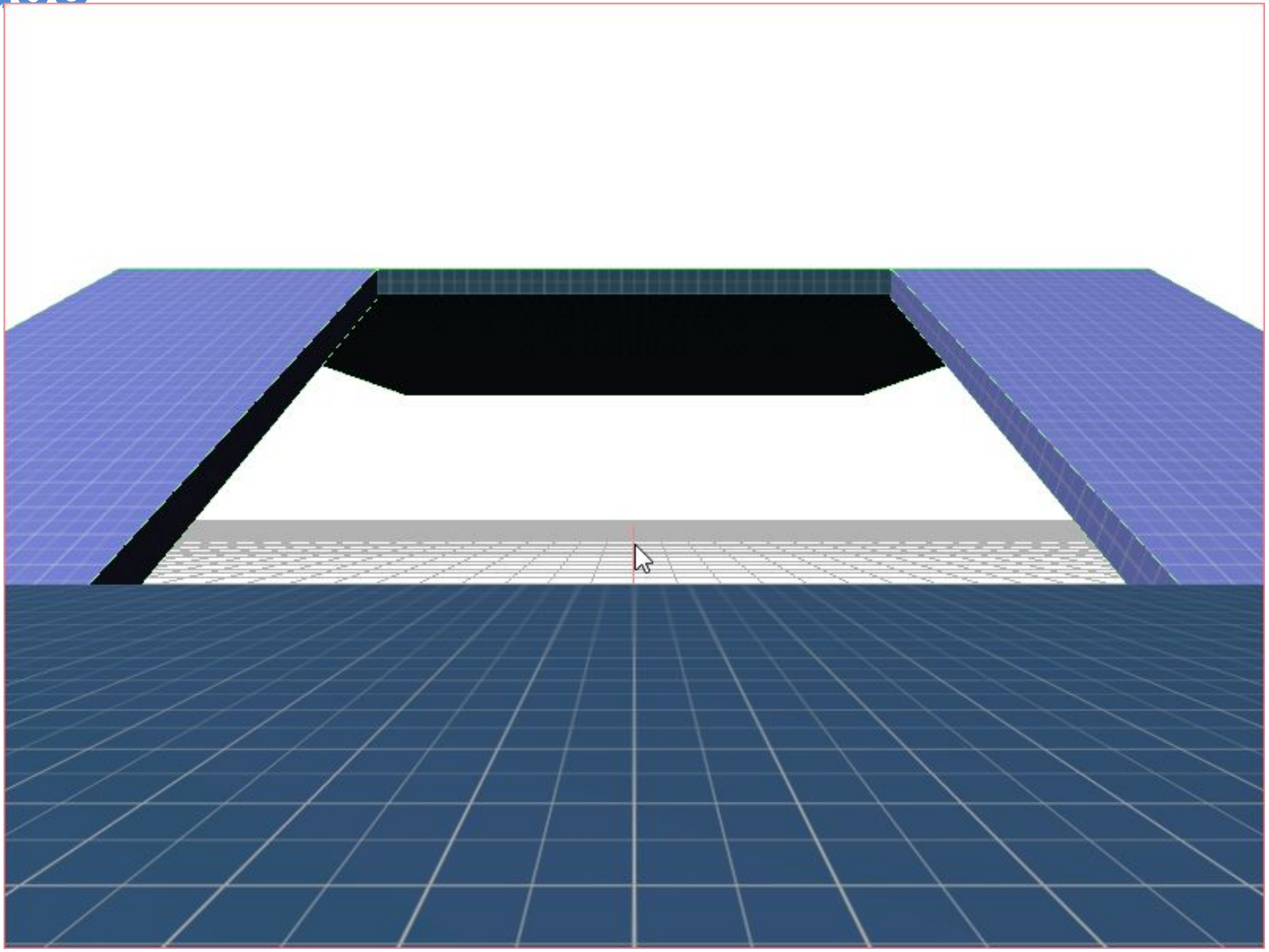
MVD: Engine Programming

09 - GUI

alunthomas.evans@salle.url.edu

laSalle ENG
Universitat Ramon Llull

Video



Real-time tracking/editing

The possibility of tracking and modifying our scene in real-time in the engine is super useful.

So far, the only tracking we've done is with the console. Which obviously is a bit crude.

Could we debug using our current GUI system?

