Dear ImGui

A graphical user interface library for C++

Dear ImGui

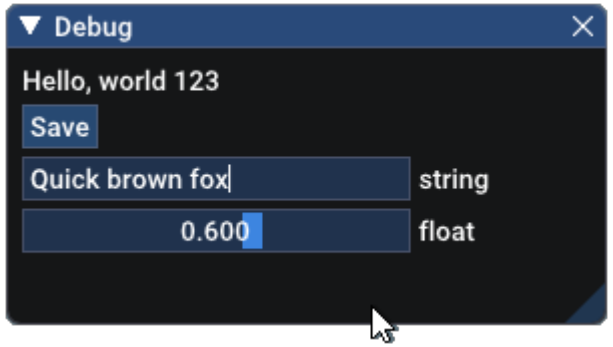
- Also known simply as “ImGui”
  - Immediate Mode Graphical User Interface
- What’s “Immediate Mode”?
  - UI for the frame is built as the code runs
  - Rather than being designed by hand and saved (“Retained Mode”)
  - Run-time vs. design-time
- Our code will responsively (re)create the interface each frame

What’s it look like?

- The code

```
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);
```

The result



- Pretty easy, right?

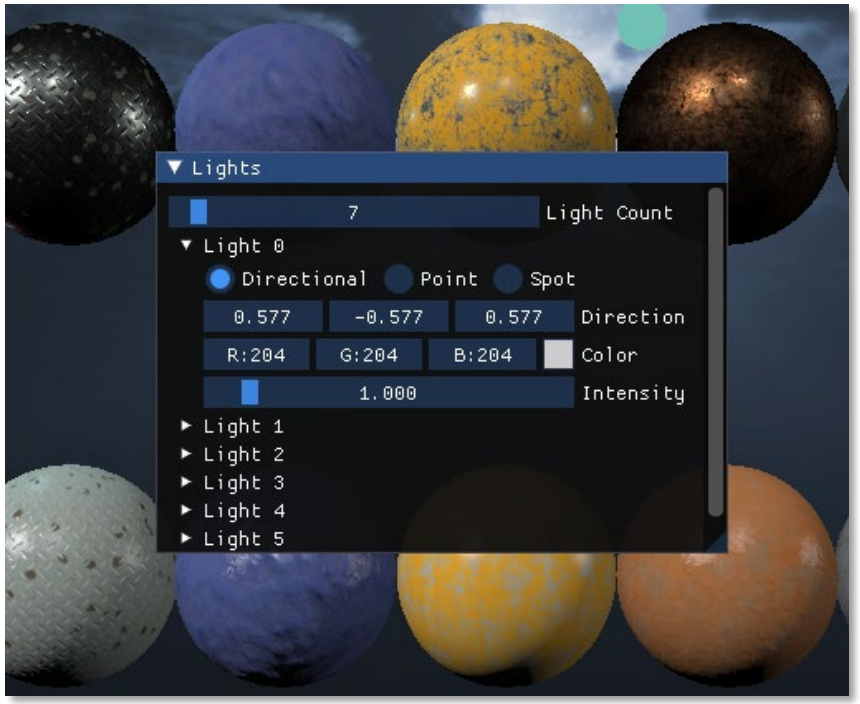
What else can it do?

- A lot

What can we use it for?

- Best suited for debug / demo / utility interfaces
  - Less so for a game's final user interface

- We'll use it for things like…
  - Turning effects on/off
  - Editing game entities live
  - Seeing intermediate results
  - Adjusting values on the fly
  - Etc.



Integrating ImGui

Integrating ImGui – The files

- https://github.com/ocornut/imgui
- Copy the following into your project folder
  - All .h & .cpp files from the repo root
  - DX11 and Win32 files from /backends
- Organize them, please!
  - Put ‘em all in a subfolder

