```
...
float pos_x = scale * BOX_LEFT_PADDING;
pos_x = _draw_gain_box(pos_x, drawlist, scale);
pos_x = _draw_comp_box(pos_x + BOX_SPACING * scale, drawlist, scale);
pos_x = _draw_tone_box(pos_x + BOX_SPACING * scale, drawlist, scale);
pos_x = _draw_scope_box(pos_x + BOX_SPACING * scale, drawlist, scale);
_draw_master_box(pos_x + BOX_SPACING * scale, drawlist, scale);
...

float Editor::_draw_gain_box(float pos_x, ImDrawList* drawlist, float scale)
{
    ...
    return pos_x + width * scale;
}
```

Fonts

- Glyphs are rasterized to a texture (Font Atlas) and rendered
- Uses stb_truetype per default (no hinting), support for FreeType
- Ideally one font per scale and Reload/rebuild on window size change
- Tool to compile fonts into the binary

Custom widgets

- Using only the public api:
  - *ImGui::InvisibleButton()* and custom drawing
- Using the internal api:
  - Use *ImGui::ButtonBehavior(), Imgui::SliderBehavior()* with custom drawing
- Plenty of third party widgets, knobs, file dialogs, node graphs, etc
- Rich ecosystem of extensions and widgets, see:

  github.com/ocornut/imgui/wiki/Useful-Extensions

- Use FontAwesome or OpenFontIcons for symbols

Drawing

- Use *ImGui::Drawlist*
- Draws lines, polygons, basic shapes
- Can use textures

```
// Primitives
// - Filled shapes must always use clockwise winding order. The anti-aliasing fringe depends on it. Counter-clockwise shap
// - For rectangular primitives, "p_min" and "p_max" represent the upper-left and lower-right corners.
// - For circle primitives, use "num_segments == 0" to automatically calculate tessellation (preferred).
//   In older versions (until Dear ImGui 1.77) the AddCircle functions defaulted to num_segments == 12.
//   In future versions we will use textures to provide cheaper and higher-quality circles.
//   Use AddNgon() and AddNgonFilled() functions if you need to guarantee a specific number of sides.
IMGUI_API void  AddLine(const ImVec2& p1, const ImVec2& p2, ImU32 col, float thickness = 1.0f);
IMGUI_API void  AddRect(const ImVec2& p_min, const ImVec2& p_max, ImU32 col, float rounding = 0.0f, ImDrawFlags flags = 0
IMGUI_API void  AddRectFilled(const ImVec2& p_min, const ImVec2& p_max, ImU32 col, float rounding = 0.0f, ImDrawFlags flag
IMGUI_API void  AddRectFilledMultiColor(const ImVec2& p_min, const ImVec2& p_max, ImU32 col_upr_left, ImU32 col_upr_right
IMGUI_API void  AddQuad(const ImVec2& p1, const ImVec2& p2, const ImVec2& p3, const ImVec2& p4, ImU32 col, float thickness
IMGUI_API void  AddQuadFilled(const ImVec2& p1, const ImVec2& p2, const ImVec2& p3, const ImVec2& p4, ImU32 col);
IMGUI_API void  AddTriangle(const ImVec2& p1, const ImVec2& p2, const ImVec2& p3, ImU32 col, float thickness = 1.0f);
IMGUI_API void  AddTriangleFilled(const ImVec2& p1, const ImVec2& p2, const ImVec2& p3, ImU32 col);
IMGUI_API void  AddCircle(const ImVec2& center, float radius, ImU32 col, int num_segments = 0, float thickness = 1.0f);
IMGUI_API void  AddCircleFilled(const ImVec2& center, float radius, ImU32 col, int num_segments = 0);
IMGUI_API void  AddNgon(const ImVec2& center, float radius, ImU32 col, int num_segments, float thickness = 1.0f);
IMGUI_API void  AddNgonFilled(const ImVec2& center, float radius, ImU32 col, int num_segments);
IMGUI_API void  AddEllipse(const ImVec2& center, const ImVec2& radius, ImU32 col, float rot = 0.0f, int num_segments = 0,
IMGUI_API void  AddEllipseFilled(const ImVec2& center, const ImVec2& radius, ImU32 col, float rot = 0.0f, int num_segments
IMGUI_API void  AddText(const ImVec2& pos, ImU32 col, const char* text_begin, const char* text_end = NULL);
IMGUI_API void  AddText(const ImFont* font, float font_size, const ImVec2& pos, ImU32 col, const char* text_begin, const c
IMGUI_API void  AddBezierCubic(const ImVec2& p1, const ImVec2& p2, const ImVec2& p3, const ImVec2& p4, ImU32 col, float th
IMGUI_API void  AddBezierQuadratic(const ImVec2& p1, const ImVec2& p2, const ImVec2& p3, ImU32 col, float thickness, int n

// General polygon
