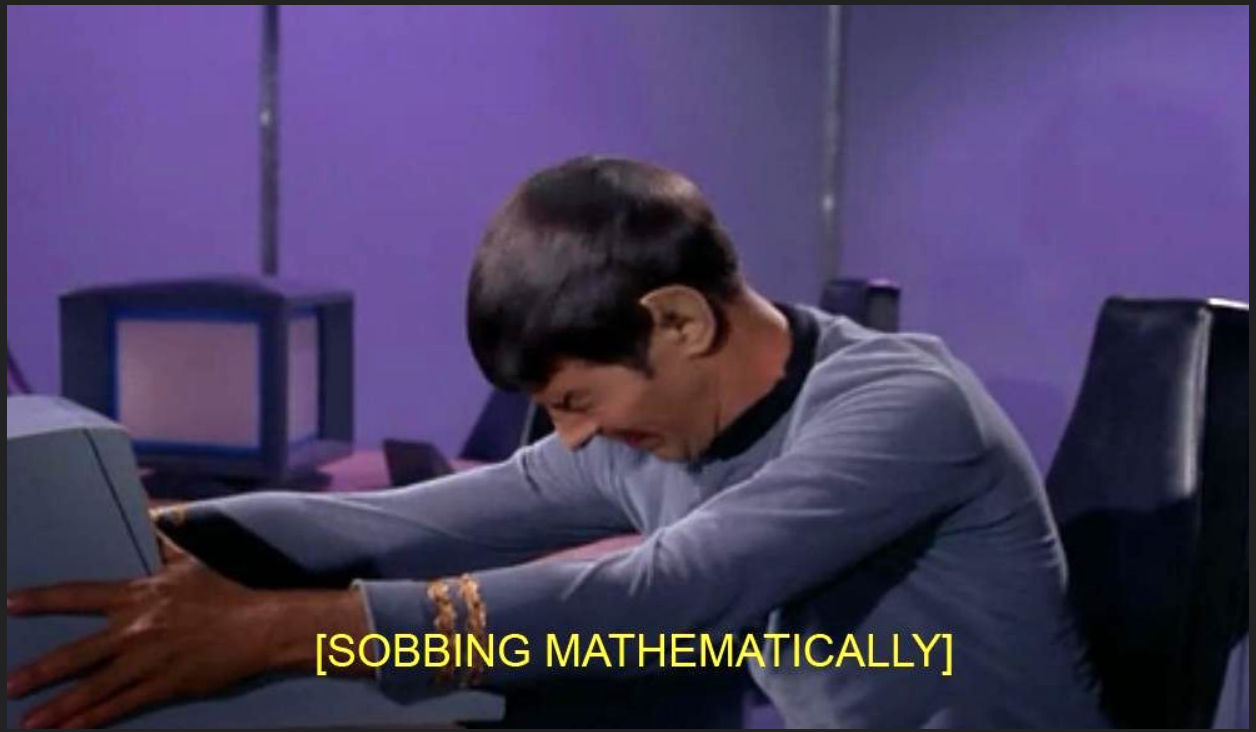
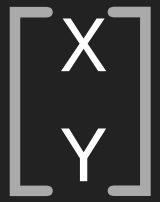
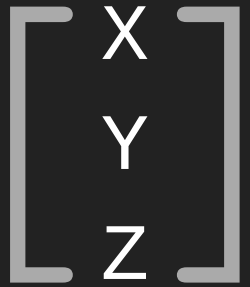
More Math!



Let’s Review...

We can represent Vectors as a matrix.

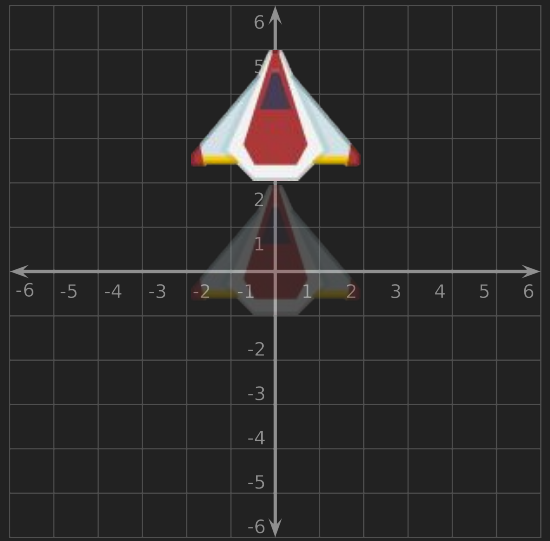




We use
matrix multiplication
to perform transformations.