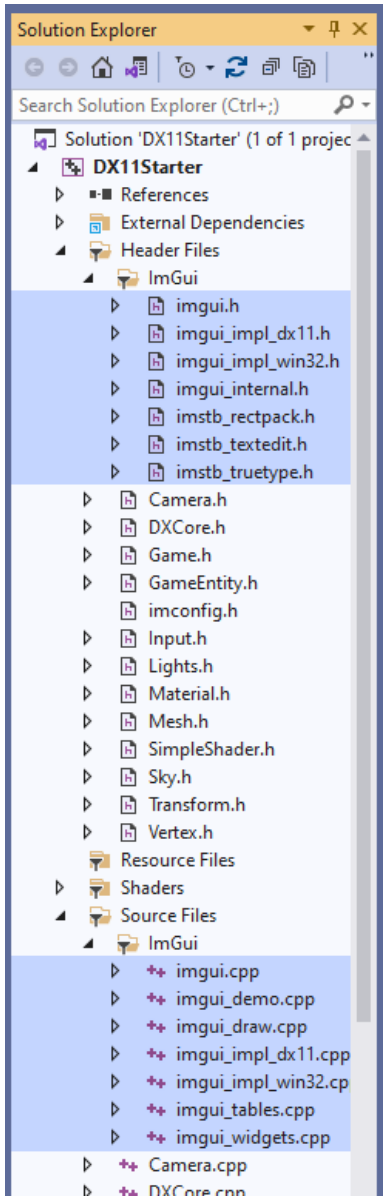
imconfig.h
imgui.cpp
imgui.h
imgui_demo.cpp
imgui_draw.cpp
imgui_internal.h
imgui_tables.cpp
imgui_widgets.cpp
imstb_rectpack.h
imstb_textedit.h
imstb_truetype.h

imgui_impl_dx11.cpp
imgui_impl_dx11.h
imgui_impl_win32.cpp
imgui_impl_win32.h

Integrating ImGui – Visual Studio

- Add all of the files to your project

- You can use “filters” to organize here, too!
  - Filters are like folders for project files
  - They’re not actual folders, though
  - Just a visual organization tool



Integrating ImGui – The code

- Several places we'll need to hook up ImGui to our engine

- Game::~Game()
- Game::Init()
- Game::Update()
- Game::Draw()
- DXCore::ProcessMessage()

Integrating – Header Files

- Include the main header anywhere you want to use ImGui

```
// Assumes files are in "ImGui" subfolder!
// Adjust path as necessary
#include "ImGui/imgui.h"
#include "ImGui/imgui_impl_dx11.h"
#include "ImGui/imgui_impl_win32.h"
```

- At a minimum, include in:
  - Game.cpp
  - DXCore.cpp

Integrating – Game::~Game()

- Clean up at the end of the program

```
// ImGui clean up
ImGui_ImplDX11_Shutdown();
ImGui_ImplWin32_Shutdown();
ImGui::DestroyContext();
```

Integrating – Game::Init()

- Initialize ImGui and set up the various backends

```
// Initialize ImGui
IMGUI_CHECKVERSION();
ImGui::CreateContext();

// Pick a style (uncomment one of these 3)
ImGui::StyleColorsDark();
//ImGui::StyleColorsLight();
//ImGui::StyleColorsClassic();

// Setup Platform/Renderer backends
ImGui_ImplWin32_Init(hWnd);
ImGui_ImplDX11_Init(device.Get(), context.Get());
```

Integrating – Game::Update()

- Turn off input/gui connection
- Feed current data to ImGui
- Prepare the new frame
- Set new input/gui connection
- Show the demo window

- Do this all at the *top* of Update()
  - Even better: make a helper method for it

```
// Feed fresh input data to ImGui
ImGuiIO& io = ImGui::GetIO();
io.DeltaTime = deltaTime;
io.DisplaySize.x = (float)this->windowWidth;
io.DisplaySize.y = (float)this->windowHeight;

// Reset the frame
ImGui_ImplDX11_NewFrame();
ImGui_ImplWin32_NewFrame();
ImGui::NewFrame();

// Determine new input capture
Input& input = Input::GetInstance();
input.SetKeyboardCapture(io.WantCaptureKeyboard);
input.SetMouseCapture(io.WantCaptureMouse);

// Show the demo window
ImGui::ShowDemoWindow();
```

Integrating – Game::Draw()

- Draw ImGui after everything else
  - Immediately before swapChain->Present();

```
// Draw ImGui
ImGui::Render();
ImGui_ImplDX11_RenderDrawData(ImGui::GetDrawData());
```

Integrating – DXCore::ProcessMessage()

