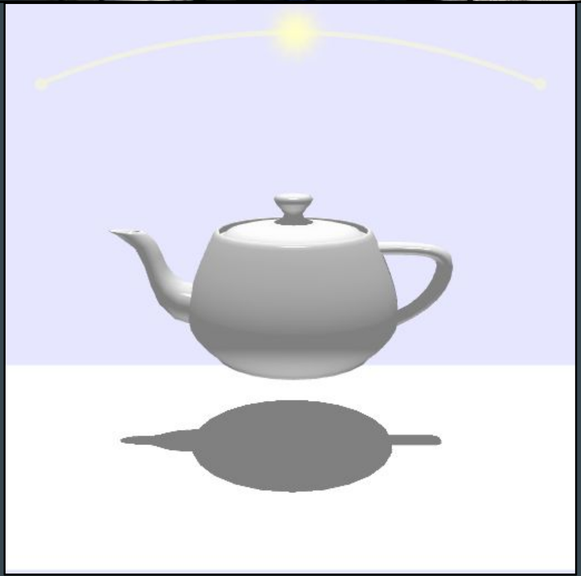
Game Camera

Ricard Pillosu | Marc Garrigó

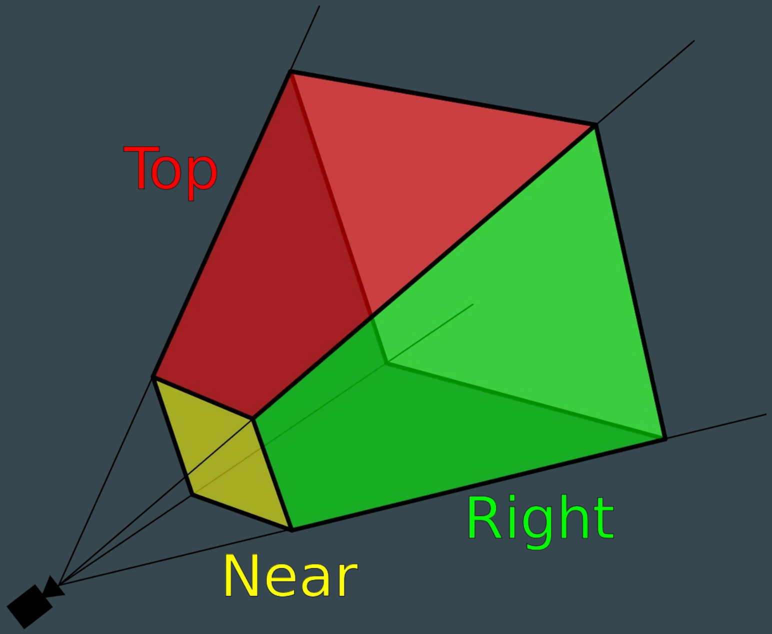
The camera

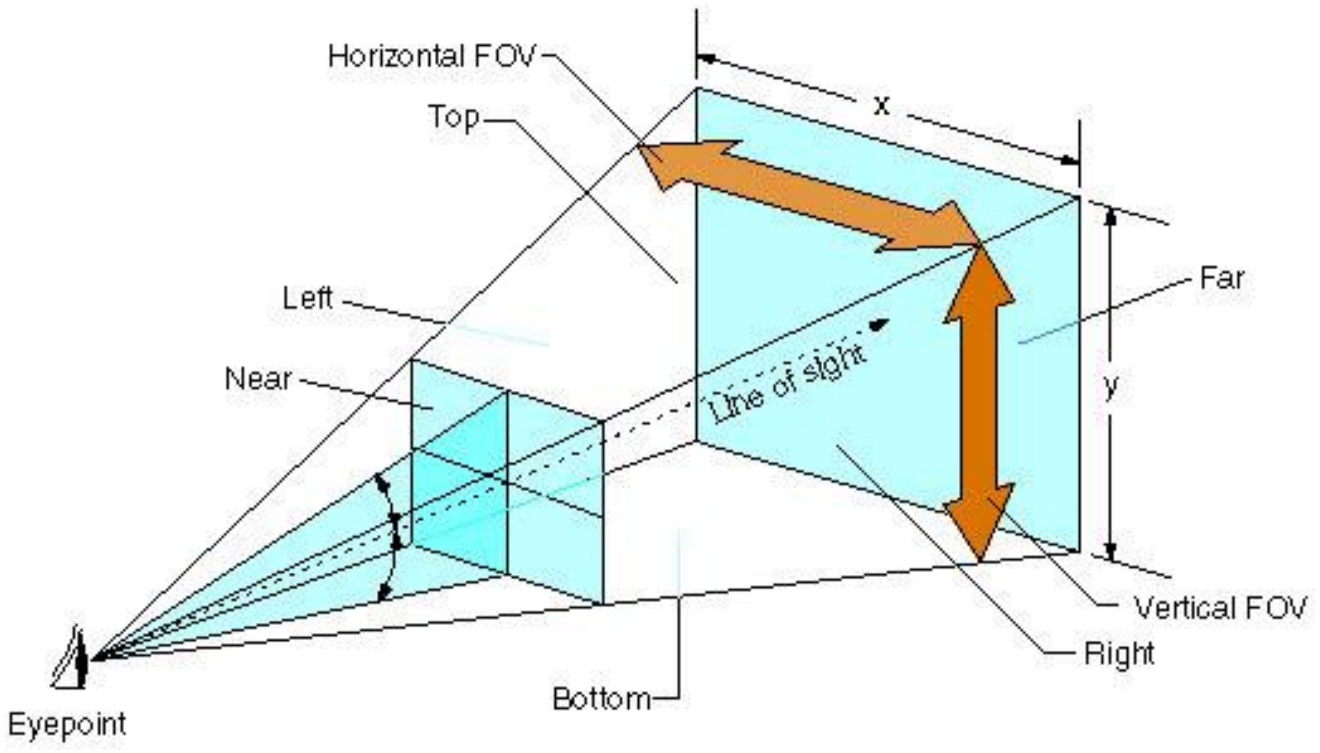
- Video games have to handle multiple cameras
- Obvious uses include split-screen, mirrors, etc.
- Non-Obvious are shadow map computation, reflections, etc.
- We could have many cameras and only one being the graphical camera