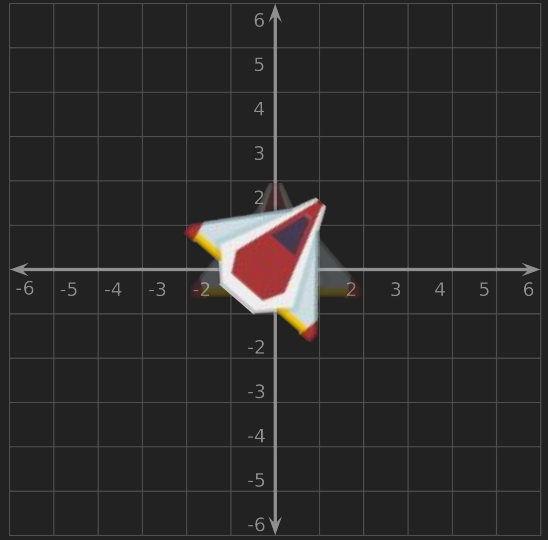
Multiplying by the Identity Matrix has no effect

Translation



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

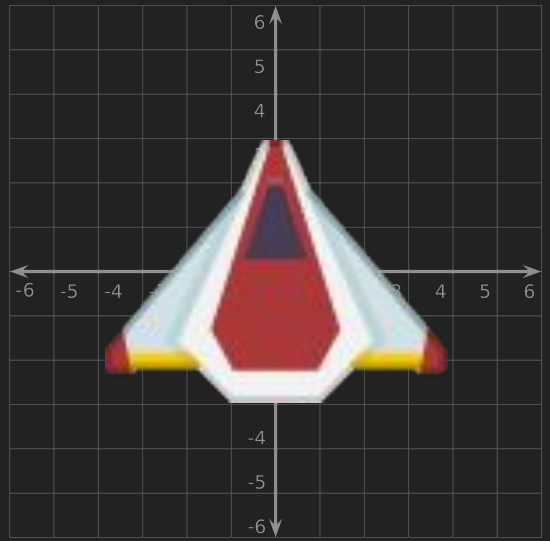
Rotation



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

(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.

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

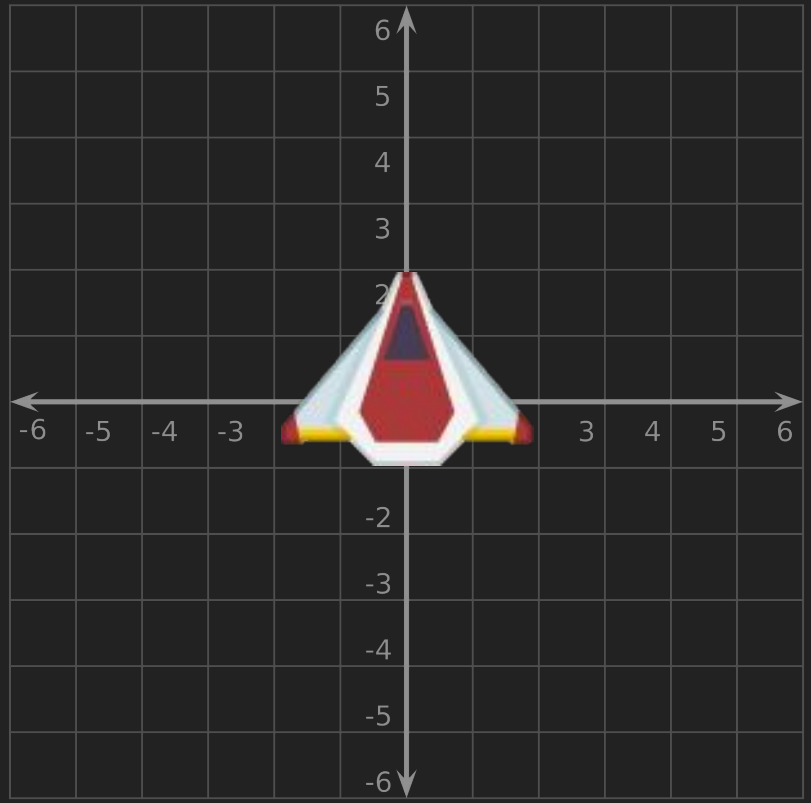
Matrix multiplication is not commutative!

(the order matters)

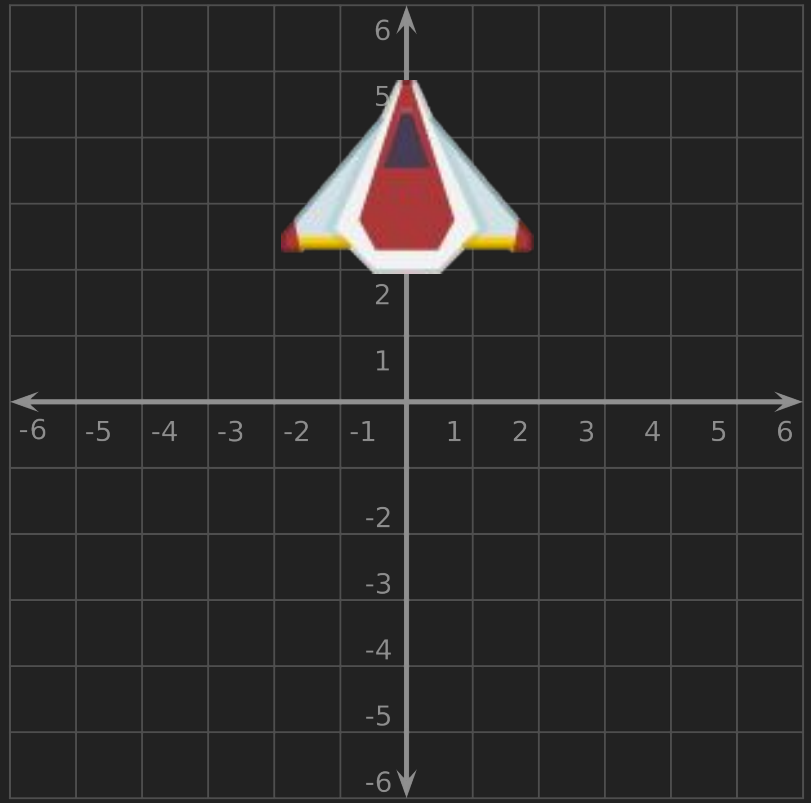
$$M = T * R$$

(translate then rotate)