Retained Mode vs Immediate mode GUI

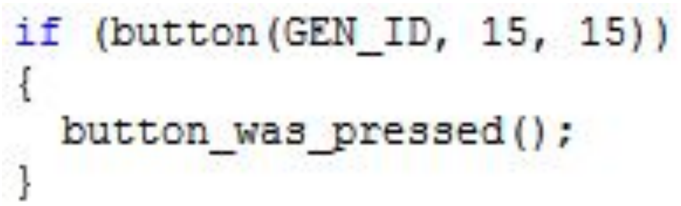
Our current GUI is ‘retained’.

This means that all rendering buffers are *stored* (retained), and have a *state*.

If we want to update the GUI you have to change its state.

An **immediate mode GUI** paradigm is designed to *eliminate state* as much as possible.

Immediate Mode



```
if (button(GEN_ID, 15, 15))
{
  button_was_pressed();
}
```

All buffers generated every frame.

No state stored

Low memory usage

Super easy to code and maintain - no need to have spaghetti code across multiple classes

Creation of buffers every frame is not optimal

More work to customize appearance

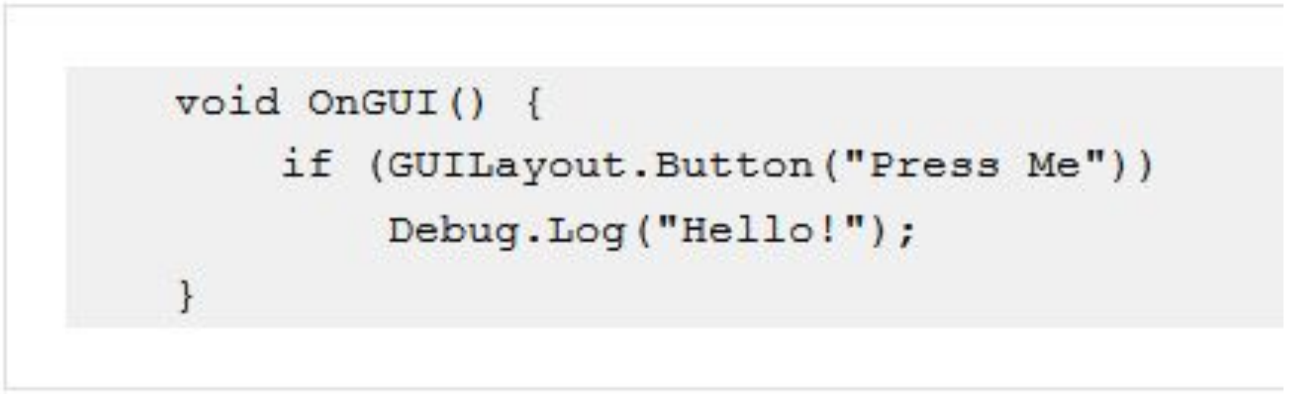
Relative positioning and sizing is difficult

Needs bindings for different platforms/APIs

Case study: Unity

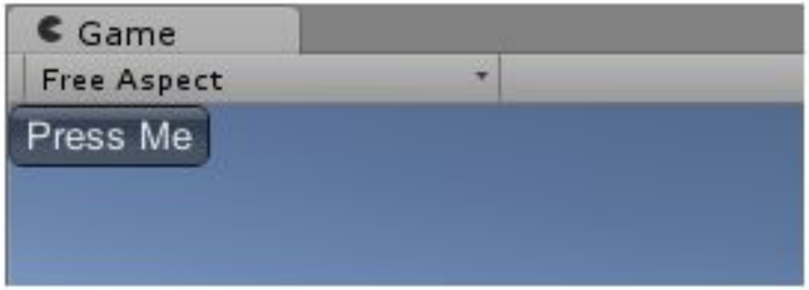
The original GUI system in Unity was an IMGUI system

You can still use it, it’s very useful for debugging



```
void OnGUI() {
    if (GUILayout.Button("Press Me"))
        Debug.Log("Hello!");
}
```

Would result in a button displayed like so:

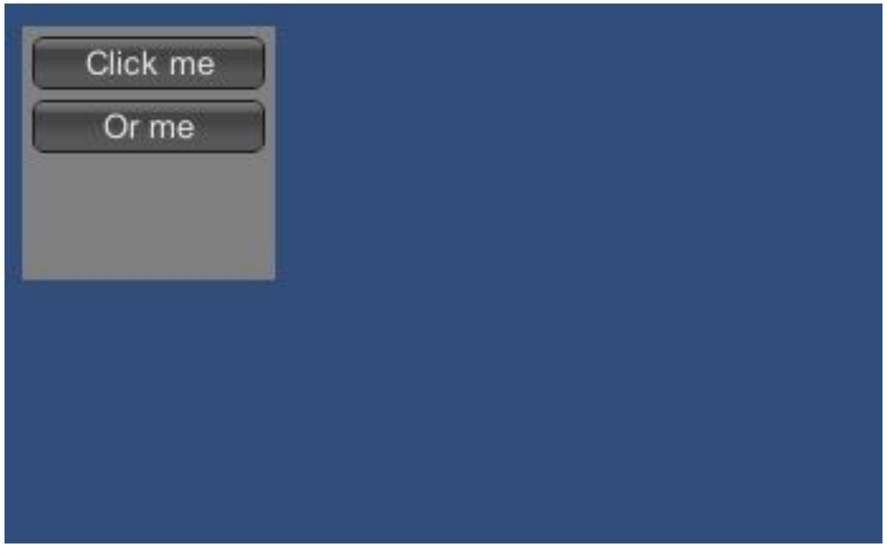


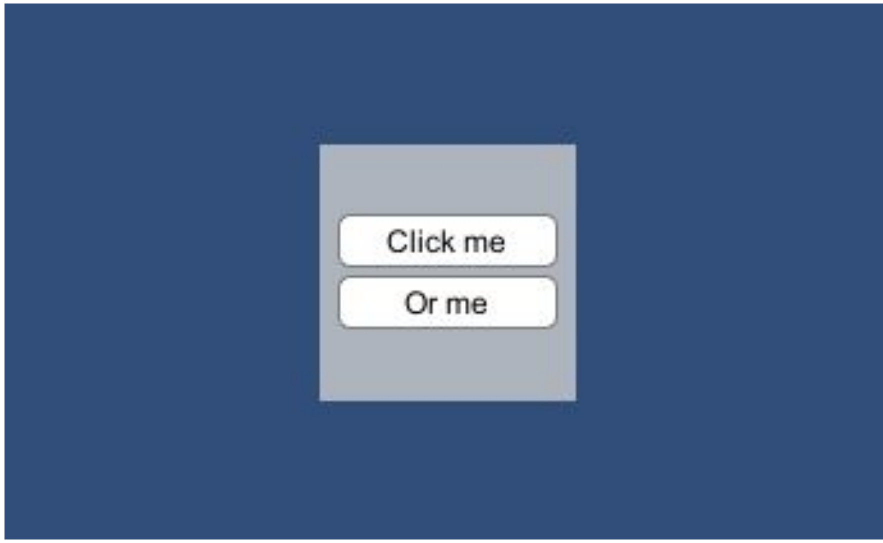
Case Study: Unity

IMGUI vs Canvas

Unity created a canvas system in order to make the GUI system easier to customize

Created UI in Editor to link events etc.