The camera

- Mathematically we will express it as a Frustum





$$\text{Aspect Ratio} = \frac{x}{y} = \frac{\tan(\text{horizontal FOV}/2)}{\tan(\text{vertical FOV}/2)}$$

Field of View

$$r = \frac{w}{h} = \frac{\tan\left(\frac{H}{2}\right)}{\tan\left(\frac{V}{2}\right)}$$

$$H = 2\arctan\left(\tan\left(\frac{V}{2}\right) \times \frac{w}{h}\right)$$

$$V = 2\arctan\left(\tan\left(\frac{H}{2}\right) \times \frac{h}{w}\right)$$

- Near / Far plane are very intuitive, FOV not so much :-S
  - Games / engines usually only mention "FOV" value
- Most common technique is to adapt horizontal FOV to aspect ratio changes
- This technique is called HOR+, basically to support wide screens
  - WebGL Online FOV viewer here uses HOR+, try resizing the browser window
- Projection Matrix . . .
  - Only needs recalculation when FOV / Aspect Ratio or near / far change

FOV manipulation cases in video games

Field of View in games: 90 to 60 FOV

Field of View in movies: dolly zoom or “vertigo effect”

Field of View: sprint effect

Field of View: FPS weapon and hands

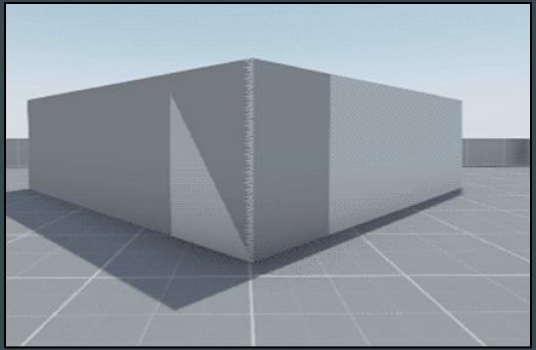
Z-Buffer

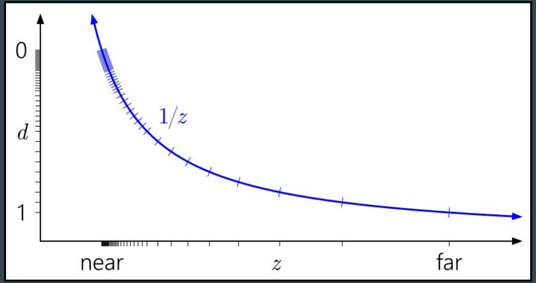
- Buffer that stores per pixel depth *(very interesting read in general graphics [here](#))*

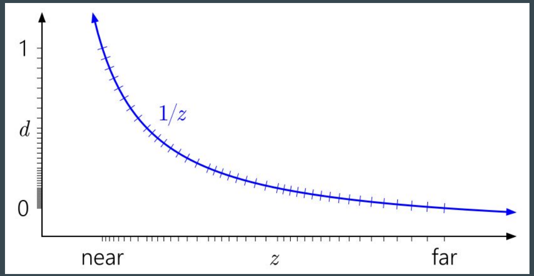
File
View
Help
Configuration
Application
Window
Renderer3D
Module Actived
Switch wireframe
Switch Depth Test
Switch Z-Buffer Visualization
Switch Cull Face
Switch Lighting
Switch Color Material
Switch Texture 2D
Input
Hardware