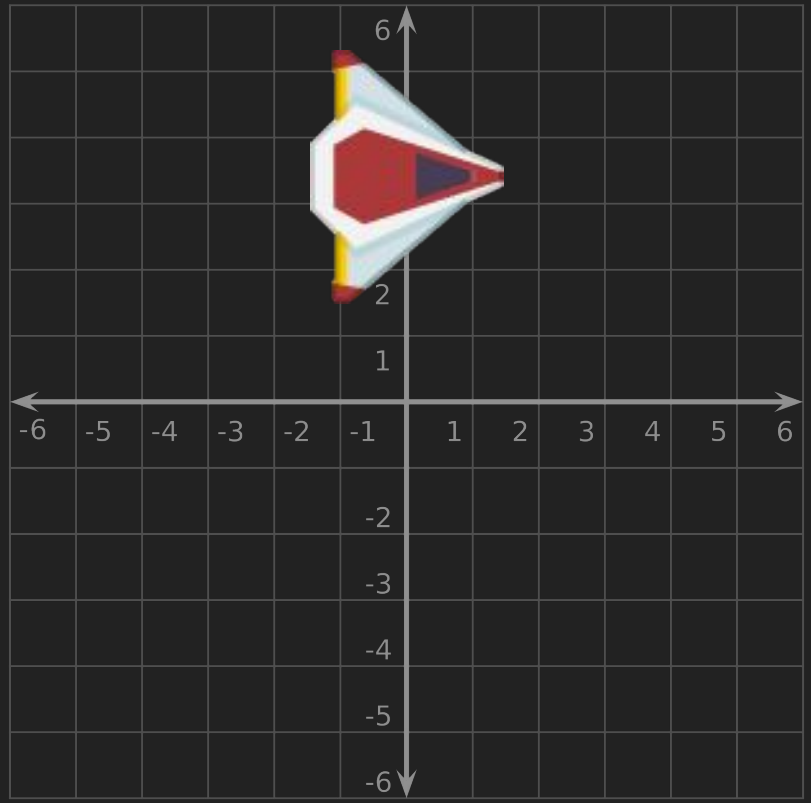
Identity



Translation



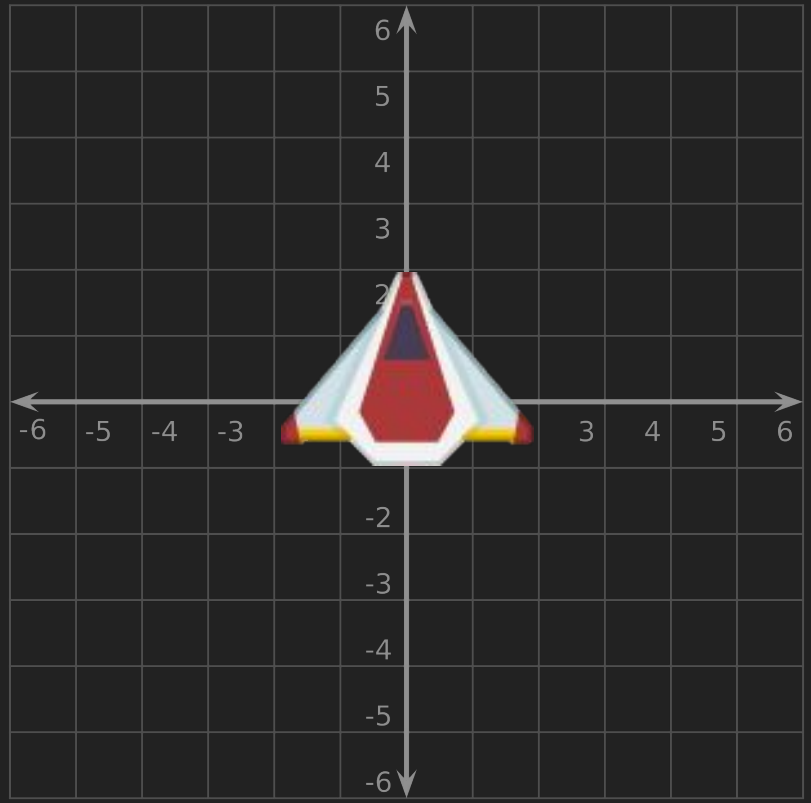
Rotation



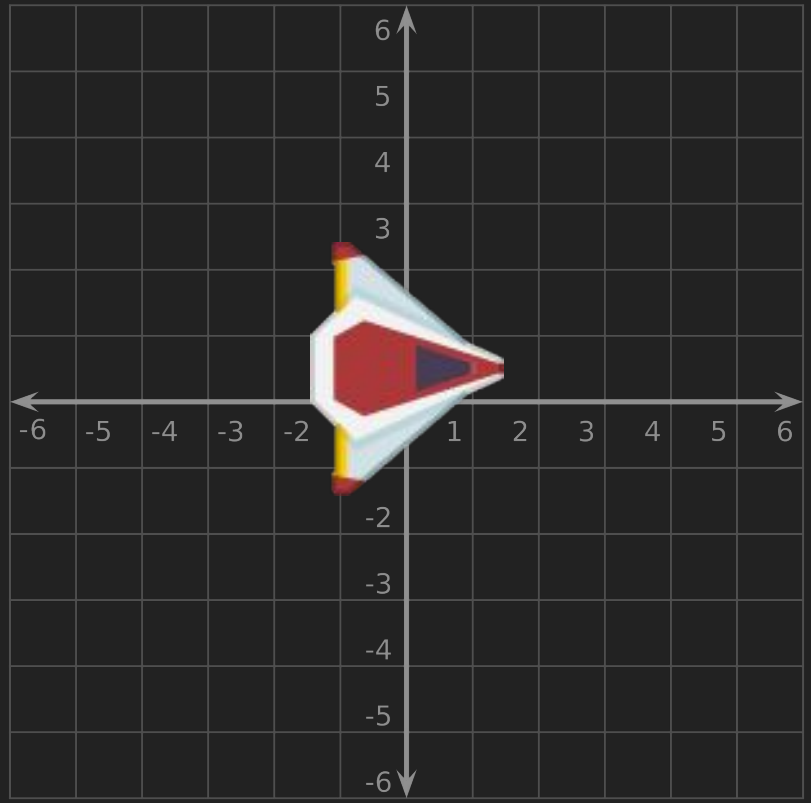
$$M = R * T$$

(rotate then translate)

