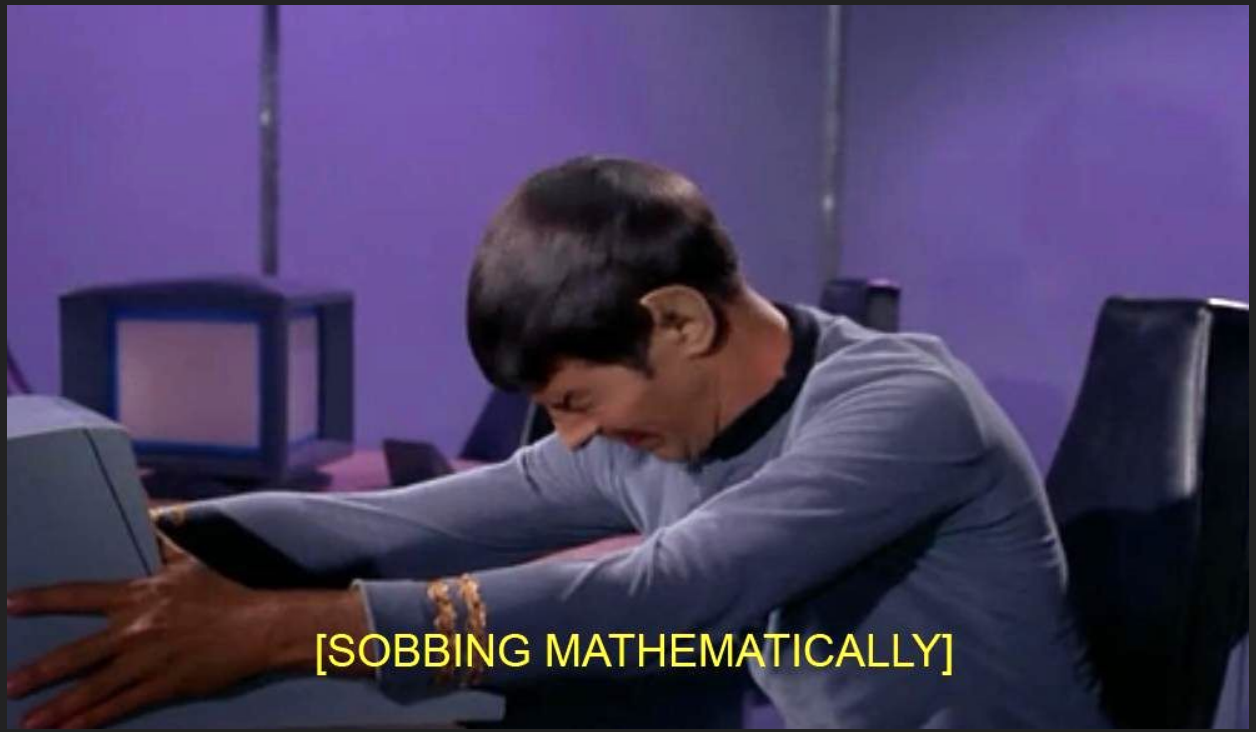
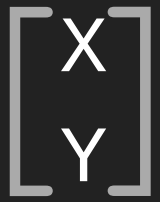
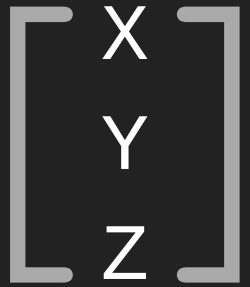
More Math!



Let’s Review...

We can represent Vectors as a matrix.

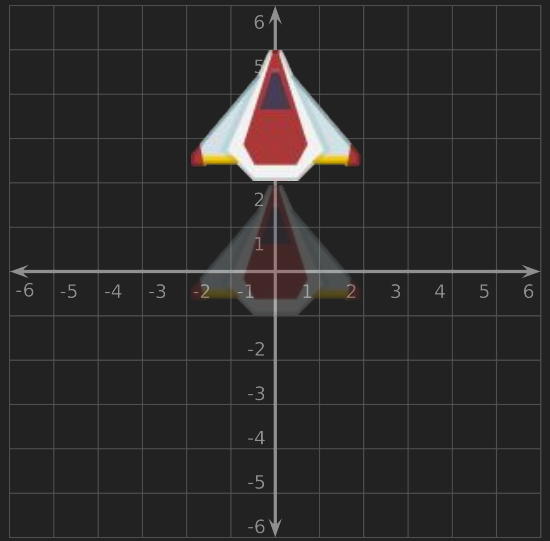




We use
matrix multiplication
to perform transformations.

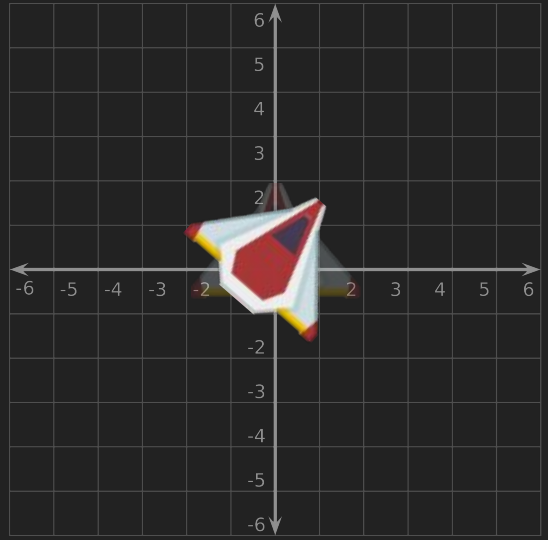
Multiplying by the Identity Matrix has no effect

Translation



$$\begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

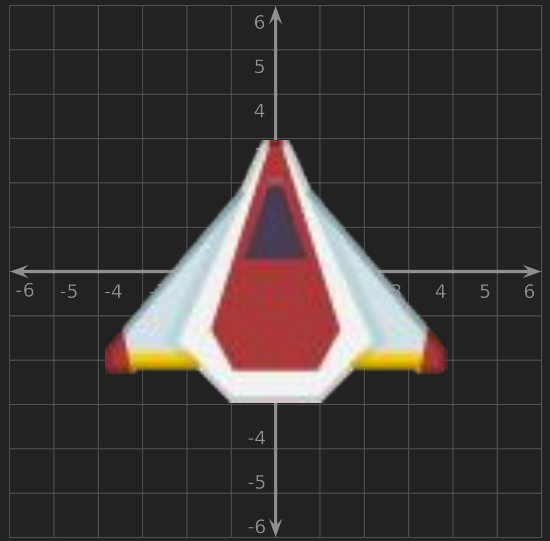
Rotation



$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(this is for Z rotate, it's a little different for X and Y)

Scale



$$\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The vertex shader applies the matrix we provide to every vertex in our model.

```
modelMatrix = glm::mat4(1.0f);

modelMatrix = glm::translate(modelMatrix, glm::vec3(3.0f, 2.0f, 0.0f));

program.SetModelMatrix(modelMatrix);
```

New Stuff!

