

A collection of approximately 15 squares in various shades of blue and grey, scattered across the top half of the slide.

# MVD: Engine Programming

08 - GUI

[alunthomas.evans@salle.url.edu](mailto:alunthomas.evans@salle.url.edu)



# GUI

*What do you think we need for a GUI system?*

*How do we draw it?*

# GUIs can be very complicated!

Top 3 things to tackle:

1. Content
2. Positioning
3. Rendering
4. Logic



# GUI Content - Game





# GUI Content - Debugging

The image displays a game's debugging interface with several panels overlaid on a 3D game world. The background shows a landscape with a large structure and some vehicles.

**Rendering control**

- Rendering mode: ☐ Default ☐ Diffuse ☐ Normals ☐ Materials
- Lighting & Shadows
  - Shadows: ☐ r\_shadowsdirectional ☐ r\_shadowspoint
  - Lighting:
    - R: 169 G: 102 B: 0 A: 148 Lighting color
    - R: 162 G: 255 B: 255 A: 100 Ambient lighting color
    - azimuth: 142 deg elevation: 117 deg r\_lightingorientation
    - assets\data\weather\lighting\_b12.json Reload Lighting file (modified)
- Occluded colors
  - R: 0 G: 255 B: 234 A: 255 r\_occludedcolorplayer
  - R: 191 G: 255 B: 250 A: 255 r\_occludedcolorplayerdamaged
  - R: 16 G: 64 B: 61 A: 255 r\_occludedcolorplayerflash
  - R: 255 G: 0 B: 0 A: 255 r\_occludedcolorenyemy
  - R: 255 G: 191 B: 191 A: 255 r\_occludedcolorenyemydamaged
  - R: 64 G: 16 B: 16 A: 255 r\_occludedcolorenyemyflash
  - R: 159 G: 159 B: 159 A: 255 r\_occludedcolorsalvage
  - R: 255 G: 255 B: 255 A: 255 r\_occludedcolorsalvagedamaged
  - R: 64 G: 64 B: 64 A: 255 r\_occludedcolorsalvageflash

**Debug (F1 to toggle panels)**

- devmode ☐ invisible ☐ deaf ☐ infinitemmo ☐ godprops ☐
- God mode: ☐ Off ☐ Player ☐ Everyone
- Audio
  - 0.050 Master
  - 0.500 Music
  - 0.500 Effects
  - 0.800 Voices
- Panels
  - Rendering control Mech debug Range finder
  - Graphics info Money debug Alarm debug
  - Weapon drops Ping
- Miscellaneous
  - Test/help
  - Hide panels
- Build: e0cad2f7520be1b81181761cd0551138d50ad0a 16:51:33 Feb 12 2015 Release-assertions
- FMOD header: 1.05.09
- FMOD linked: 1.05.09
- SDL header: 2.0.3
- SDL linked: 2.0.3
- Copy build string

**Mech debug**

- Mode: ☐ Inspect ☐ Select mech
- Selected 0x102786c8 (assets\data\units\loyalists\loy\_tank\_01.json)
- Clear selection Kill
- AI state
  - NORMAL\_SEARCH state: ambushExitState
  - PLAYER
  - NORMAL\_IDLE
  - NORMAL\_SEARCH
  - NORMAL\_GOTO\_FIRING\_POSITION
  - NORMAL\_SHOOT\_TARGET 4073709551615
  - NORMAL\_GOTO\_LAST\_KNOWN 4073709551615
  - NORMAL\_INVESTIGATE\_POSITION 4073709551615
  - NORMAL\_FLEE
  - AMBUSH
  - BIKEJOUST\_IDLE
  - Invalid
  - Invalid
  - Set Toggle valid Zero
  - Last known velocity: vec2: (0.000000, 0.000000) timestamp: 0.000000
  - Invalid
  - Last known orientation: vec2: (0.000000, 0.000000) timestamp: 0.000000
  - Invalid
  - Last known height: 0.000000
  - Last known radius: 0.000000
  - Priority: 5 (AGGRO\_LOWEST)
  - No LoS
  - Clear Set as custom
  - Set Toggle valid Zero
  - Last known height: 0.000
  - Last known radius: 0.000
  - Priority: AGGRO\_LOWEST
  - Have LoS
  - Clear Set as current Continous set
- Mech state
  - 70.000 health 70.000 max health
  - 30.000 armor 30.000 max armor
  - 35.000 sight\_radius
  - 15.000 sound\_radius
  - 0.900 size
  - 1.435 height
  - 0.000 hover\_height
  - 0.000 danger\_lifetime
  - Dmg rate: 0.000000
  - Expected lifetime: inf

# GUI Content

Images

Text

Live game elements

# Positioning

## Absolute positioning

Element position = (30, 30); size = (100,100)

Pros:

Simple!