General conclusion

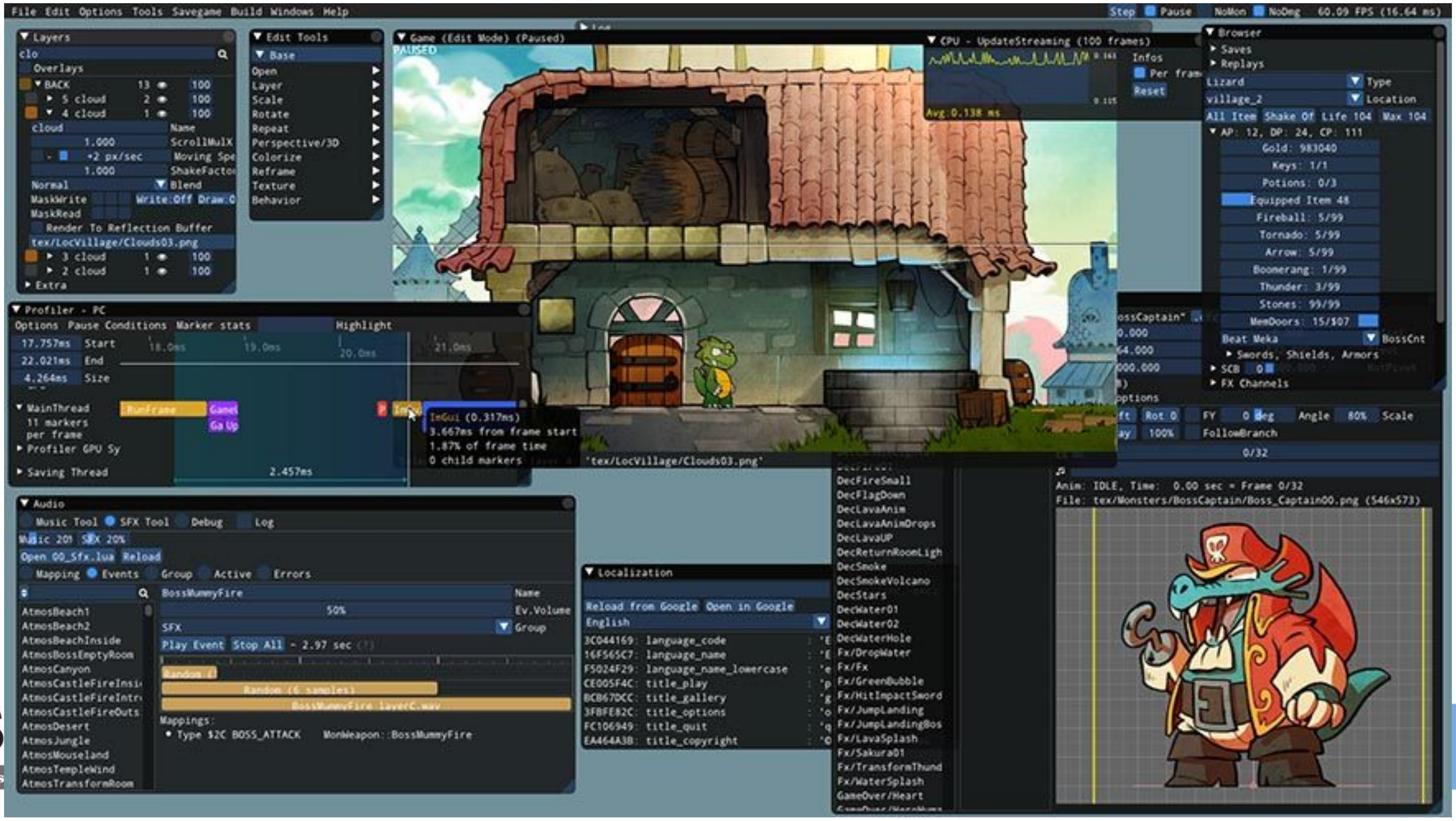
Traditional GUIs are *generally* better for in-game GUIs as **appearance is more controllable, performance can be optimized**

IMGUIs are generally better for any debugging GUI as **events and drawing is much simpler**; **GUI can be created quicker**

Dear imGUI

Open source C++ imGUI library, with multiple crossplatform bindings

https://github.com/ocornut/imgui


File Edit Options Tools Savegame Build Windows Help
Step Pause NoMon NoDmg 60.09 FPS (16.64 ms)
Layers
Edit Tools
Game (Edit Mode) (Paused)
CPU - UpdateStreaming (100 frames)
Browser
Profiler - PC
Audio
Localization
Reload from Google Open in Google
Anim: IDLE, Time: 0.00 sec = Frame 0/32
File: tex/Monsters/BossCaptain/Boss_Captain00.png (546x573)

Dear imGui general usage:

```
ImGui initialisation //in main.cpp
```

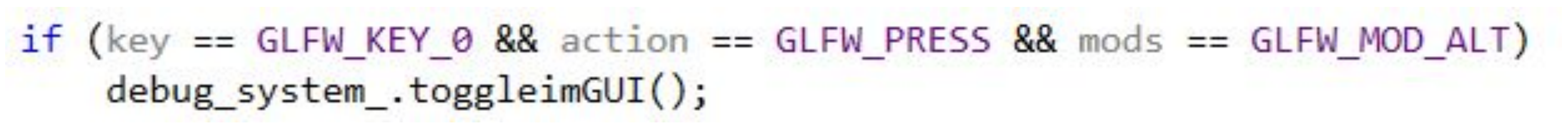
```
//somewhere in the loop
ImGui::Begin()
List widgets in order of appearance in window
ImGui::End()
```

```
ImGui::Render() // can be separate from other code
```