Multiplying matrices combines their transformations.

$$\begin{bmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 4 \\ 0 & 2 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

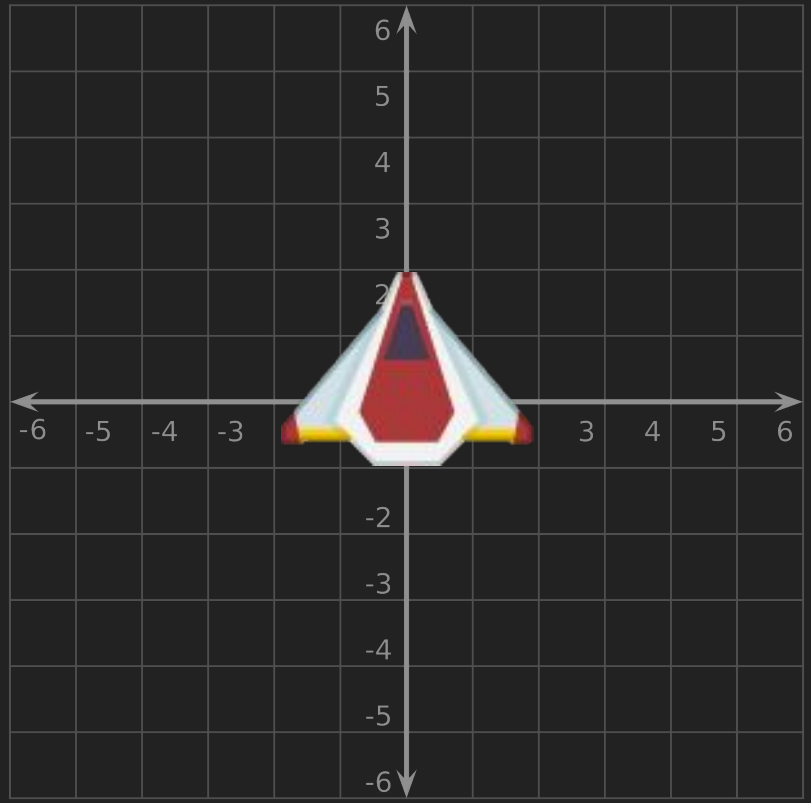
Matrix multiplication is not commutative!

(the order matters)

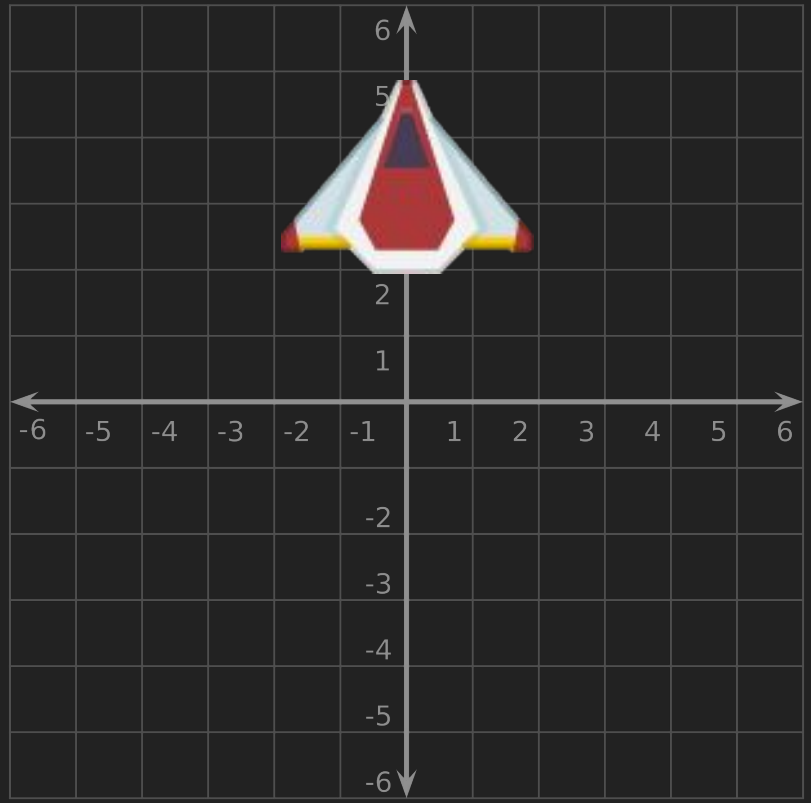
$$M = T * R$$

(translate then rotate)

