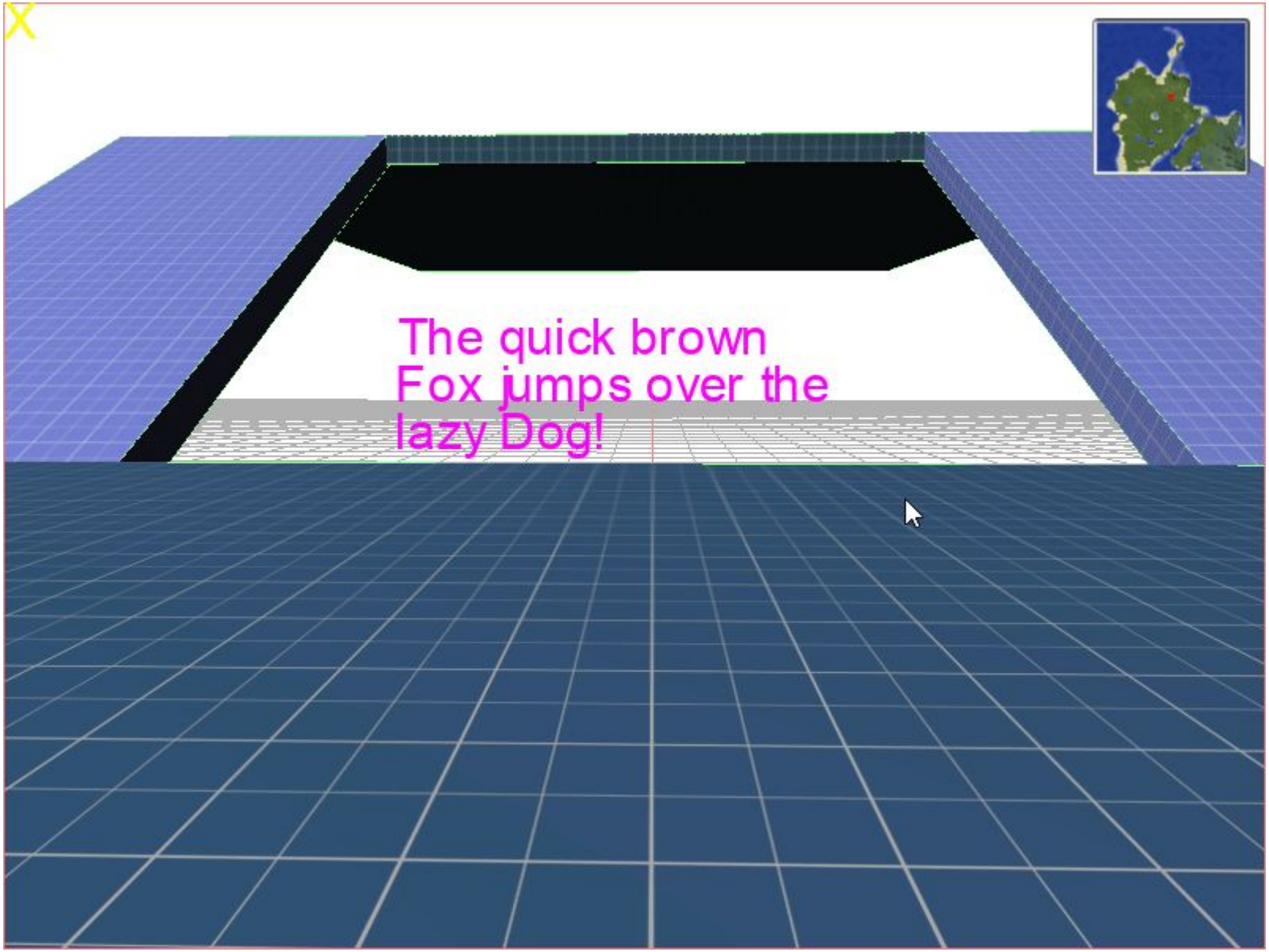
MVD: Engine Programming

08 - GUI

alunthomas.evans@salle.url.edu

laSalle ENG
Universitat Ramon Llull


The quick brown
Fox jumps over the
lazy Dog!