// - Only simple polygons are supported by filling functions (no self-intersections, no holes).
// - Concave polygon fill is more expensive than convex one: it has O(N^2) complexity. Provided as a convenience fo user b
IMGUI_API void  AddPolyline(const ImVec2* points, int num_points, ImU32 col, ImDrawFlags flags, float thickness);
IMGUI_API void  AddConvexPolyFilled(const ImVec2* points, int num_points, ImU32 col);
IMGUI_API void  AddConcavePolyFilled(const ImVec2* points, int num_points, ImU32 col);

// Image primitives
// - Read FAQ to understand what ImTextureID is.
// - "p_min" and "p_max" represent the upper-left and lower-right corners of the rectangle.
// - "uv_min" and "uv_max" represent the normalized texture coordinates to use for those corners. Using (0,0)->(1,1) textu
IMGUI_API void  AddImage(ImTextureID user_texture_id, const ImVec2& p_min, const ImVec2& p_max, const ImVec2& uv_min = ImV
IMGUI_API void  AddImageQuad(ImTextureID user_texture_id, const ImVec2& p1, const ImVec2& p2, const ImVec2& p3, const ImVe
IMGUI_API void  AddImageRounded(ImTextureID user_texture_id, const ImVec2& p_min, const ImVec2& p_max, const ImVec2& uv_m

// Stateful path API, add points then finish with PathFillConvex() or PathStroke()
// - Important: filled shapes must always use clockwise winding order! The anti-aliasing fringe depends on it. Counter-cl
//   so e.g. 'PathArcTo(center, radius, PI * -0.5f, PI)' is ok, whereas 'PathArcTo(center, radius, PI, PI * -0.5f)' won't
inline    void  PathClear()                                          { _Path.Size = 0; }
inline    void  PathLineTo(const ImVec2& pos)                        { _Path.push_back(pos); }
inline    void  PathLineToMergeDuplicate(const ImVec2& pos)          { if (_Path.Size == 0 || memcmp(&_Path.Data[_
inline    void  PathFillConvex(ImU32 col)                            { AddConvexPolyFilled(_Path.Data, _Path.Size,
inline    void  PathFillConcave(ImU32 col)                           { AddConcavePolyFilled(_Path.Data, _Path.Size
inline    void  PathStroke(ImU32 col, ImDrawFlags flags = 0, float thickness = 1.0f) { AddPolyline(_Path.Data, _Path.Size
IMGUI_API void  PathArcTo(const ImVec2& center, float radius, float a_min, float a_max, int num_segments = 0);
IMGUI_API void  PathArcToFast(const ImVec2& center, float radius, int a_min_of_12, int a_max_of_12);          // Use
IMGUI_API void  PathEllipticalArcTo(const ImVec2& center, const ImVec2& radius, float rot, float a_min, float a_max, int n
IMGUI_API void  PathBezierCubicCurveTo(const ImVec2& p2, const ImVec2& p3, const ImVec2& p4, int num_segments = 0); // Cub
IMGUI_API void  PathBezierQuadraticCurveTo(const ImVec2& p2, const ImVec2& p3, int num_segments = 0);          // Qua
IMGUI_API void  PathRect(const ImVec2& rect_min, const ImVec2& rect_max, float rounding = 0.0f, ImDrawFlags flags = 0);
```

- Summary & conclusions

Immediate mode GUI - The good stuff

- Like the concept and fun to work with (could be the IKEA effect talking)
- Clean and succinct code
- Quick to prototype and work with
- Small binary size < 2MB

Immediate mode GUI - The bad stuff

- Text/font handling is awkward and rasterization not the best
- No multi DPI support (yet)
- Somewhat high CPU toll
- Localisation and accessibility support is basic

Was it worth it?

- Yes! I learned a lot. Hence this talk
- Actually released a plugin!
- Gave me a new perspective on how to write GUIs
- Still a passion project, would have prioritised differently for a commercial project

Resources

- github.com/ocornut/imgui
  - Dear Imgui library
- github.com/ocornut/imgui/wiki/Useful-Extensions
- github.com/free-audio/clap-imgui-support
  - Official CLAP support - message thread based
- github.com/schwaaa/clap-imgui
  - CLAP support example - message thread based
- github.com/Krasjet/imgui_juce
  - Juce support - Render thread, probably not complete
- github.com/noizebox/vstimgui
  - My own experiments (not updated, still thread local contexts)

▶ Questions!

gustav@elk.audio