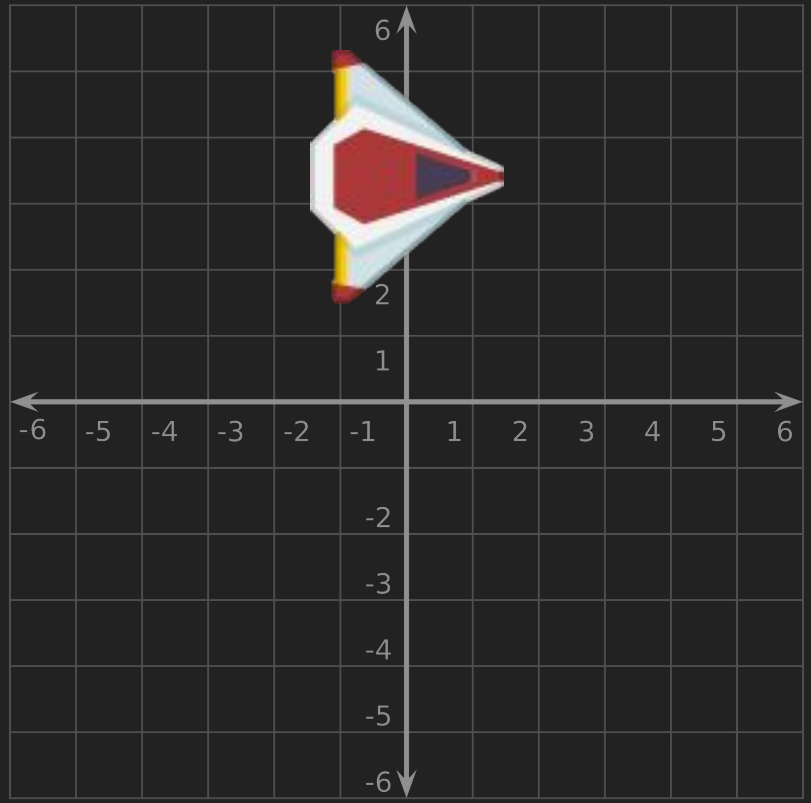
Identity



Translation



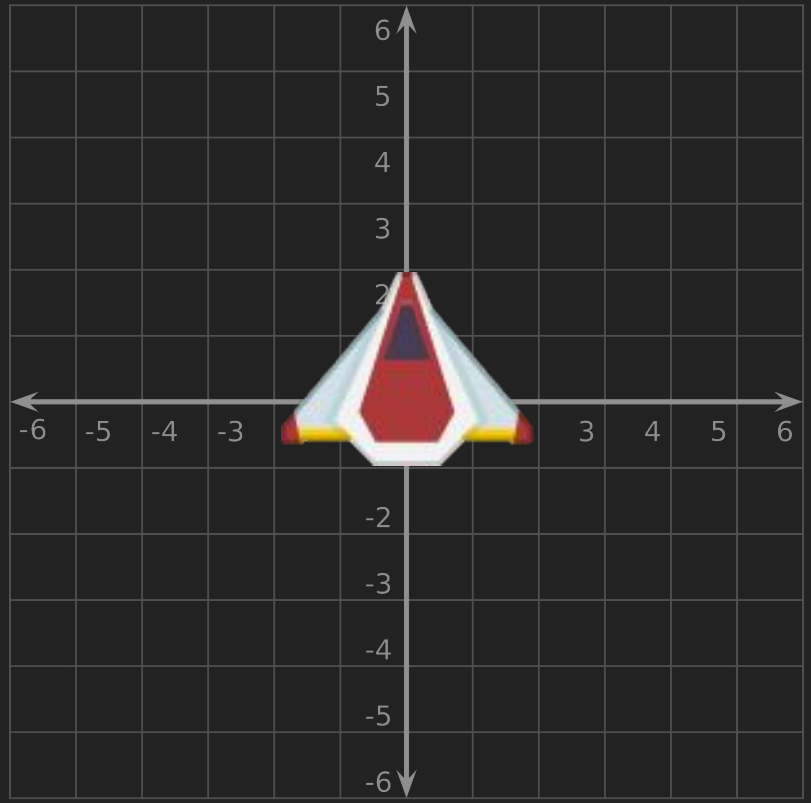
Rotation



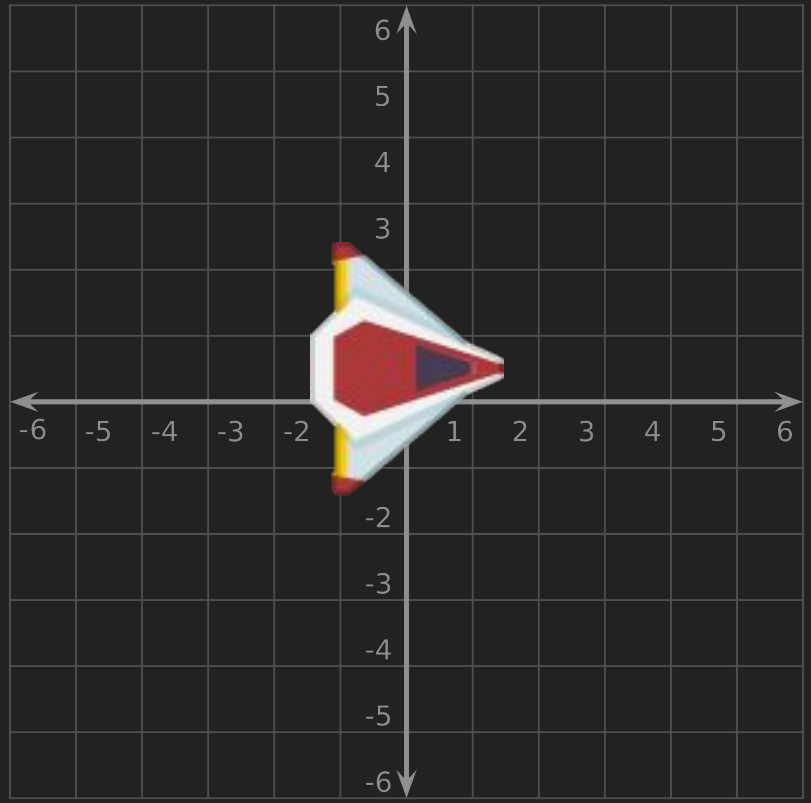
$$M = R * T$$

(rotate then translate)