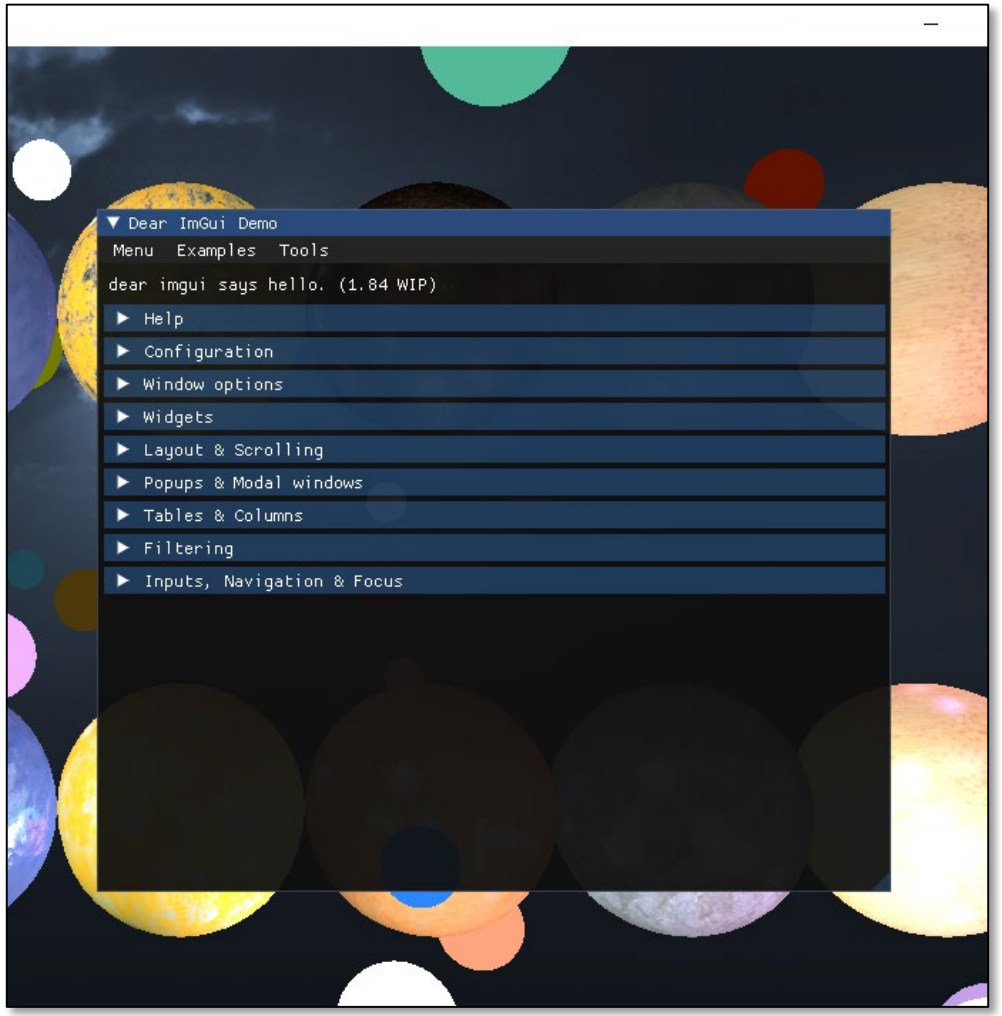
- Add the following at the top of ProcessMessage()
  - Before the switch statement

```
// Forward declare ImGui's handler, then call it
extern IMGUI_IMPL_API LRESULT ImGui_ImplWin32_WndProcHandler(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);
if (ImGui_ImplWin32_WndProcHandler(hWnd, uMsg, wParam, lParam))
{
	return true;
}
```

- Passes OS messages to ImGui
  - Necessary if you ever want text input
  - Or keyboard navigation of the UI

Testing

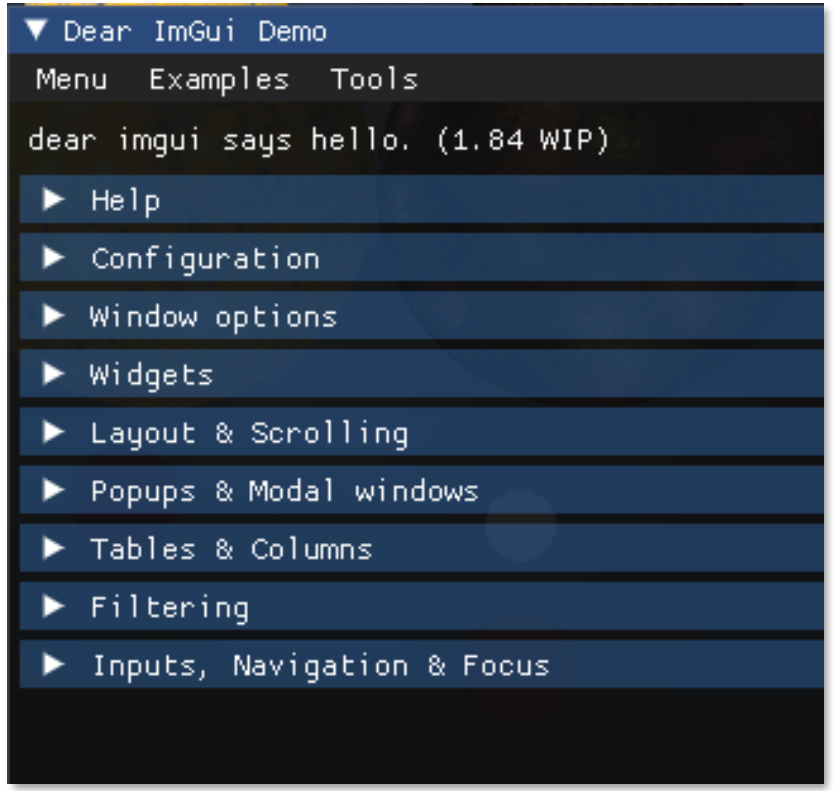
- If you did it correctly, you should see the demo window →



Building a UI

First: Check out the demo window!