Cons:

Relies on window with fixed resolution



# Positioning

## Relative Positioning

Use screen percentages to calculate sizes

Pros:

Simple!

Scales position with window size

Cons:

Scales size with window size!

# Positioning

## Anchor Positioning

Anchor elements to screen limits/center, apply offset

### Pros:

Combines flexibility of relative with accuracy of absolute

Scales position according to window, doesn't scale size

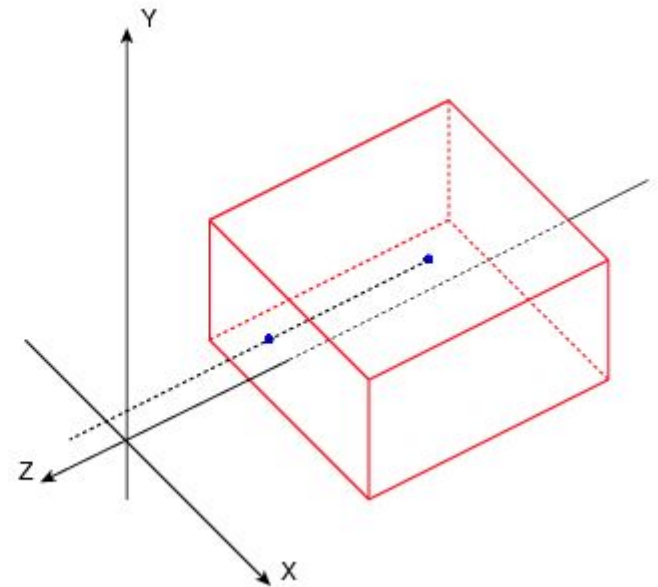
### Cons:

In smaller windows, GUI can take over; in larger windows can get lost

# Rendering

Create orthographic camera the same size as window, centered around origin

```
//create vp  
view_projection.orthographic(  
    -width_/2, //left  
    width_/2, //right  
    -height_/2, //bottom  
    height_/2, //top  
    -0.5, //near  
    -2); //far
```



# Rendering - draw quads

Each quad measures  $-1 \rightarrow +1$  in  $x$  and  $y$  axes. We must *scale* quad to size of GUI Element



## Rendering - scaling

Element model matrix, scaled to it's size - remember quad goes from -1 to +1, so element is always centered

```
model.makeScaleMatrix(  
    el.width / 2,  
    el.height / 2,  
    1.0f);
```