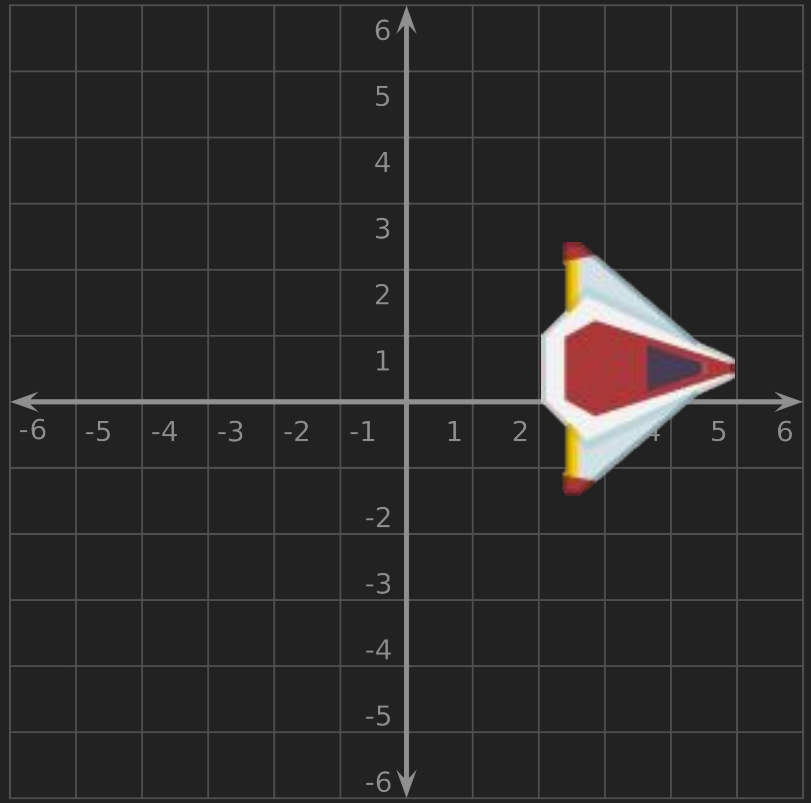
Identity



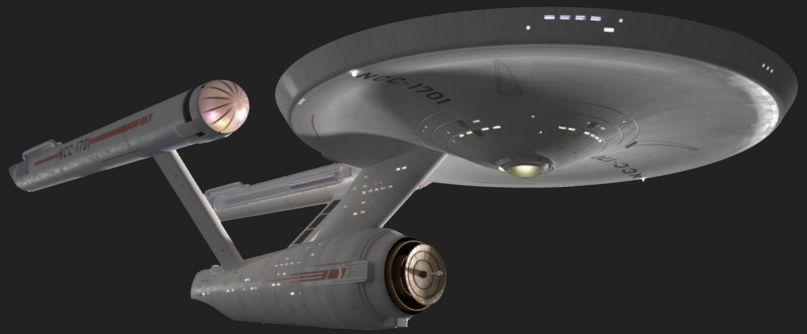
Rotation



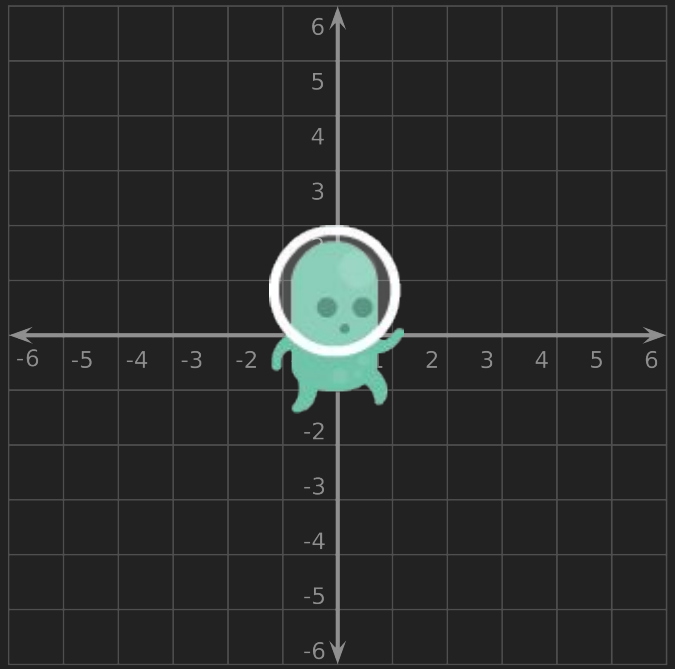
Translation

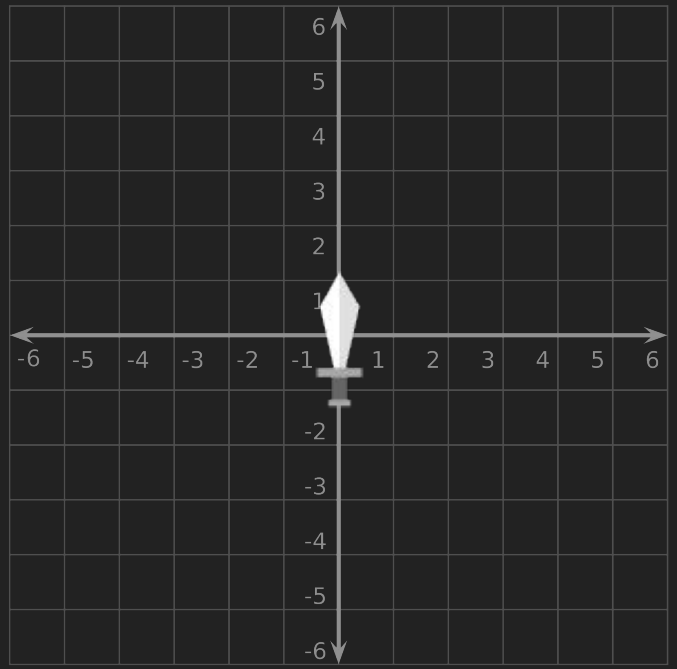


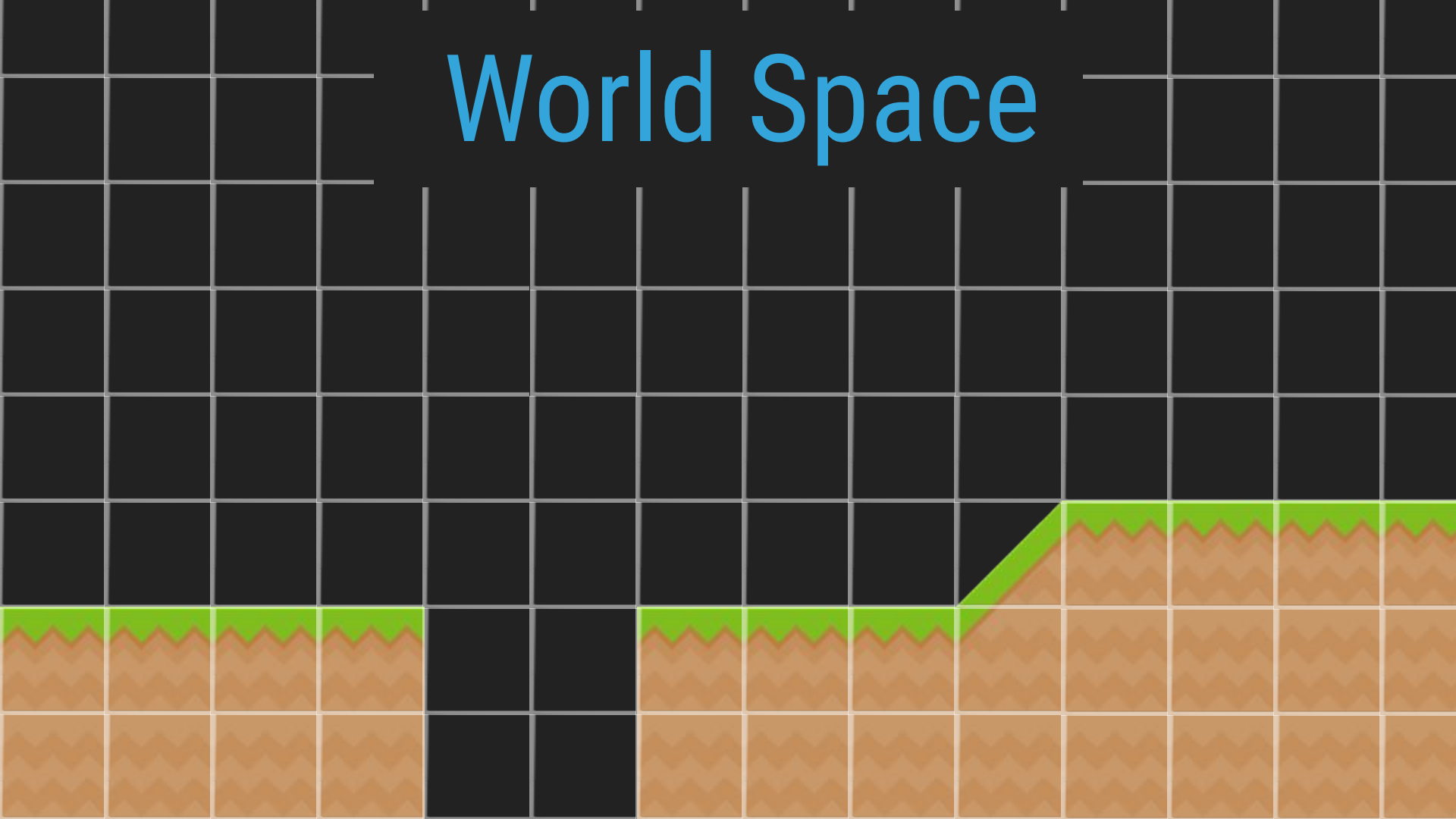
Spaces



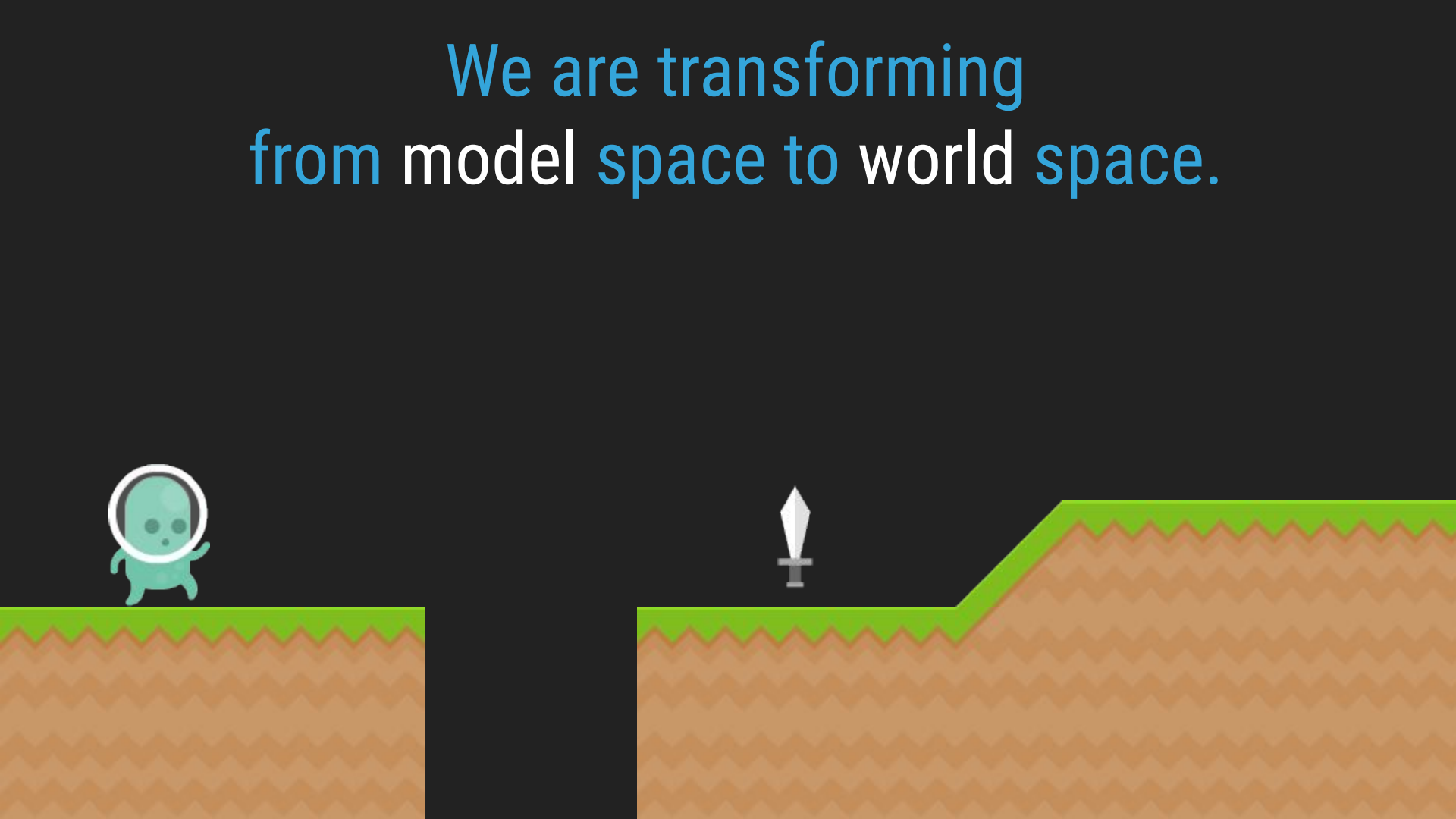
Model Space





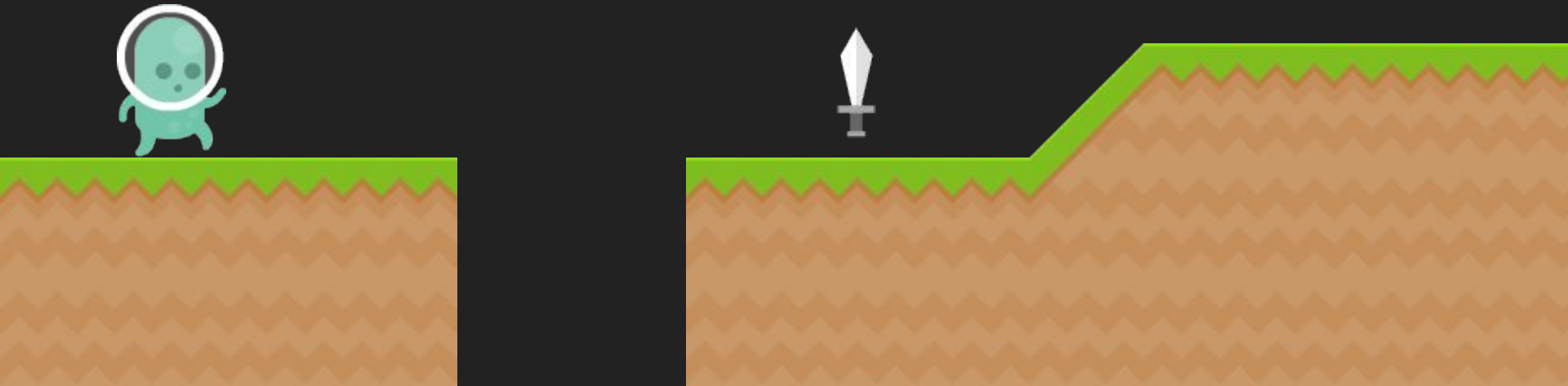