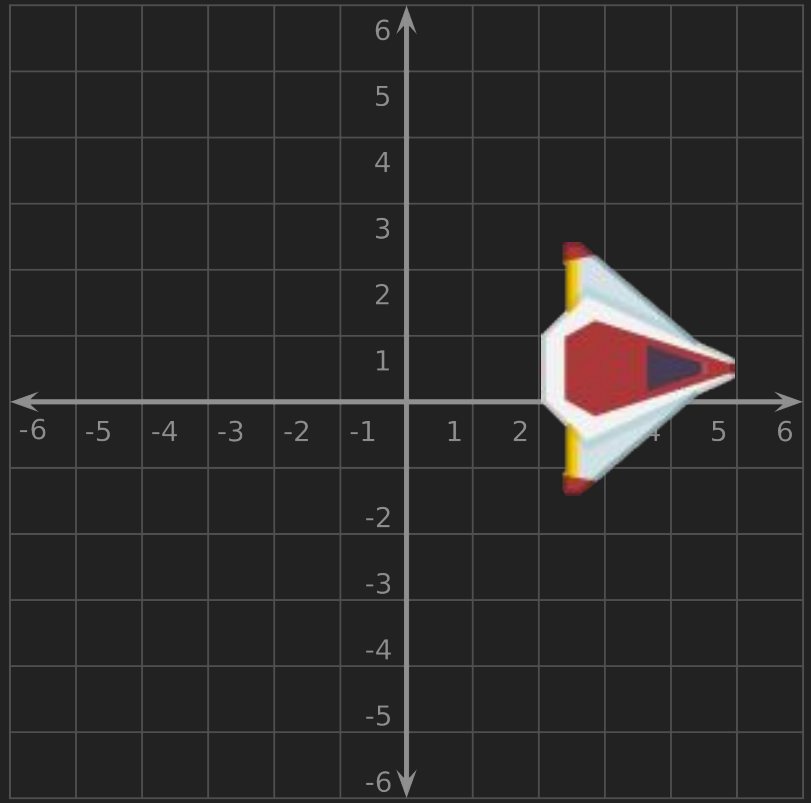
Identity



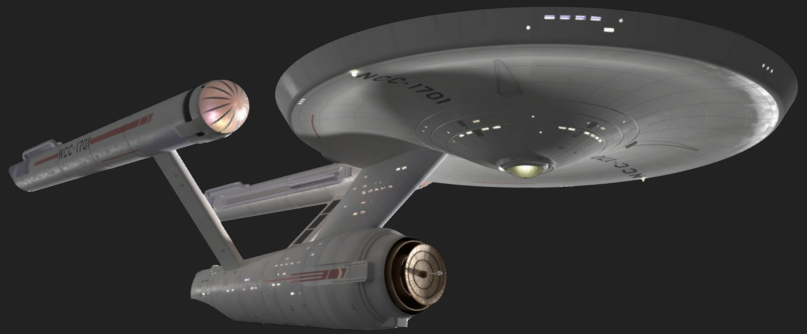
Rotation



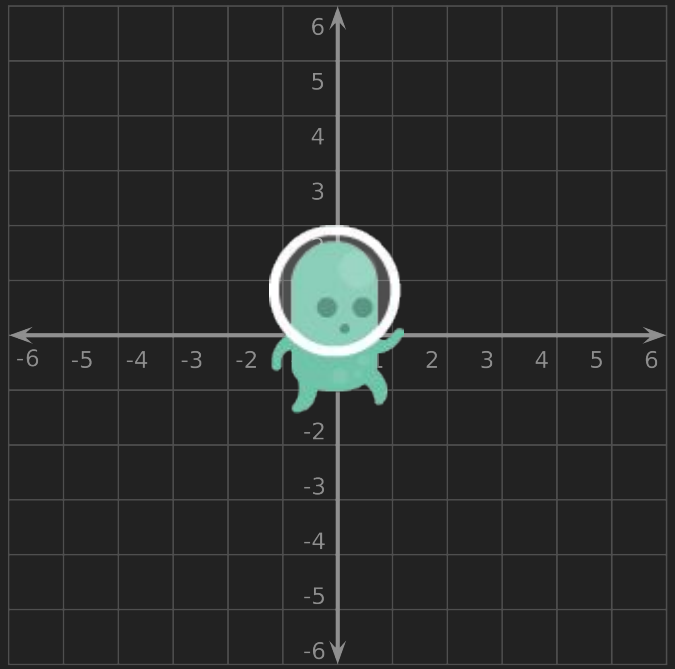
Translation

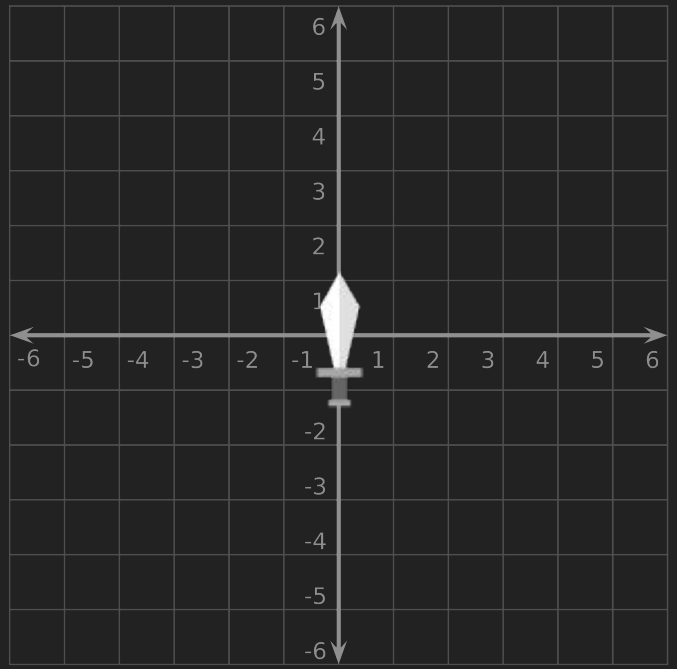


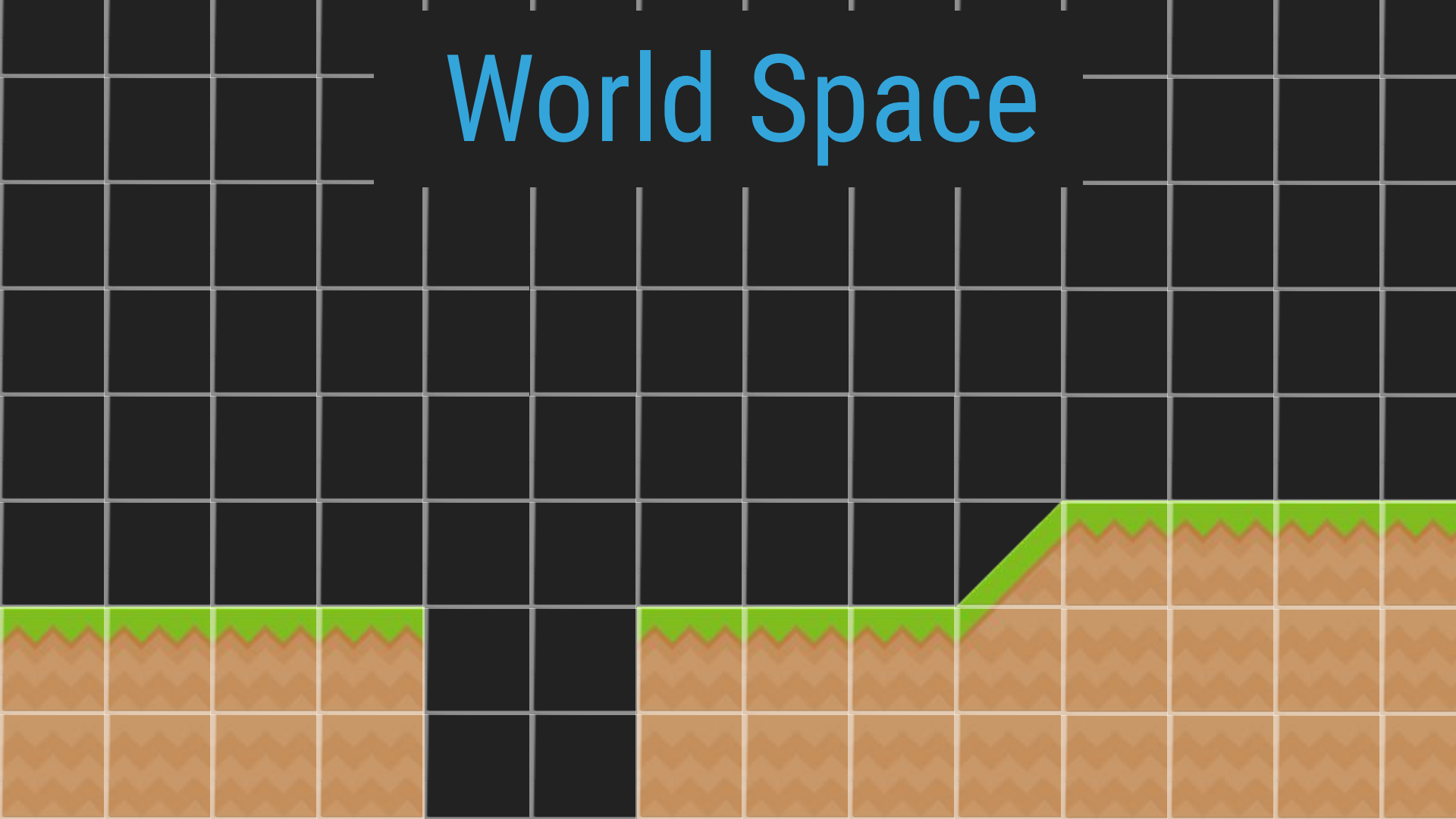
Spaces



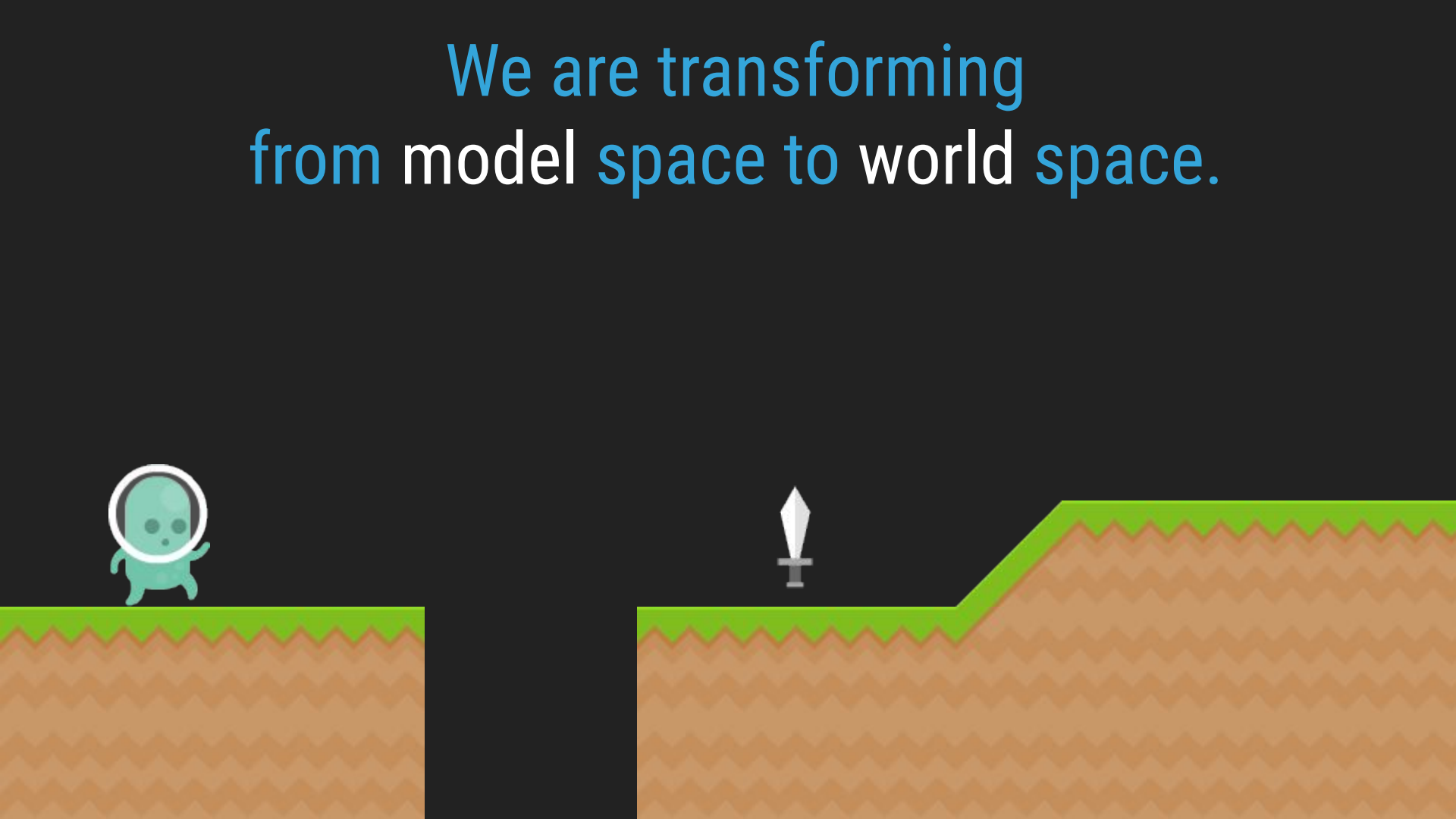
Model Space





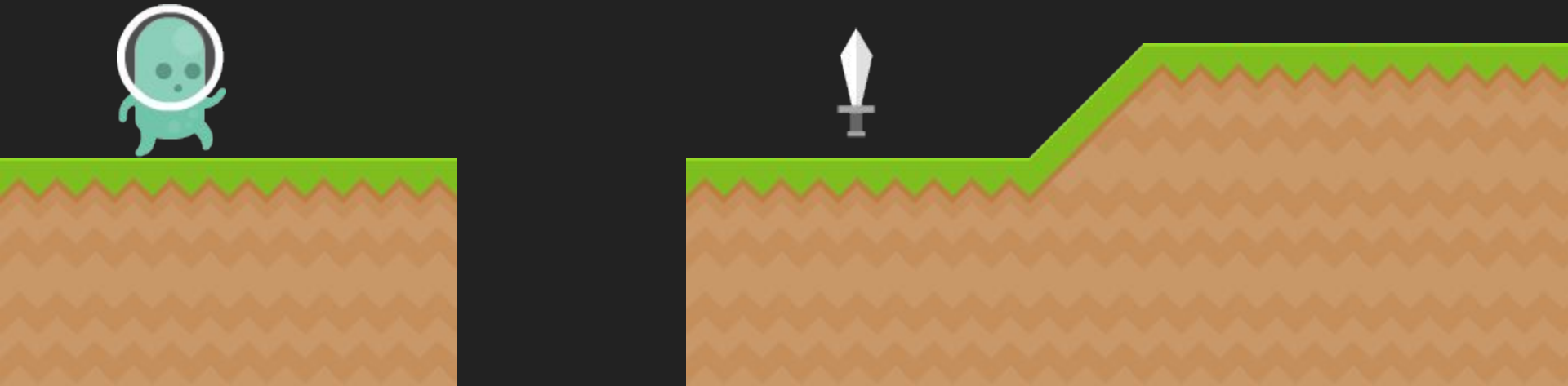

World Space