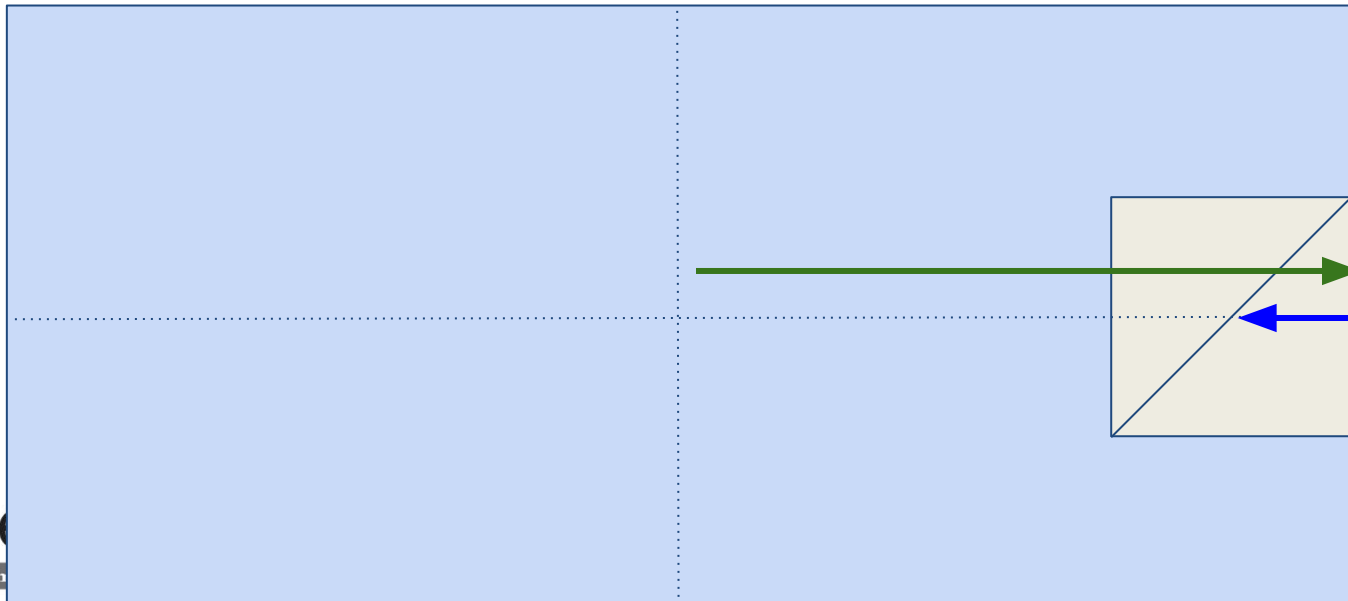


# Rendering - anchoring

Don't forget we translate *center* of element

e.g anchor to right edge of screen

```
model.translate(screen_width/2 - element_width/2, 0, 0);
```



# Rendering images

## Simple texture shader



# TODO

- write positioning and anchoring code

# Detecting interaction

# Simple structure for screen-space boundary

```
struct ScreenBounds {  
    int x_min = 0;  
    int x_max = 0;  
    int y_min = 0;  
    int y_max = 0;  
    bool pointInBounds(int x, int y) {  
        if (x > x_min && x < x_max && y > y_min && y < y_max)  
            return true;  
        return false;  
    }  
};
```

pass mouse coords to pointInBounds, detect click!



# Need to calculate screenbounds for each GUIElement after init

*What is algorithm for converting -1 -> +1 quad, positioned and scaled using model matrix, to **screen space pixel boundaries**?*

implement in GUISystem:lateInit, once all elements are defined

## On click

Pass mouse interaction from GAME to GUISystem as well as ControlSystem.

OnClick: Call pointInBounds with current mouse coords, for each GUIElement

*How do we link program behaviour to clicks?*

# Functional programming



“Uses pure functions to evaluate code”

What’s a pure function?

- doesn’t store or modify any state
- for a given input, always returns same output

Functional programming has a lot of jargon!!

90% of the time it means **passing a function as a variable**

# Functional programming in C++

Purest version uses lambda functions (see them next week)

A less pure version is the `std::function`

## std::function<T> type

T must be return type of function e.g.

```
std::function<void()> onClick;
```

can use std::bind to bind a function. e.g. to bind a class member function:

```
eli.onClick = std::bind(&MovePlatformScript::toggleMove, move_plaform_script);
```



reference to function in class

instance of class



# actual function can be anything

only restriction is that it returns same as T of std::function

```
void toggleMove() {  
    print("pepe");  
    should_move_ = !should_move_;  
}
```

# Our GUIElement Component

```
struct GUIElement : public Component {  
    GLuint texture = 0;  
    GLint width = 0;  
    GLint height = 0;  
    GUIAnchor anchor = GUIAnchorCenter;  
    lm::vec2 offset;  
    ScreenBounds screen_bounds;  
    std::function<void()> onClick;  
};
```