World Space

We are transforming
from model space to world space.



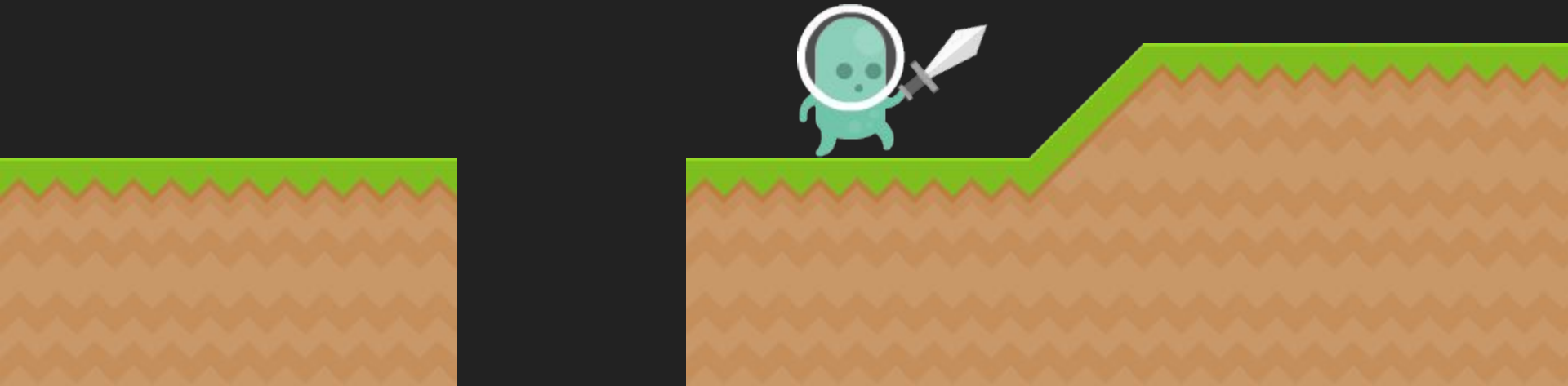
We are transforming from model space to world space.

```
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(1.0f, 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 it every frame.

(but this could get weird)