We are transforming
from model space to world space.



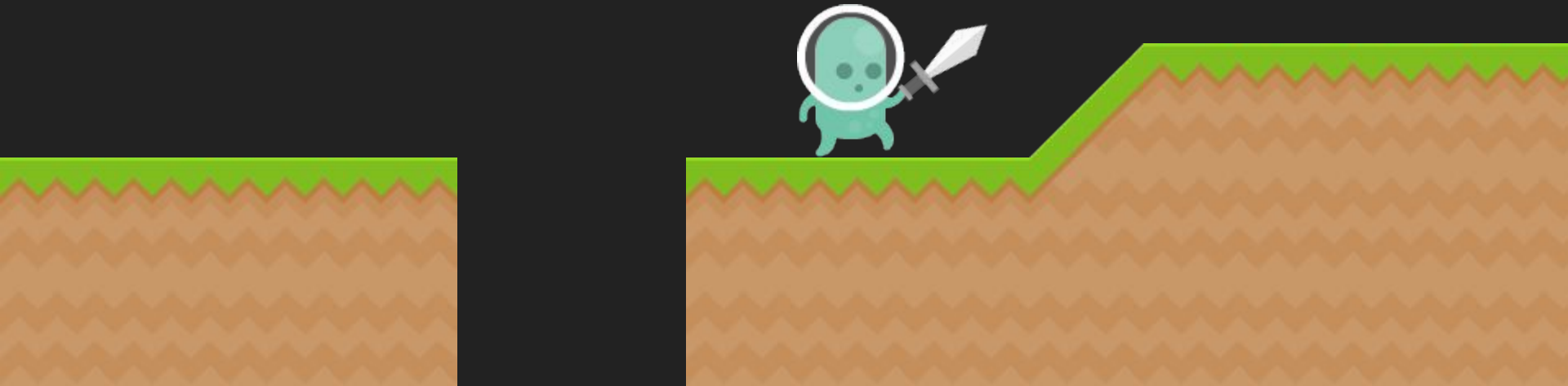
We are transforming from model space to world space.

```
playerMatrix = glm::mat4(1.0f);
playerMatrix = glm::translate(playerMatrix, glm::vec3(1.0f, 2.0f, 0.0f));

swordMatrix = glm::mat4(1.0f);
swordMatrix = glm::translate(swordMatrix, glm::vec3(7.0f, 2.0f, 0.0f));
```