GUI

What do you think we need for a GUI system?

How do we draw it?

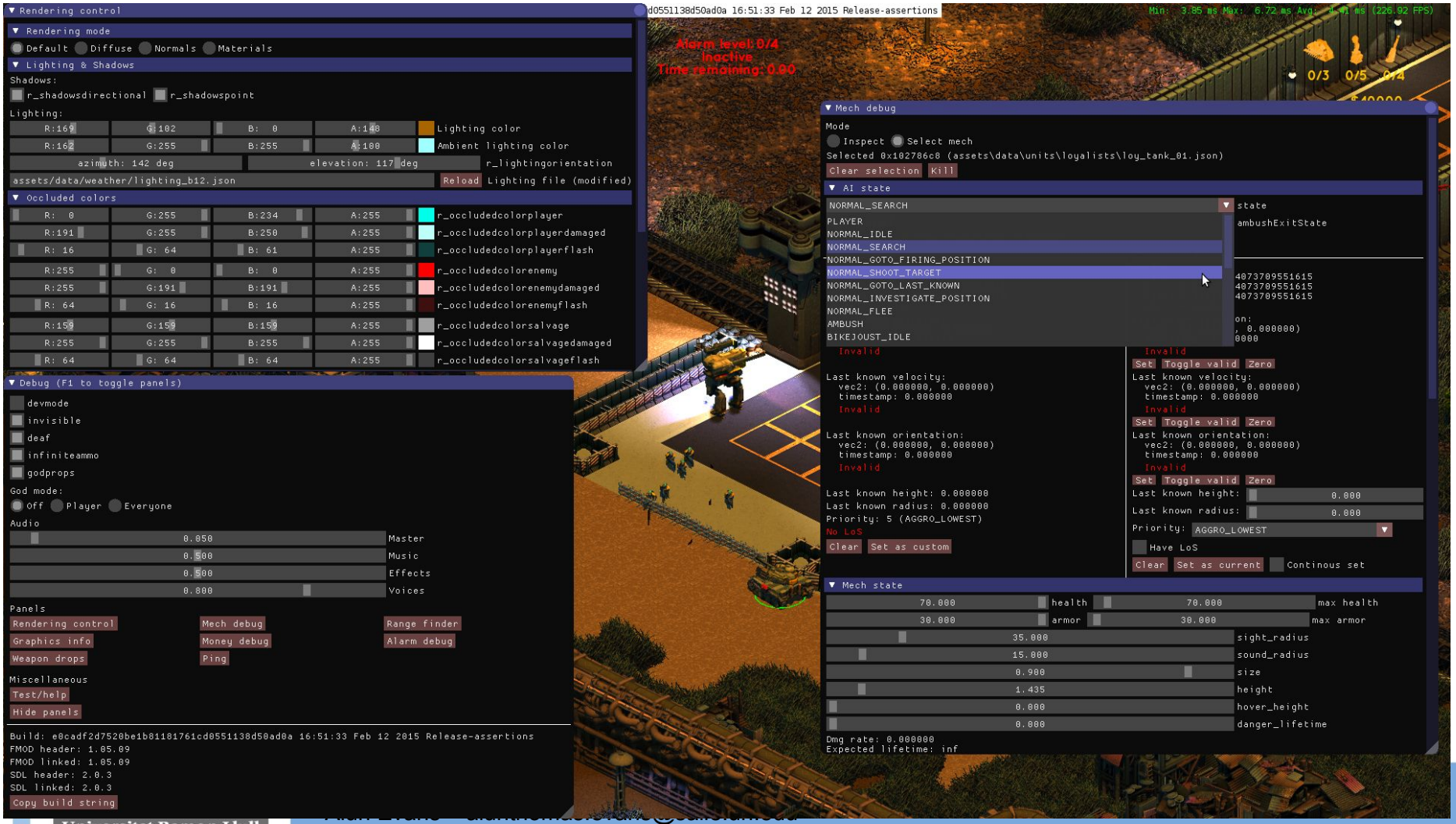
GUIs can be very complicated!