imGUI in Debug System

imGUI code in function in DebugSystem

‘Hot Key access’ defined in game.h, changes boolean to access code



```
if (key == GLFW_KEY_0 && action == GLFW_PRESS && mods == GLFW_MOD_ALT)
    debug_system_.toggleimGUI();
```

https://github.com/ocornut/imgui/blob/master/imgui_demo.cpp

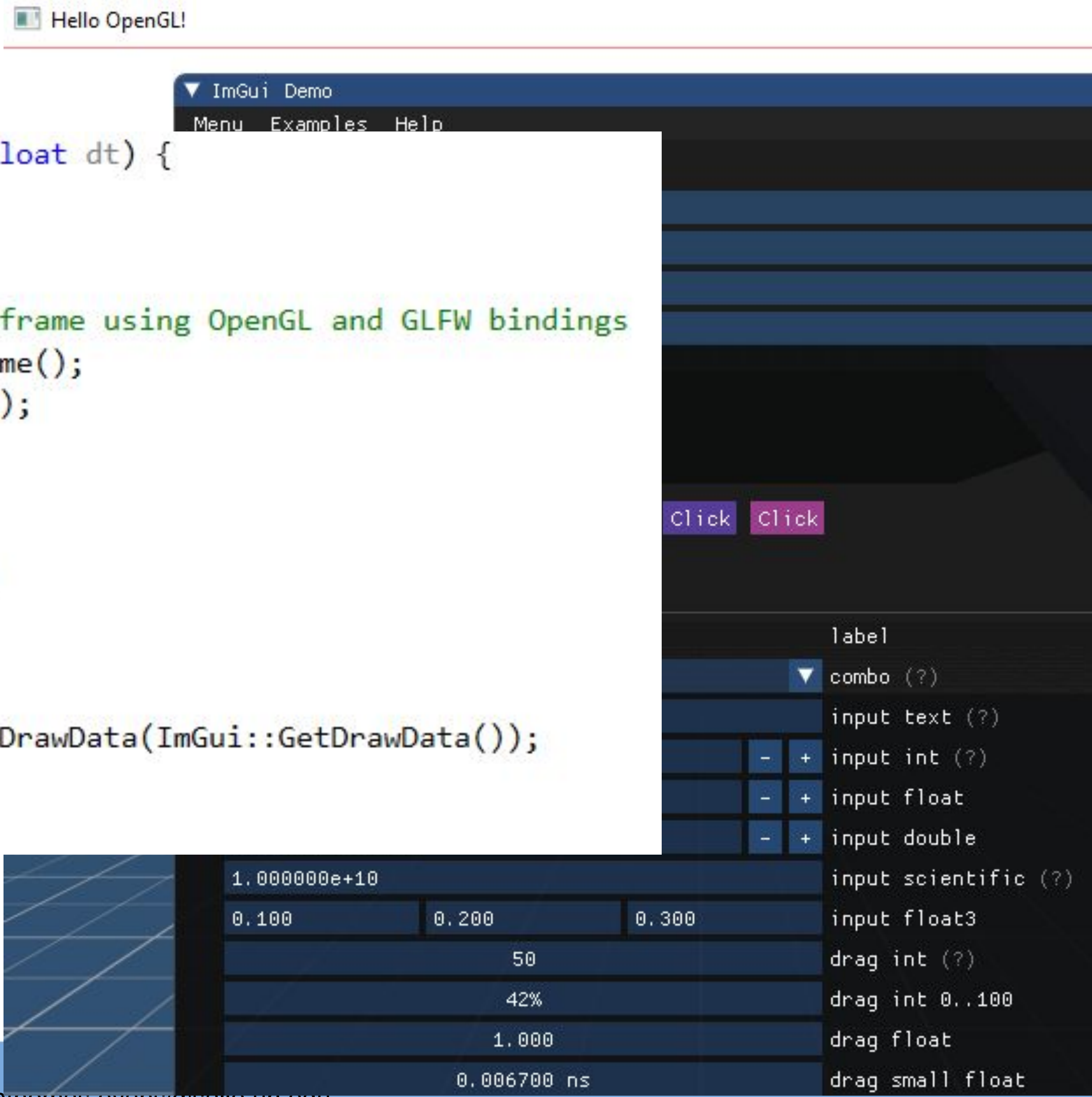
Demo Window

```
void DebugSystem::updateimGUI_(float dt) {

    if (show_imGUI_)
    {
        // Start the Dear ImGui frame using OpenGL and GLFW bindings
        ImGui_ImplOpenGL3_NewFrame();
        ImGui_ImplGlfw_NewFrame();
        ImGui::NewFrame();

        //Demo window
        ImGui::ShowDemoWindow();

        // Rendering
        ImGui::Render();
        ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
    }
}
```