You may need to make a hierarchy if something is relative to another object.

```
playerMatrix = glm::mat4(1.0f);
playerMatrix = glm::translate(playerMatrix, glm::vec3(7.0f, 2.0f, 0.0f));

swordMatrix = glm::translate(playerMatrix, glm::vec3(0.5f, 0.0f, 0.0f));
swordMatrix = glm::rotate(swordMatrix, 0.78f, glm::vec3(0.0f, 0.0f, 1.0f));
```



Our games are not static scenes,
things need to
translate, rotate and scale over time.

We could initialize the model matrix and then change the matrix every frame.

(but this could get weird)

```
void Initialize() {
    playerMatrix = glm::mat4(1.0f);
}

void Update() {
    playerMatrix = glm::translate(playerMatrix, glm::vec3(0.1f, 0.0f, 0.0f));
}
```

Instead, keep track of position, rotation and scale in variables and setup the matrix as needed.

```
void Initialize() {
    player_x = 0.0f;
    player_y = 0.0f;
}

void Update() {
    playerMatrix = glm::mat4(1.0f);
    playerMatrix = glm::translate(playerMatrix,
                                  glm::vec3(player_x, player_y, 0.0f));
}
```

Timing and FPS

Things should happen in our games at the same speeds regardless of how fast or slow the user's hardware is.

Faster hardware does more updates than slower hardware.

```
void Update() {
    player_x += 0.1f;
}

void Render() {
    playerMatrix = glm::mat4(1.0f);
    playerMatrix = glm::translate(playerMatrix,
                                  glm::vec3(player_x, player_y, 0.0f));
}
```

60 FPS

30 FPS

We can calculate the time since the last frame.

```
float lastTicks = 0.0f;

void Update() {
    float ticks = (float)SDL_GetTicks() / 1000.0f;
    float deltaTime = ticks - lastTicks;
    lastTicks = ticks;

    player_x += 1.0f * deltaTime;
}
```

deltatime

deltaTime values on different computers:

60 FPS: 16.66ms / 1000 = 0.0166

30 FPS: 33.33ms / 1000 = 0.0333

```
// Travel 1 unit per second
player_x += 1.0f * deltaTime;
```

60 FPS

30 FPS

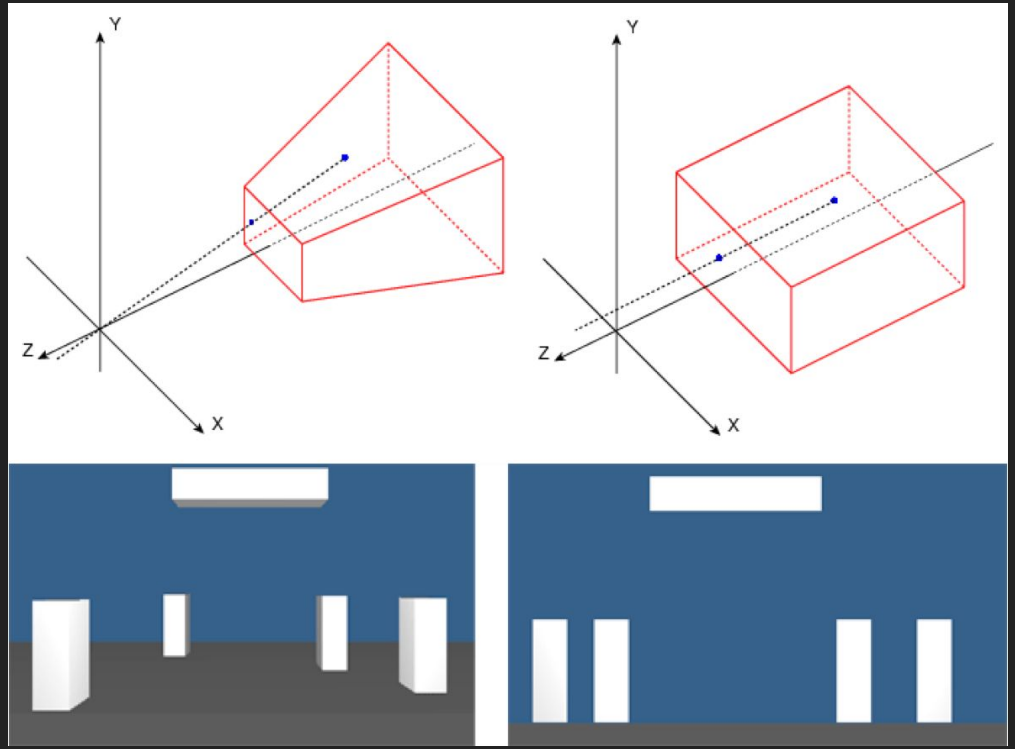
We can use timing to handle rotation as well.

```
// Rotate 45 degrees per second
rotate_z += 45.0f * deltaTime;


modelMatrix = glm::rotate(modelMatrix,
                          glm::radians(rotate_z),
                          glm::vec3(0.0f, 0.0f, 1.0f));
```