- Examples of almost everything
- Entirely within `imgui_demo.cpp`
  - Meant to be a reference
  - Dig in there for examples
- If it does something you want…
  - Go find that code
  - Structure yours similarly

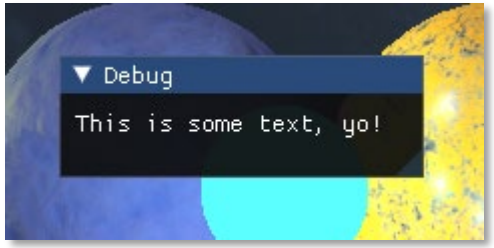


Basic usage

- Build your UI during Update(), not Draw()
  - Must happen after new frame initialization steps

- Call methods from `ImGui` namespace
  - Most create corresponding UI elements
  - Some accept pointers to update variables based on input
  - Some also return values

```
ImGui::Text("This is some text, yo!");
```



Example function calls – Custom window

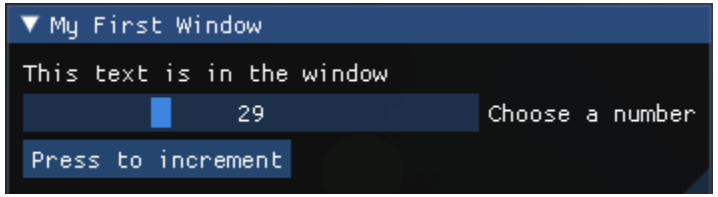
```
ImGui::Begin("My First Window"); // Everything after is part of the window

ImGui::Text("This text is in the window");

// value is an integer variable
// Create a slider from 0-100 which reads and updates value
ImGui::SliderInt("Choose a number", &value, 0, 100);

// Create a button and test for a click
if (ImGui::Button("Press to increment"))
{
      value++; // Adds to value when clicked
}

ImGui::End(); // Ends the current window
```



Example function calls – Complex data

- Want to edit a 3-component vector (like XMFLOAT3)?

```
XMFLOAT3 vec = XMFLOAT3(10.0f, -2.0f, 99.0f);

// Provide the address of the first element
ImGui::DragFloat3("Edit a vector", &vec.x);
```

- Similar for editing colors

```
// Grab first element of the color of the light
ImGui.ColorEdit3(“3-component color editor", &light.Color.x);
ImGui.ColorEdit4(“4-component color editor", &light.Color.x);
```

Odds & ends: Concatenation

- Can use `std::string` and `std::to_string()`
- Then call `.c_str()` to pass result to ImGui

```
std::string iStr = std::to_string(index);
std::string node = "Light " + iStr;

if (ImGui::TreeNode(node.c_str())){}
```

Odds & ends: ID's

- Unique Identifiers
  - Element *text* is also its internal *identifier*
  - Don't use same ID more than once!
- Need two elements w/ same text?
  - Need to generate unique IDs for each!
- IDs can, however, be decoupled from text in a few ways
  - Append ##number to the text
  - Using PushID()/PopID() to differentiate

Odds & ends: Unique ID generation

- Option 1: Name##*num* - Everything after ## is not displayed

```
ImGui::Text("Light 1");
ImGui::DragFloat3("Direction##1", &light1.Direction.x);

ImGui::Text("Light 2");
ImGui::DragFloat3("Direction##2", &light2.Direction.x);
```

- Option 2: Use PushID() / PopID() to adjust internal ID stack

```
ImGui::PushID("Camera");
ImGui::DragFloat3("Direction", &cam.Direction.x);
ImGui::PopID();

ImGui::PushID("Entity");
ImGui::DragFloat3("Direction", &entity.Direction.x);
ImGui::PopID();
```

More examples?

- Too many functions to list them all
- Dig in the demo window code
- Test different functions
- Play around!

References

- The comments of the various functions are quite detailed

- The GitHub repo
  - https://github.com/ocornut/imgui
  - The Readme has examples and links

- Interactive manual
  - https://pthom.github.io/imgui_manual_online/manual/imgui_manual.html
  - Interactively digs through demo window code
  - Very useful (if a little confusing at first)