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

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

Timing and FPS

The game loop will happen as fast as your hardware can run it.

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

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

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 values on different computers:

60 FPS: 1/60 * 1000 = 16.66

30 FPS: 1/30 * 1000 = 33.33

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

60 FPS

30 FPS

We can use timing to handle rotation as well.

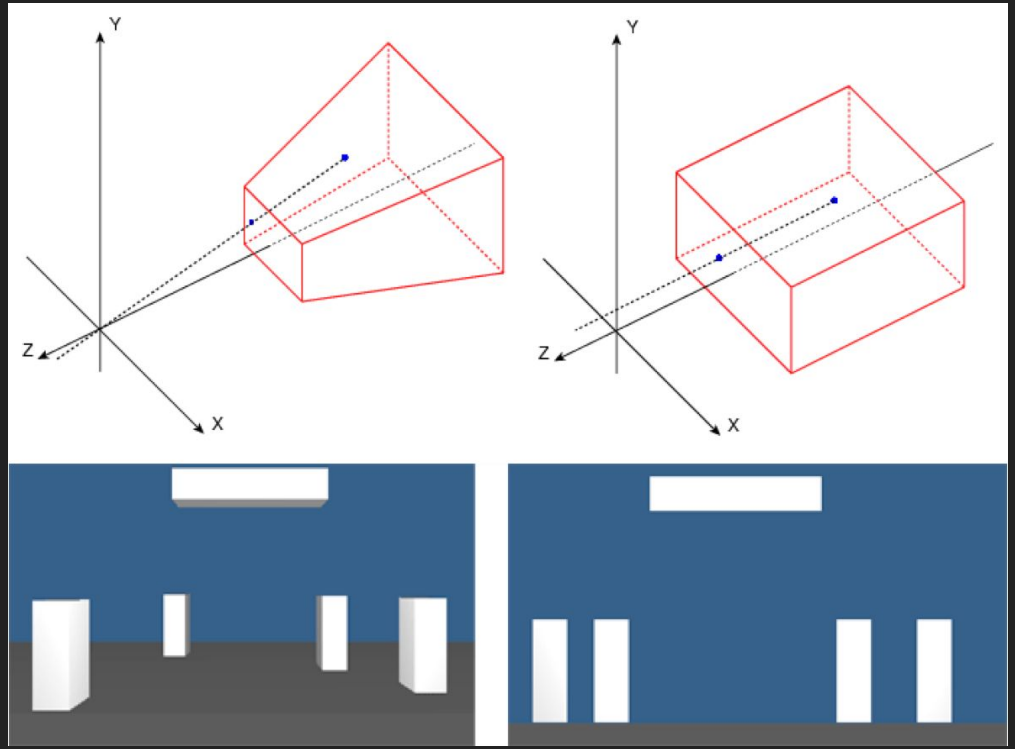
```
// Rotate 45 degrees per second
rotate_z += 45.0f * deltaTime;