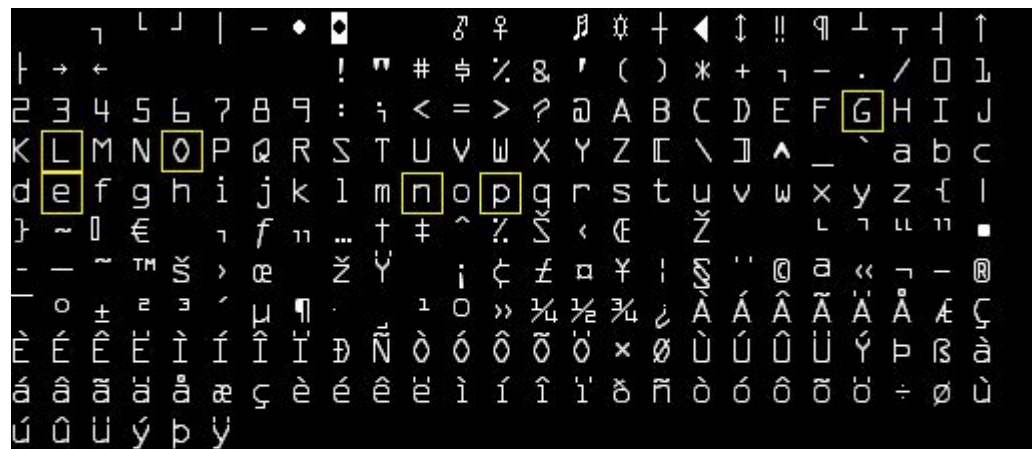
# Task

Create a new GUI Element, with a new image, and a new behaviour (either create a new script, or add a behaviour to MovePlatformScript)

# Text - Font Rendering

*Any ideas?*

# Font glyph textures





# Font glyph textures

One texture with all the glyphs of the font.

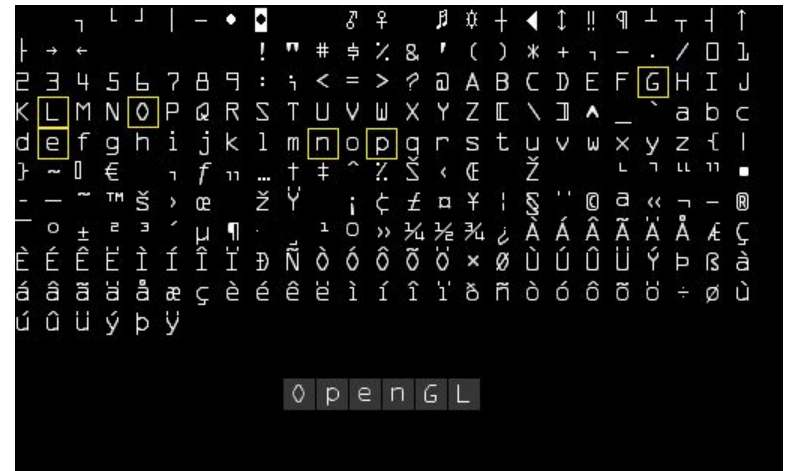
Render a textured quad, change the uv coordinates to draw the required glyph