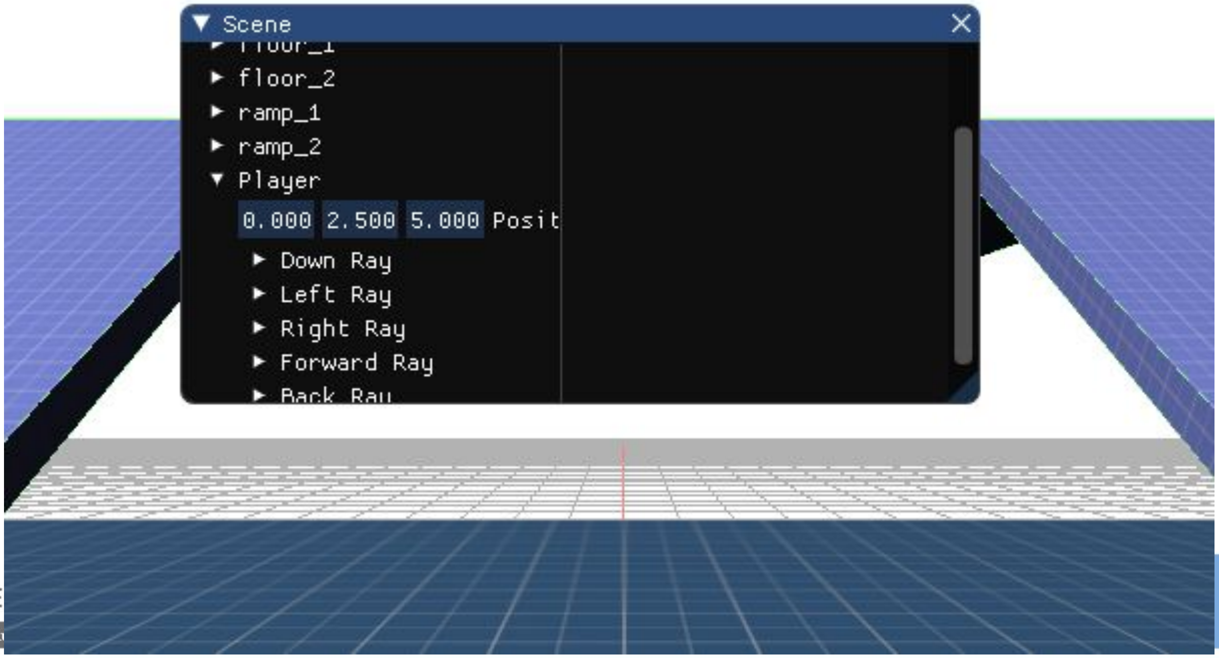
	label
	combo (?)
	input text (?)
	input int (?)
	input float
	input double
1.000000e+10	input scientific (?)
0.100 0.200 0.300	input float3
50	drag int (?)
42%	drag int 0..100
1.000	drag float
0.006700 ns	drag small float