Z-Fighting

- Our camera has a float per pixel to store its depth / z
- This is called the z buffer and has its limitations
- When pixels have very similar z they can cause an artifact called z-fighting
- This is specially relevant for decals
- No easy solution, normally we manipulate the depth







Implementation

- MathGeoLib contains a good frustum class that will do most of the work:
    - Near / Far distance
    - Horizontal and Vertical FOV
    - Aspect Ratio (relation between FOVs)
    - Handedness (left vs. right)
    - Type (perspective vs. orthographic)
    - GetPlanes(Plane* outArray) const → useful for manual calculations

*** Note: The repo source code has more utilities than the Releases section**

Direct Mode: Reference ground (avoid ASAP)

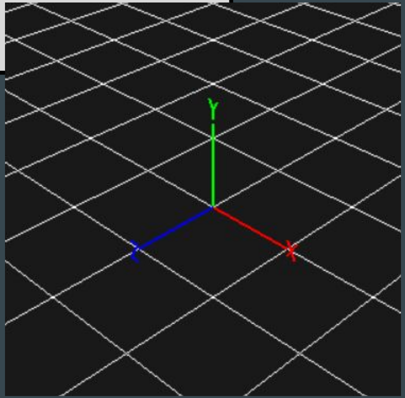
```
glLineWidth(1.0f);

float d = 200.0f;

glBegin(GL_LINES);
for(float i = -d; i <= d; i += 1.0f)
{
      glVertex3f(i, 0.0f, -d);
      glVertex3f(i, 0.0f, d);
      glVertex3f(-d, 0.0f, i);
      glVertex3f(d, 0.0f, i);
}
glEnd();
```

Direct Mode: Reference axis 1/2 (avoid ASAP)

```
glLineWidth(2.0f);
glBegin(GL_LINES);

// red X
glColor4f(1.0f, 0.0f, 0.0f, 1.0f);
glVertex3f(0.0f, 0.0f, 0.0f); glVertex3f(1.0f, 0.0f, 0.0f);
glVertex3f(1.0f, 0.1f, 0.0f); glVertex3f(1.1f, -0.1f, 0.0f);
glVertex3f(1.1f, 0.1f, 0.0f); glVertex3f(1.0f, -0.1f, 0.0f);

// green Y
glColor4f(0.0f, 1.0f, 0.0f, 1.0f);
glVertex3f(0.0f, 0.0f, 0.0f); glVertex3f(0.0f, 1.0f, 0.0f);
glVertex3f(-0.05f, 1.25f, 0.0f); glVertex3f(0.0f, 1.15f, 0.0f);
glVertex3f(0.05f, 1.25f, 0.0f); glVertex3f(0.0f, 1.15f, 0.0f);
glVertex3f(0.0f, 1.15f, 0.0f); glVertex3f(0.0f, 1.05f, 0.0f);
```

Direct Mode: Reference axis 2/2 (avoid ASAP)

```
// blue Z
glColor4f(0.0f, 0.0f, 1.0f, 1.0f);
glVertex3f(0.0f, 0.0f, 0.0f); glVertex3f(0.0f, 0.0f, 1.0f);
glVertex3f(-0.05f, 0.1f, 1.05f); glVertex3f(0.05f, 0.1f, 1.05f);
glVertex3f(0.05f, 0.1f, 1.05f); glVertex3f(-0.05f, -0.1f, 1.05f);
glVertex3f(-0.05f, -0.1f, 1.05f); glVertex3f(0.05f, -0.1f, 1.05f);

glEnd();
glLineWidth(1.0f);
```



Implementation

- For now we will create a new module for the editor camera
  - In the future, when (if) we include support for multiple cameras, we will improve this code
- It should wrap the *Frustum* class from MathGeoLib
- Add methods to manipulate it:
  - SetFOV() ... should set the **horizontal FOV** keeping the aspect ratio
  - SetAspectRatio() ... should change the **vertical FOV** to meet the new aspect ratio.
  - SetPlaneDistances() / Position() / Orientation() / LookAt(x,y,z)
  - GetProjectionMatrix() OpenGL matrix order is different from default MathGeoLib! *Math info here*
  - GetViewMatrix ()... OpenGL matrix order is different from default MathGeoLib! *Math info here*
- Detect window resize and trigger FOV recalculation accordingly