**Pros:** only use one texture

**Cons:** tricky to change color/size

Have to prepare font in advance

Non-fixed-width fonts need extra metadata

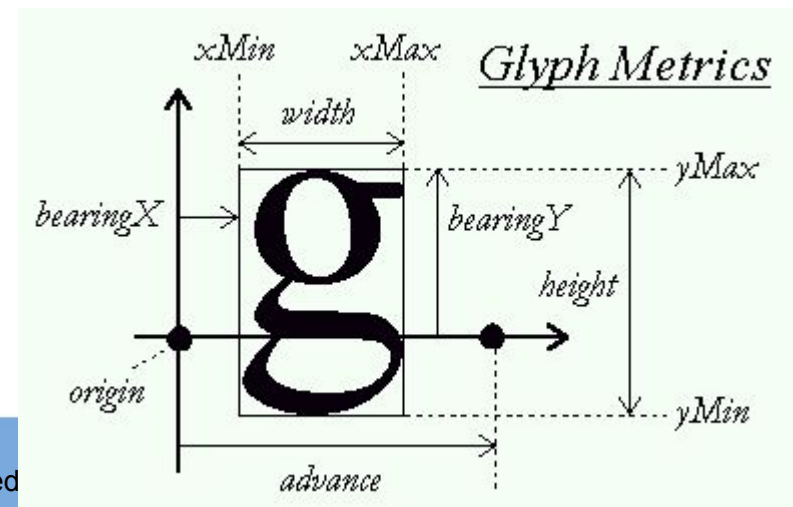


# In engine text rendering

[Freetype](#) - Open source font rendering engine, written in C

Loads fonts from ttf files

Load 'glyph' for individual characters



# FreeType with OpenGL - simple

1. Load FreeType font
2. Set size
3. Load character glyph
4. Create OpenGL texture using character glyph
5. Draw texture on quad of correct size

```
FT_Load_Char(face, 'X', FT_LOAD_RENDER);

glTexImage2D( GL_TEXTURE_2D, 0,
              GL_RED, // SINGLE CHANNEL per pixel
              face->glyph->bitmap.width,
              face->glyph->bitmap.rows,
              0,
              GL_RED,
              GL_UNSIGNED_BYTE, //1 unsigned byte (0-255) per channel
              face->glyph->bitmap.buffer //pointer to data
            );
```

# Freetype shader

Freetype outputs SINGLE CHANNEL TRANSPARENCY

So in shader we read on the red channel, and pass it to alpha channel of out colour.

Remaining channels we set according to a uniform, which controls text colour:

```
#version 330
in vec2 v_uv;


out vec4 fragColor;

uniform sampler2D u_icon;
uniform vec3 u_color;

void main() {
    float final_color = texture(u_icon, v_uv).r;
    fragColor = vec4(u_color, final_color);
}
```

# FreeType - draw quad for each character in string

Draw a separate quad for each character in string



This is sample text



## Pros:

Easiest way of using freetype

## Cons:

Separate drawcall for every character!!!

# FreeType - render string to single texture

More efficient way is to create an 'empty' GL texture, then draw/copy freetype glyph data to partial areas, using `glTexSubImage2D`.

Empty texture:

```
std::vector<unsigned char> empty_data(tex_width * tex_height, 0);

glTexImage2D(GL_TEXTURE_2D, 0,
             GL_RED, tex_width, tex_height,
             0, GL_RED, GL_UNSIGNED_BYTE,
             &(empty_data[0]));
```

# glTexSubImage2D

```
//draw only a portion of the gl texture
glTexSubImage2D(GL_TEXTURE_2D, //type of texture
    0, //mipmap target
    x, //x offset (from left)
    y, //y offset (from right)
    face->glyph->bitmap.width, //width
    face->glyph->bitmap.rows, //height
    GL_RED, //format
    GL_UNSIGNED_BYTE, //data type
    face->glyph->bitmap.buffer); //pointer to data
```

# How to calculate x and y?

Useful freetype metrics:

//height of glyph from baseline

face->glyph->metrics.horiBearingY

//how much to advance pen between character and lines

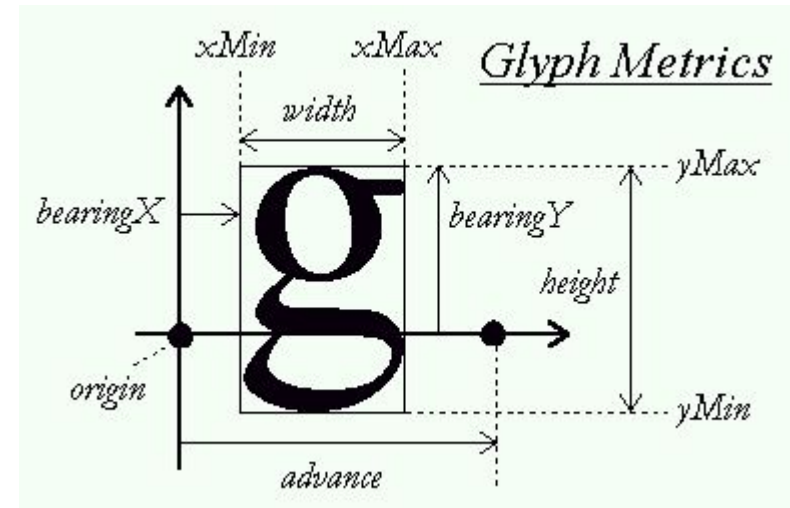
face->glyph->metrics.horiAdvance

face->glyph->metrics.vertAdvance

//glyph width and height

face->glyph->bitmap.width

face->glyph->bitmap.rows





# Freetype units and orientation

...are 1/64th of a pixel!!