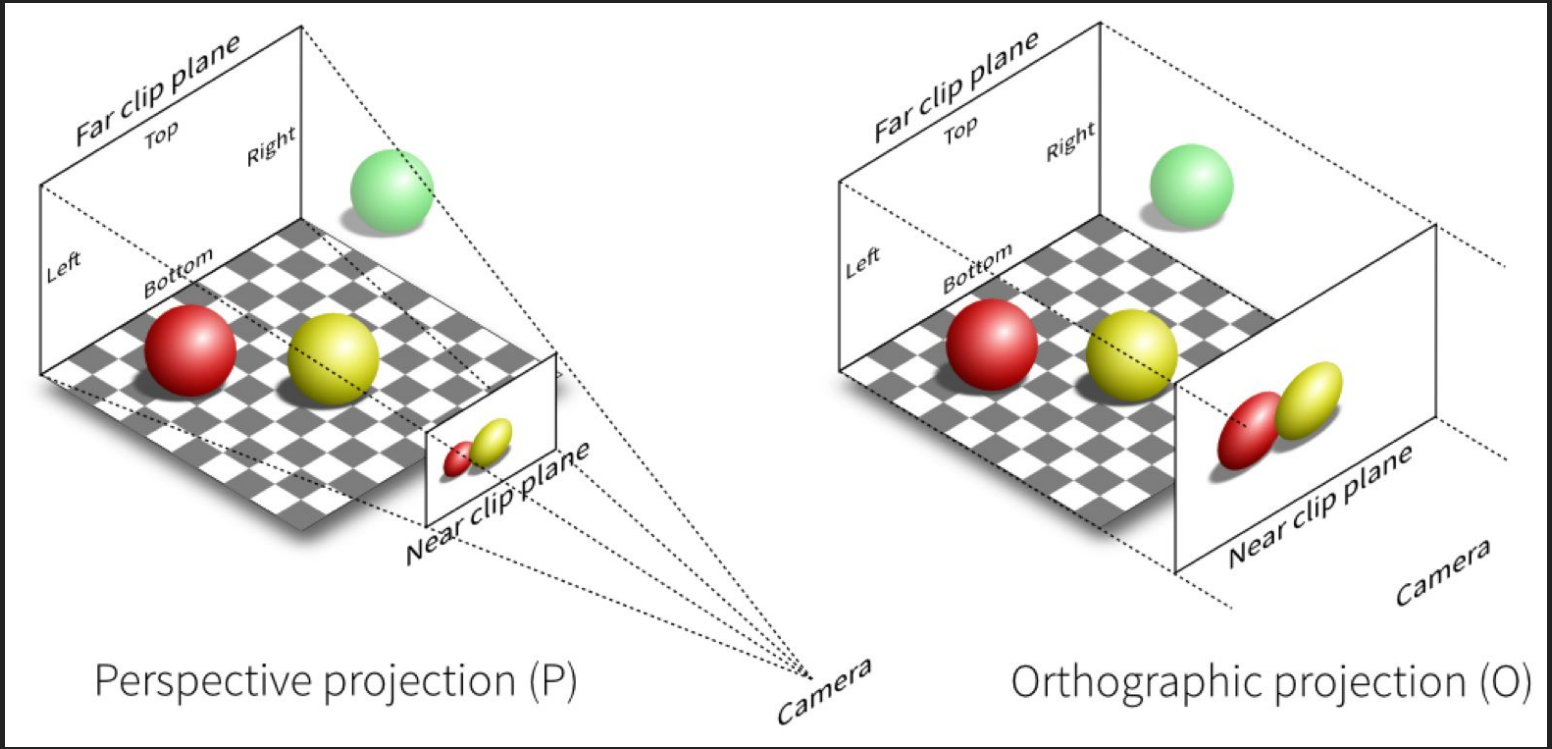
You may have noticed:

```
projectionMatrix = glm::ortho(-5.0f, 5.0f, -3.75f, 3.75f, -1.0f, 1.0f);
program.SetProjectionMatrix(projectionMatrix);
```

Perspective vs. Orthographic



Perspective vs. Orthographic



Perspective vs. Orthographic

```
projectionMatrix = glm::ortho(-5.0f, 5.0f, -3.75f, 3.75f, -1.0f, 1.0f);
program.SetProjectionMatrix(projectionMatrix);
```

Window Resolution: 640 / 480 = 1.3333

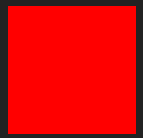
Orthographic View: 10 / 7.5 = 1.3333

Let’s Code!

Color and Textures

Instead of values 0 - 255 or #00 - #ff,
OpenGL colors have 3 or 4 channels ranging from
0.0 to 1.0 (floating point).

RGB



```
1.0, 0.0, 0.0
```

```
1.0, 0.5, 0.0
```

RGBA

```
1.0, 0.0, 0.0, 0.75
```

```
1.0, 0.5, 0.0, 0.2
```

Start with a clear screen each frame.

glClearColor

Sets the color to use when clearing the screen.

```
glClearColor(float red, float green, float blue, float alpha);
```

```
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
```

glClear

Clears the screen using the color last set by glClearColor.

```
void Initialize() {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
}

void Render() {
    glClear(GL_COLOR_BUFFER_BIT);
}
```

Experiment!

Need a blue sky? Dark cave? Desert?

You can use clear color!

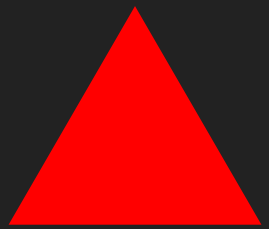
Setting a solid color of an untextured polygon.

ShaderProgram::SetColor

Sets the color to use when drawing a polygon.

```
ShaderProgram::SetColor(float red, float green, float blue,
                                                  float alpha);
```

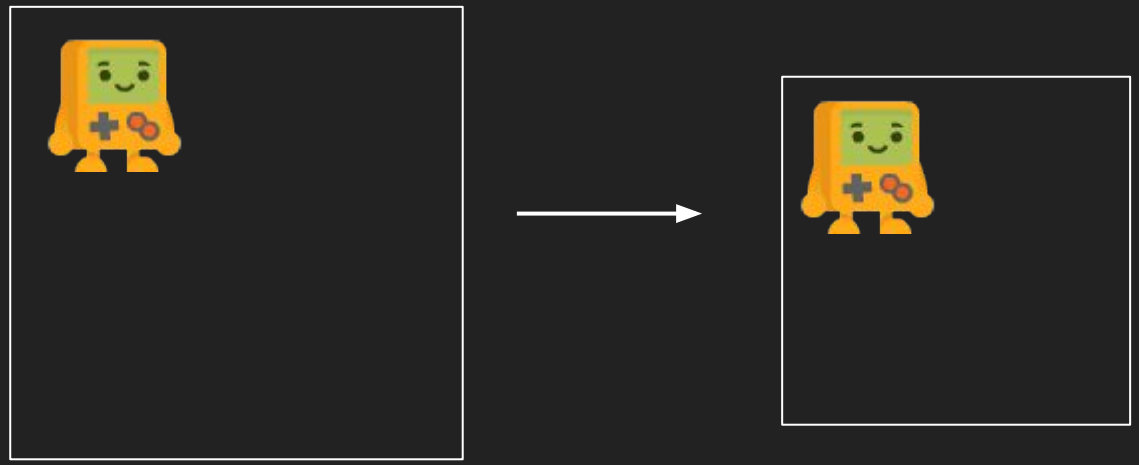
```
program.SetColor(1.0f, 0.0f, 0.0f, 1.0f);
```



Loading and Preparing Textures/Images

You do this during your setup, not every frame!