Task

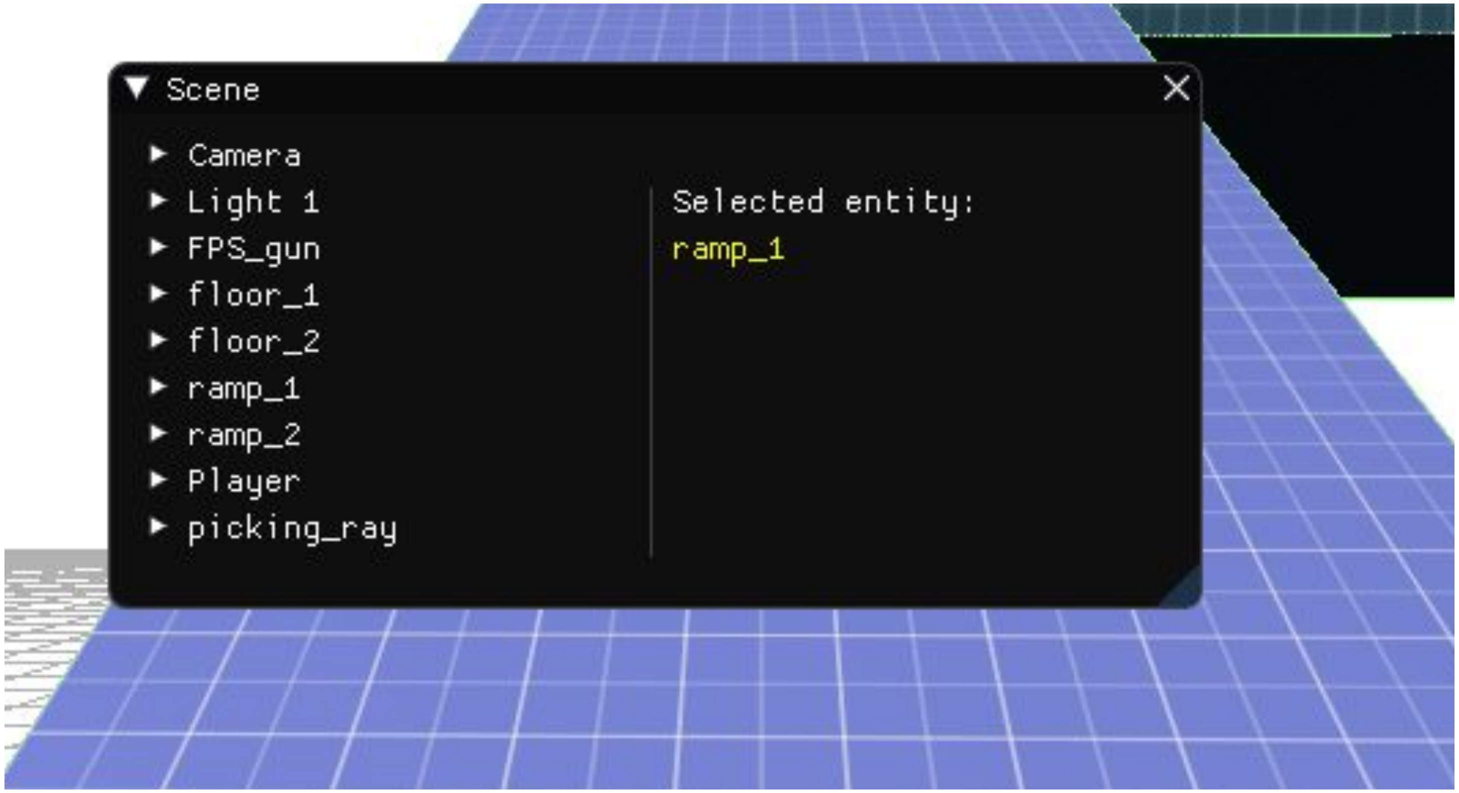
Use ImGui::TreeNode to list all entities in the scene, organised by transform hierarchy

Use the camera code as an example

Advanced - parse entities to create hierarchy



Picking



```
▼ Scene
  ▶ Camera
  ▶ Light 1            Selected entity:
  ▶ FPS_gun            ramp_1
  ▶ floor_1
  ▶ floor_2
  ▶ ramp_1
  ▶ ramp_2
  ▶ Player
  ▶ picking_ray
```

Picking is a super essential part of many games

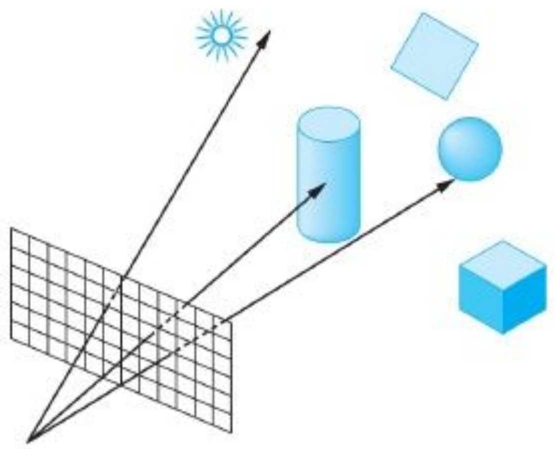
Challenge: When user clicks the mouse on the screen, detect the object currently under that pixel.