So must divide all numbers by 64 to get pixel value!

ALSO

OpenGL texture *y* axis goes down...

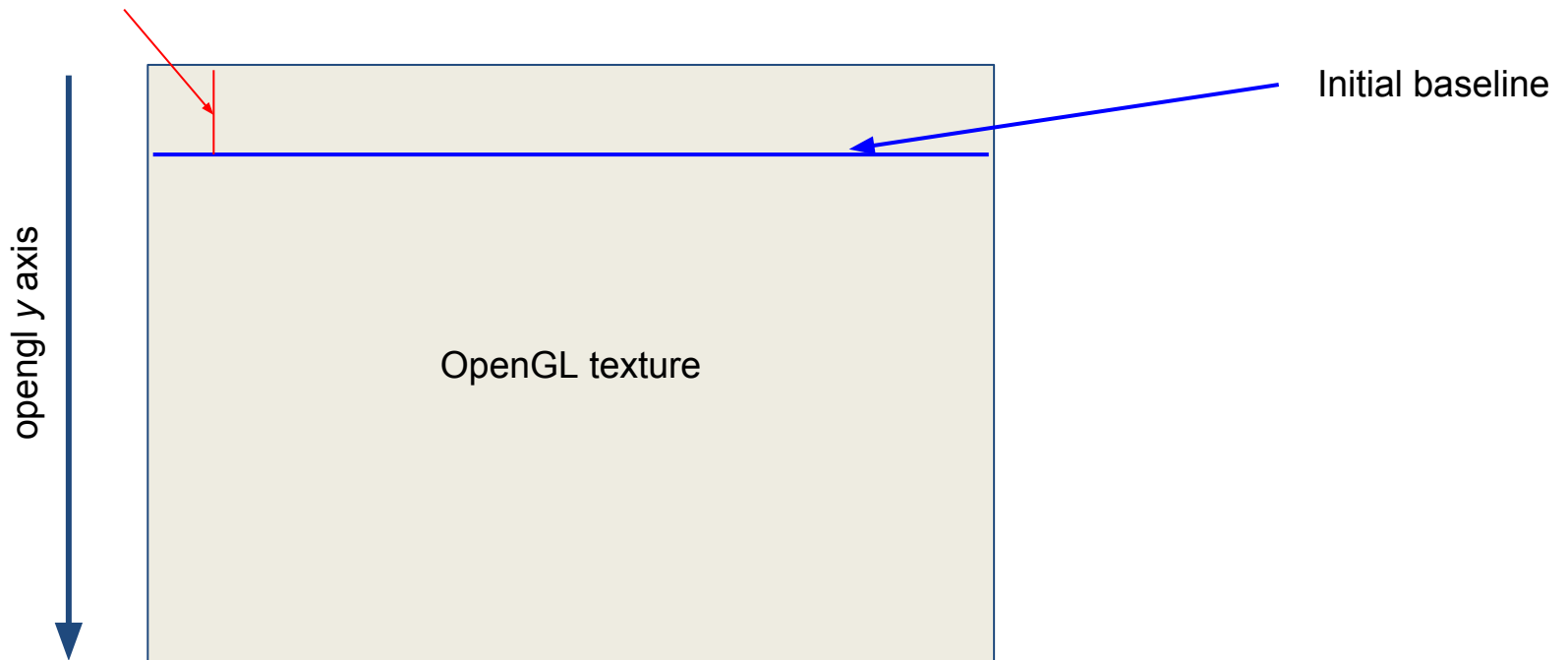
...Freetype *y* axis goes up



# Multiple Freetype glyph to texture algorithm

1. 1st loop through string characters, to get maximum bearing in Y (this will be initial baseline of texture)

Maximum detected  
horiBearingY in string



# Multiple Freetype glyph to texture algorithm

2. 2nd loop through characters, copying their data to texture using `glTexSubImage2D`
3. Use glyph metrics to advance x and y variables to correct spot to draw next glyph

*What happens at end of row?*

*How do you deal with newline characters?*

# Task

Write multiletter, multirow text texture using FreeType