OpenGL Textures



RAM

Video Card RAM

Loading an image with STB_image

You must include STB_IMAGE_IMPLEMENTATION in one of the files you are including it from!

```
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
```

Use stbi_load to load the pixel data from an image file.

```
int w, h, n;
unsigned char* image = stbi_load("pacman.png",
                                 &w, &h, &n, STBI_rgb_alpha);
```

After you are done loading the image data, you must free it.

```
stbi_image_free(image);
```

Create a texture in OpenGL

```
GLuint textureID;
glGenTextures(1, &textureID);
```

Binding a texture

```
glBindTexture(GL_TEXTURE_2D, textureID);


// GL_TEXTURE_2D is a “target”
// Next slide will make this make sense...
```

Setting the texture pixel data

This is what sends the image to the graphics card.

```
// Our images must be RGBA!

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
                                    GL_UNSIGNED_BYTE, image);
```

Texture Filtering

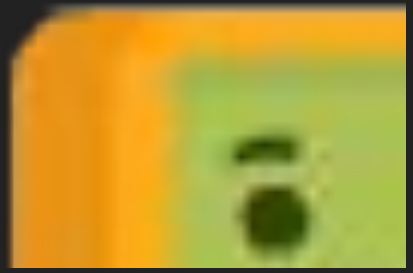
Original

Minification

Magnification

Texture Filtering

(try both and see what works better for your project)



Linear

Good for high resolution textures and textures with anti-aliasing.

Nearest neighbor

Good for pixel art.

Texture filtering settings.

```
// Use GL_LINEAR or GL_NEAREST
// MIN = Minifying, MAG = Magnifying

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

Let’s make that into a function.

```
GLuint LoadTexture(const char* filePath) {
    int w, h, n;
    unsigned char* image = stbi_load(filePath, &w, &h, &n, STBI_rgb_alpha);

    if (image == NULL) {
        std::cout << "Unable to load image. Make sure the path is correct\n";
        assert(false);
    }

    GLuint textureID;
    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_2D, textureID);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

    stbi_image_free(image);
    return textureID;
}
```

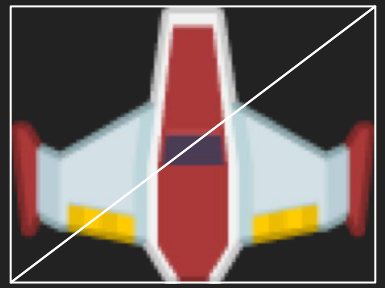
Now that the texture is loaded, we can apply it to our models as we draw each frame.

Texture Coordinates

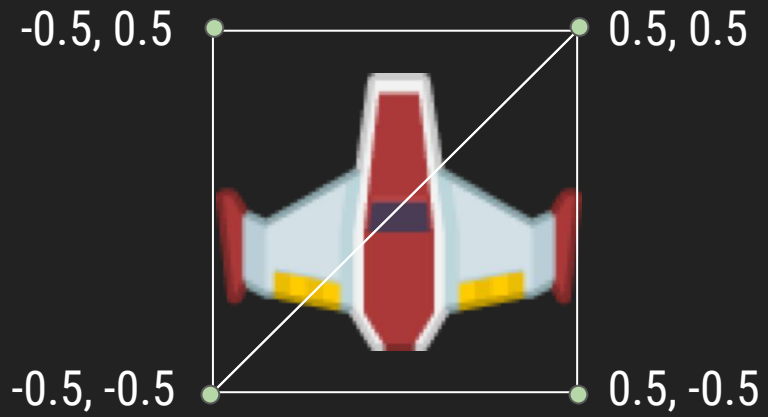
Texture coordinates are referred to as UV coordinates (X, Y and Z were already taken) :)

Notice the range from 0.0 to 1.0 and not by pixels.

2D Sprite Made of 2 Triangles



Need to match vertices to UV coordinates




```
float vertices[]  = {-0.5, -0.5, 0.5, -0.5, 0.5, 0.5, -0.5, -0.5, 0.5, 0.5, -0.5, 0.5};
float texCoords[] = {0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0};
```

How do we code that?

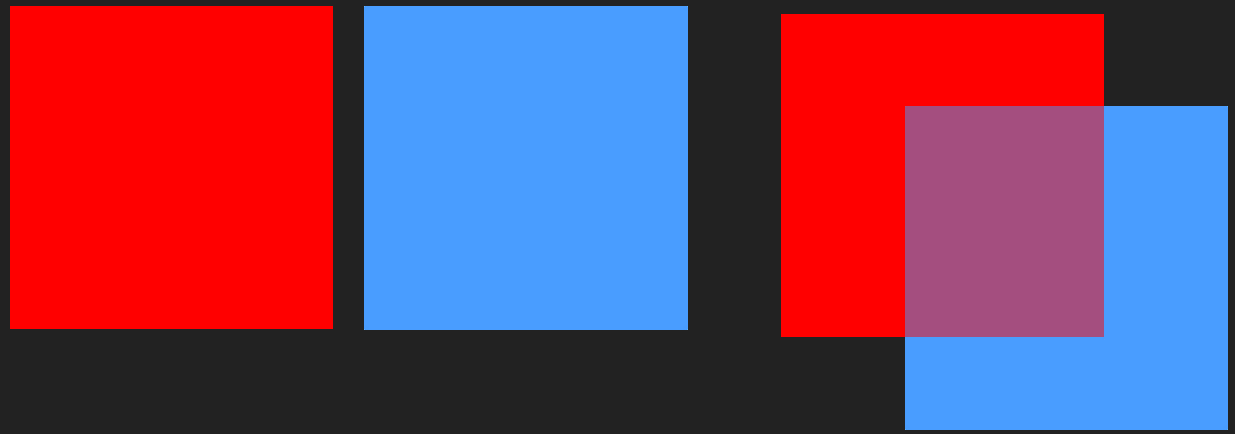
Initialization

```
GLuint playerTextureID;

void Initialize()
{

    // Load the shaders for handling textures!
    program.Load("shaders/vertex_textured.glsl",
                 "shaders/fragment_textured.glsl");

    // Load our player image
    playerTextureID = LoadTexture("player.png");
}
```

Rendering

```
void Render() {
    glClear(GL_COLOR_BUFFER_BIT);

    program.SetModelMatrix(modelMatrix);

    float vertices[]  = { -0.5, -0.5, 0.5, -0.5, 0.5, 0.5, -0.5, -0.5, 0.5, 0.5, -0.5, 0.5 };
    float texCoords[] = { 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0 };

    glVertexAttribPointer(program.positionAttribute, 2, GL_FLOAT, false, 0, vertices);
    glEnableVertexAttribArray(program.positionAttribute);
    glVertexAttribPointer(program.texCoordAttribute, 2, GL_FLOAT, false, 0, texCoords);
    glEnableVertexAttribArray(program.texCoordAttribute);

    glBindTexture(GL_TEXTURE_2D, playerTextureID);
    glDrawArrays(GL_TRIANGLES, 0, 6);

    glDisableVertexAttribArray(program.positionAttribute);
    glDisableVertexAttribArray(program.texCoordAttribute);

    SDL_GL_SwapWindow(displayWindow);
}
```

Blending



Blending

(blending is off by default, we need to enable it so our images are transparent)

```
glEnable(GL_BLEND);

// Good setting for transparency
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

https://learnopengl.com/Advanced-OpenGL/Blending

If your image does not load…

(In Xcode) Go to “Build Phases” and add your image to the Copy Files area. Make sure “Copy only when installing” is unchecked!

(Visual Studio) Use the file explorer to copy images into your project’s folder.