What are the approaches?

Approach 1: Ray casting

1) Calculate Ray from camera through point on near plane

2) Calculate intersection of that ray with scene



Calculate Camera ray

Calculate point on near plane in NDC

$p_{near\_plane} = (p_{pixel} / screen\_dimensions) * 2 - 1$

Calculate NDC point in homogenous world coords

$p_{world} = view\_projection^{-1} * p_{near\_plane}$

Calculate Ray:

Ray.origin = camera.position

Ray.direction = $(p_{world}$ - camera.position).normalize()

Collision picking considerations

Projection matrices return **4-component** vectors in homogenous coordinates:

$$[x, y, z, w]$$

where w > 1.

So we must normalize this vector (divide entire vector by w) to get correct result

Calculate collision

Only works if object has a collider!

Not all objects have colliders.

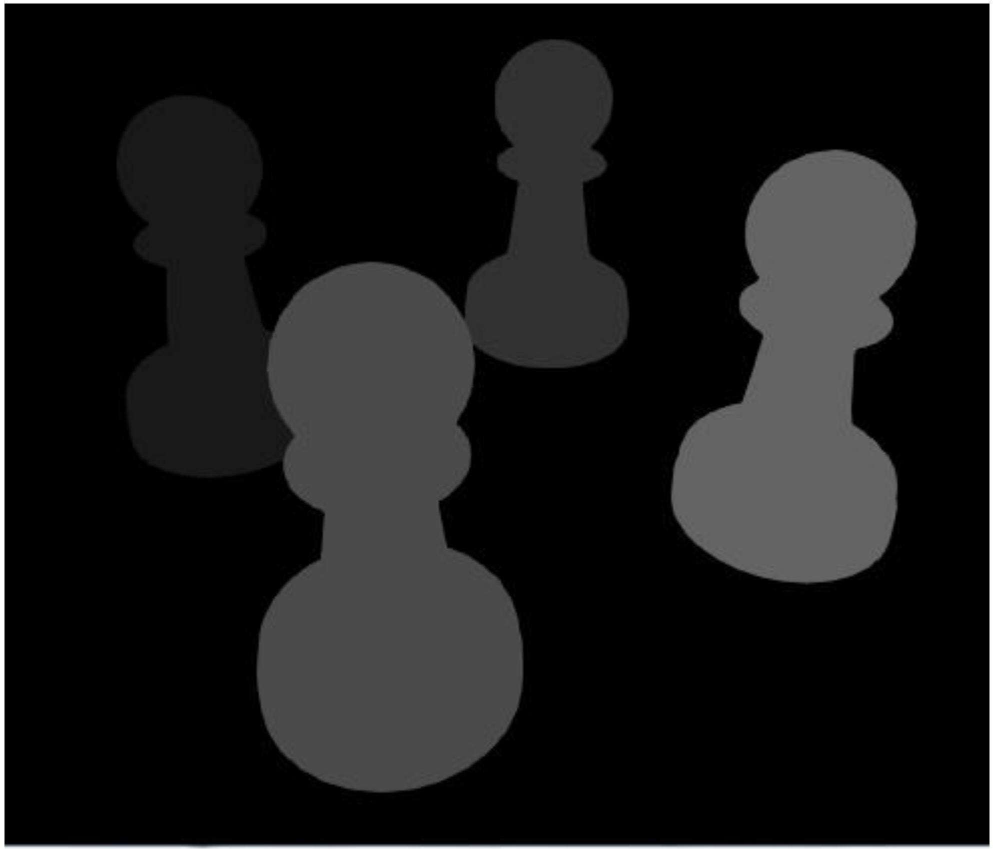
Some engines (ThreeJS) autogenerate an bounding box for each mesh, and traverse entire scene. Expensive but guaranteed to work.

Approach 2: Separate Colour Buffer

Draw scene every frame to low resolution colour buffer.

glReadPixel at location of buffer. **Much faster picking that detecting collisions!**

Need to be able to render to texture first - will learn about this in three or four weeks time



Task

The sample code has a ‘picking ray’ already set up (see DebugSystem.h)

When user clicks mouse, change picking ray to fire into scene (see setPickingRay function)

imGUI reads collider of picking ray, outputs name into the window (code already written).