Top 3 things to tackle:

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

GUI Content - Game

GUI Content - Debugging



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

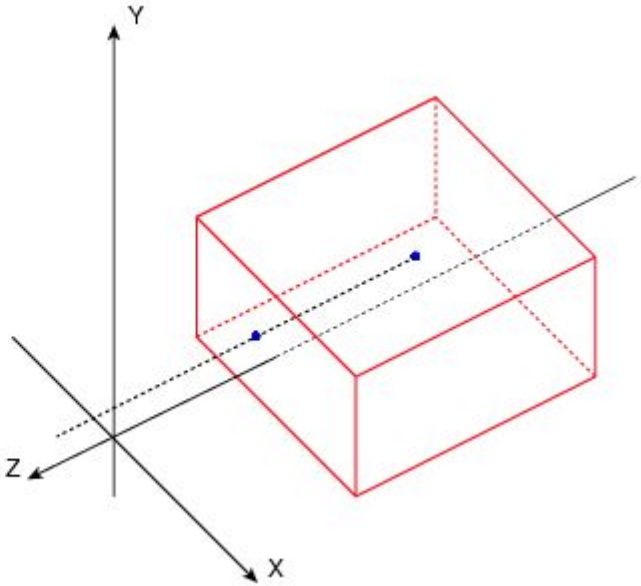
Cons:

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

Rendering

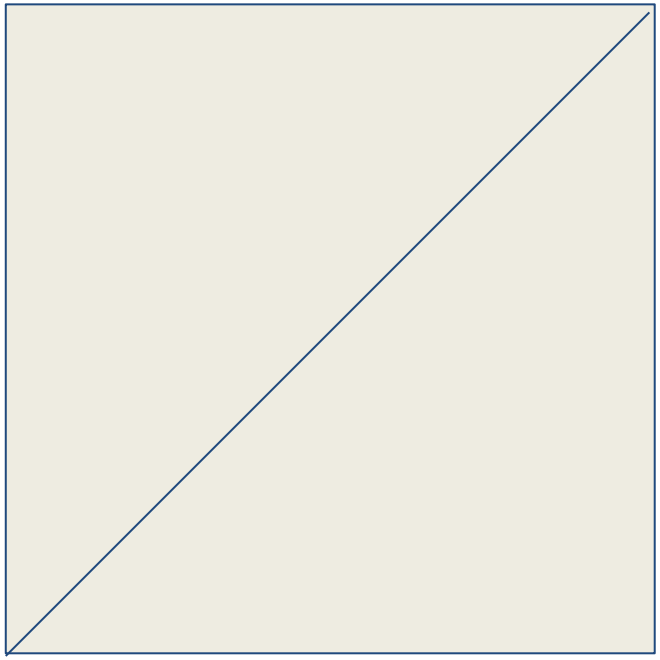
Create orthographic camera the same size as window, centered around origin

```
//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 -> +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

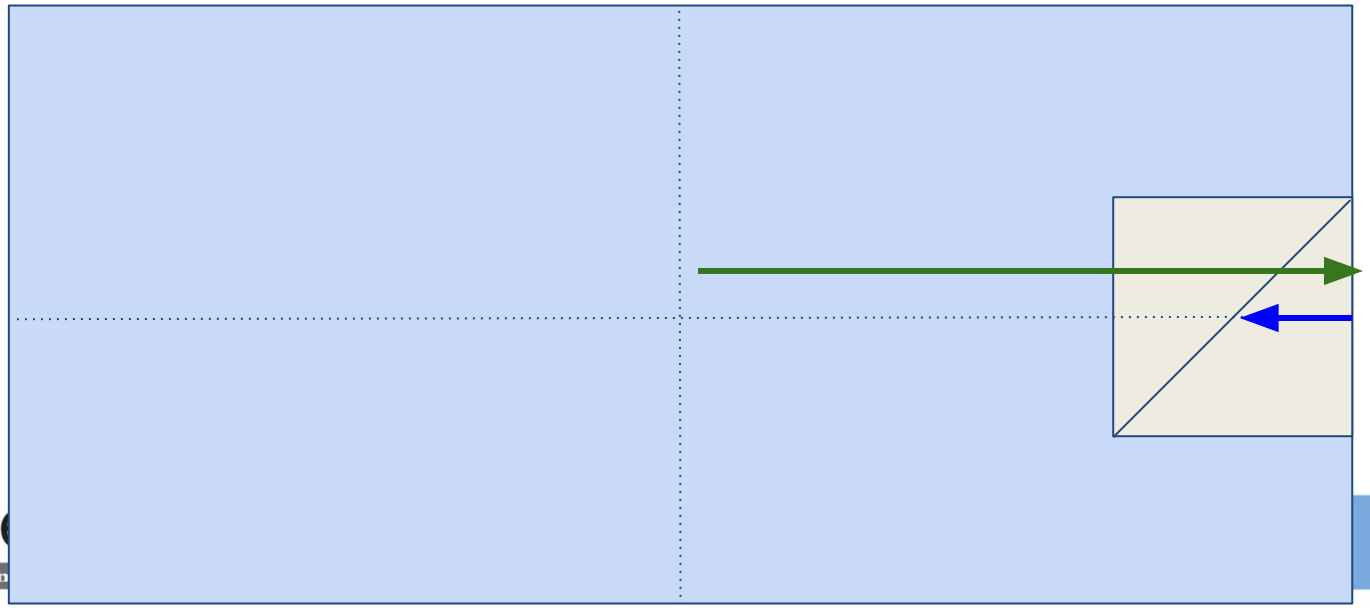
```
model.makeScaleMatrix(
   el.width / 2,
   el.height / 2,
   1.0f);
```

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;
};
```