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

One more thing...

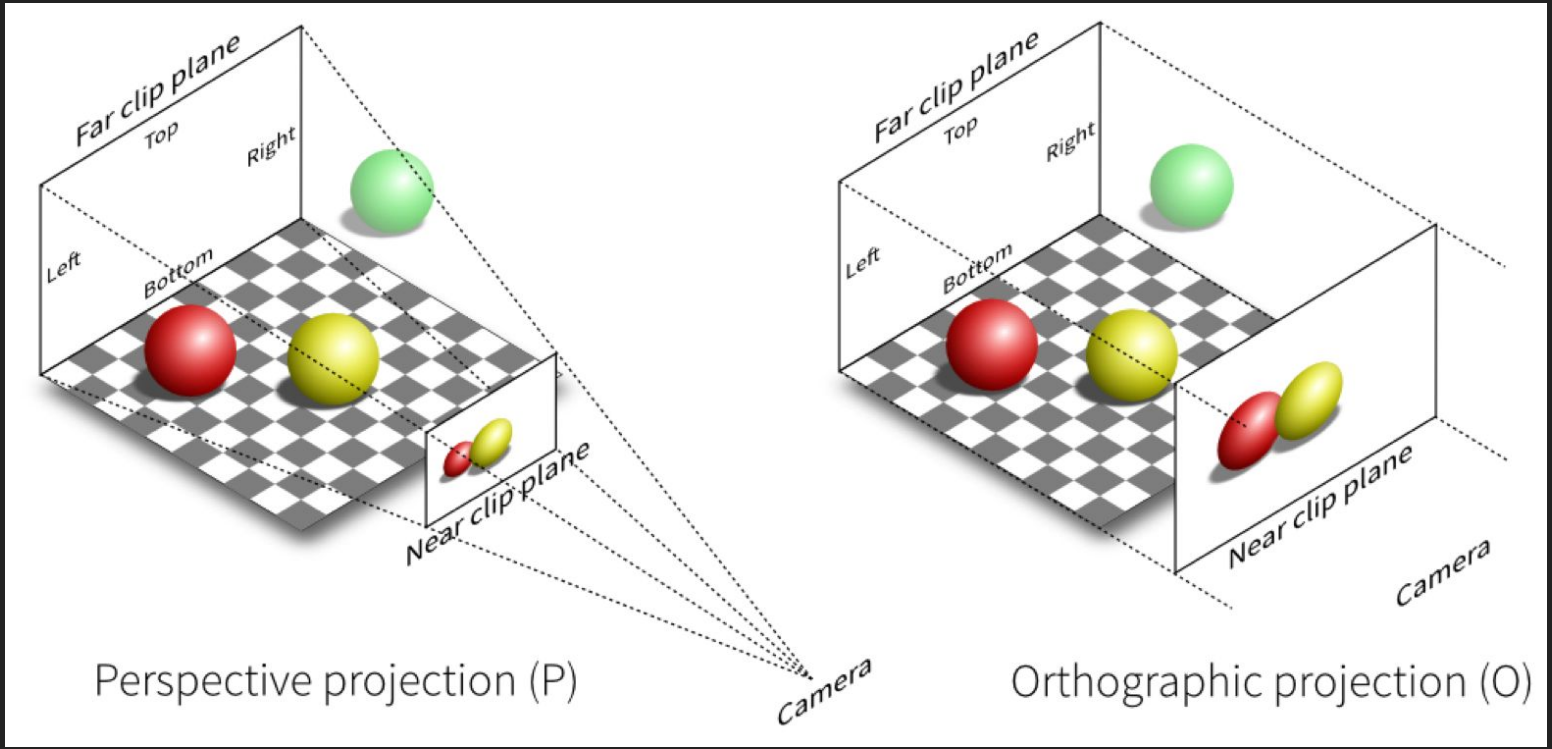
You may have noticed:

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

Perspective vs. Orthographic



Perspective vs. Orthographic



Let’s Code!