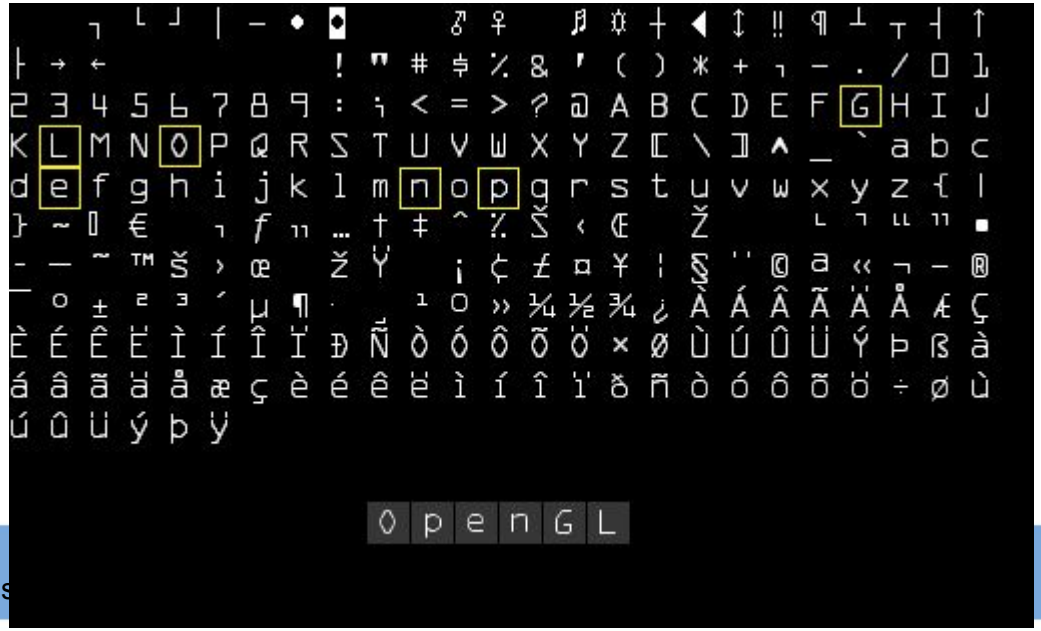
Task

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?

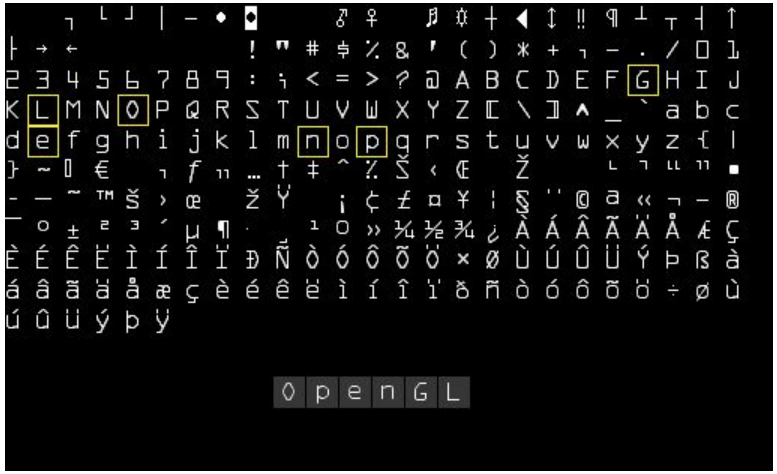
Font glyph textures



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

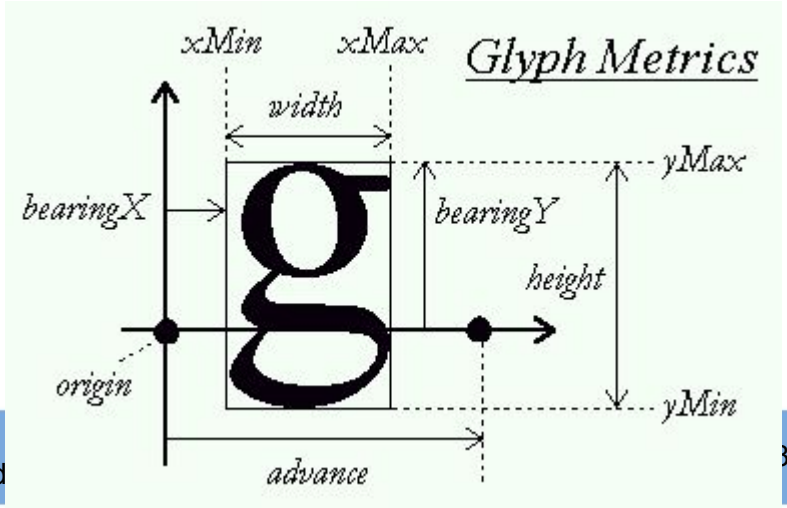
Non-fixed-width fonts need extra metadata

In engine text rendering

Freetype - Open source font rendering engine, written in C

Loads fonts from ttf files

Load ‘glyph’ for invididual 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

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:

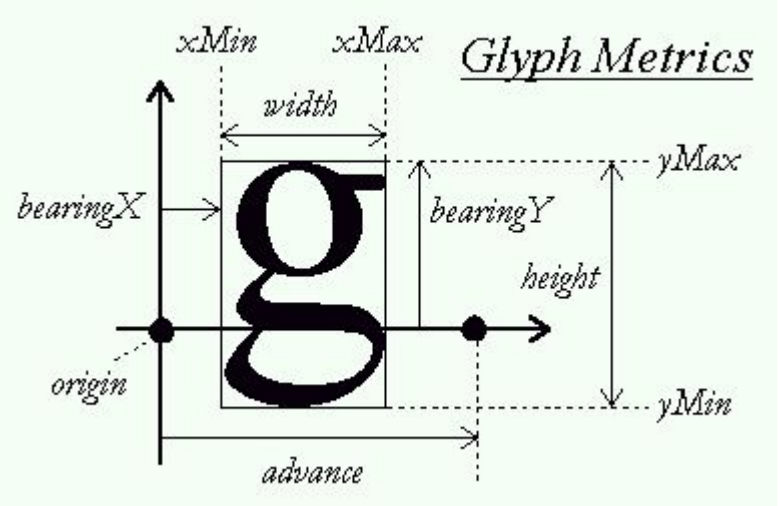
```
std::vector<unsigned char> empty_data(tex_width * tex_height, 0);

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!

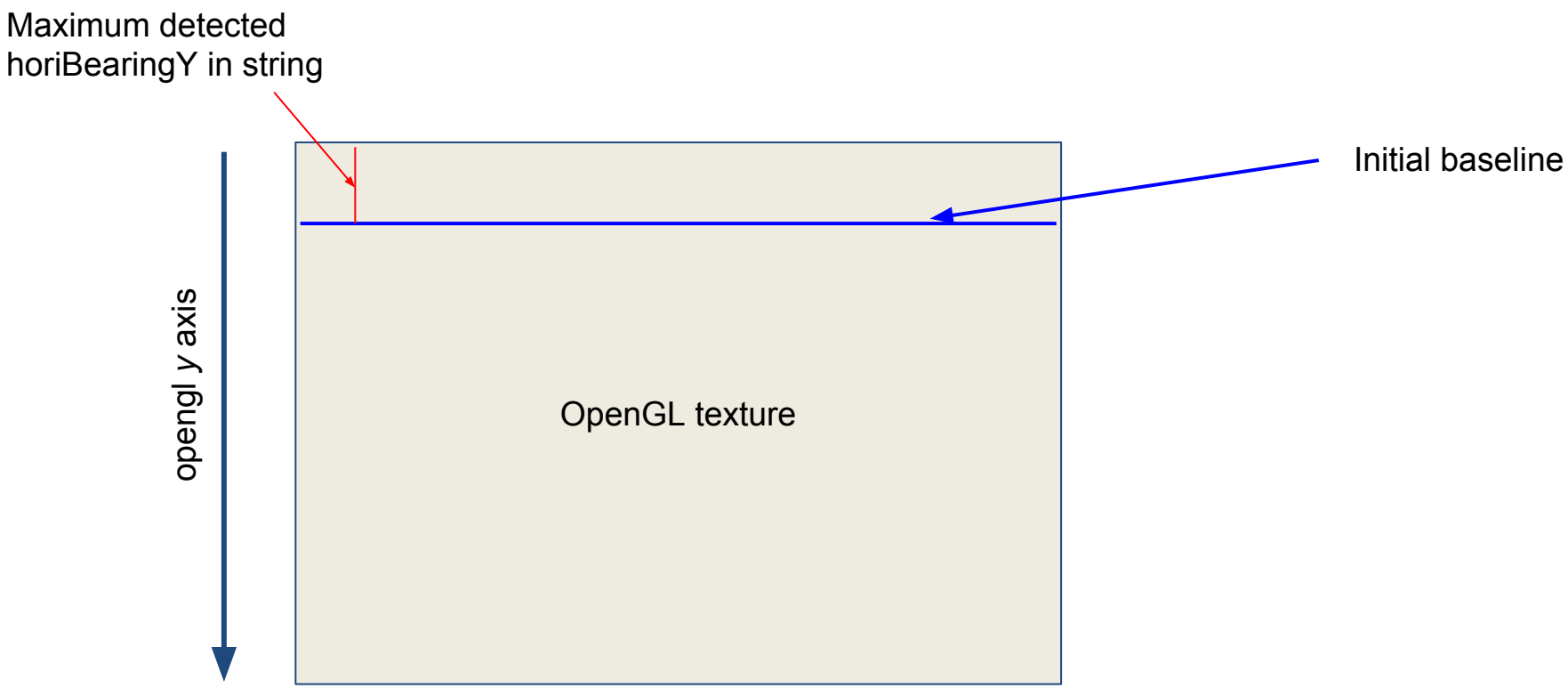
ALSO

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)



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