Perspective matrix

- Using Frustum from MathGeoLib to generate projection matrix

```
Frustum frustum;

frustum.SetKind(FrustumSpaceGL, FrustumRightHanded);
frustum.SetViewPlaneDistances(0.1f, 200.0f);
frustum.SetHorizontalFovAndAspectRatio(DEGTORAD * 90.0f, 1.3f);

frustum.SetPos(float3(0.0f, 1.0f, -2.0f));
frustum.SetFront(float3::unitZ);
frustum.SetUp(float3::unitY);

float4x4 projectionGL = frustum.ProjectionMatrix().Transposed();//<-- Important to transpose!

//Send the frustum projection matrix to OpenGL
// direct mode would be:
glMatrixMode(GL_PROJECTION);
glLoadMatrixf(*projectionGL.v);
```

View matrix

- Start playing with different frustum transformations (position and rotation)
- Send the camera's view matrix to OpenGL at the beginning of the frame

```
frustum.SetPos(/* some position value*/)

float3x3 rotationMatrix; // = some rotation value (or LookAt matrix)

frustum.SetFront(rotationMatrix.WorldX());
frustum.SetUp(rotationMatrix.WorldY());

//Send the frustum view matrix to OpenGL
// direct mode would be:
float4x4 viewGL = float4x4(frustum.ViewMatrix()).Transposed();
glMatrixMode(GL_MODELVIEW);
glLoadMatrixf(*viewGL.v);
```

View matrix - Handling Rotations

- We will soon have to manipulate the camera rotation through user input
- MathGeoLib already gives you a lot of utility functions for manipulating transforms
- Adding rotation values to the current orientation can be done as so:

```
float3x3 rotationDeltaMatrix; // = some rotation delta value

vec oldFront = frustum.Front().Normalized();
frustum.SetFront(rotationDeltaMatrix.MultDir(oldFront);

vec oldUp = frustum.Up().Normalized();
frustum.SetUp(rotationDeltaMatrix.MultDir(oldUp);
```

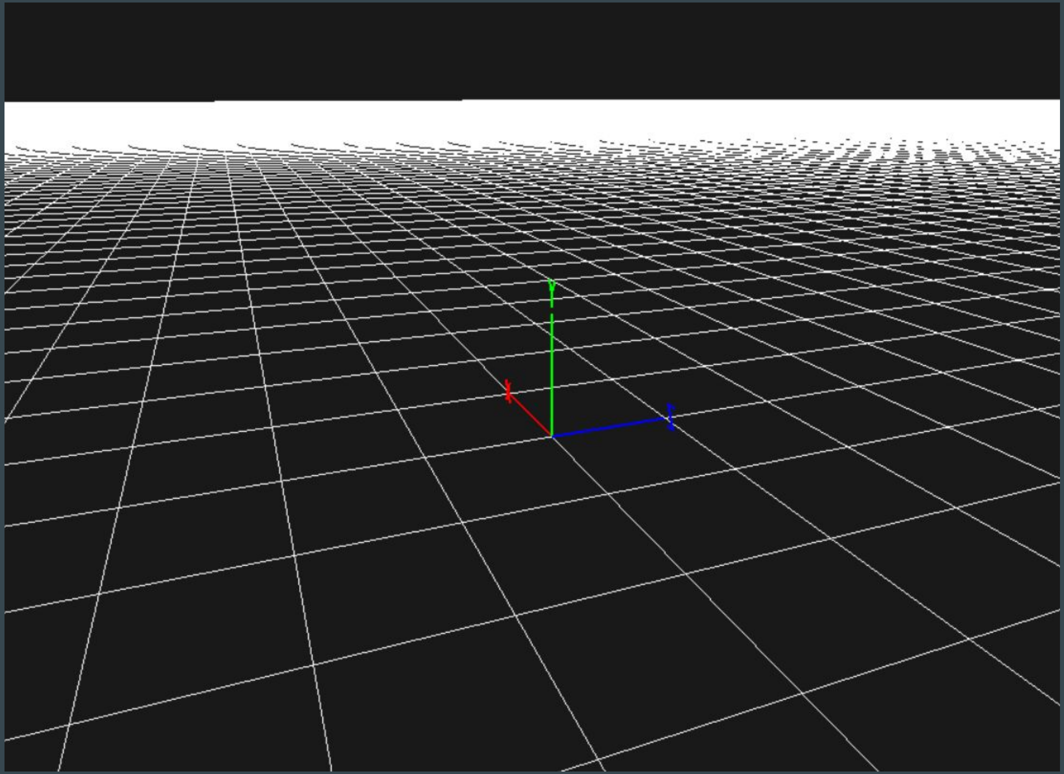
- Basic transformations tutorial here

Test

Now draw the grid and a gizmo at world center of coordinates

Try different positions and look-at points

* Bonus: Test resizing your window: shapes should not appear distorted / twisted



Camera movement in the editor

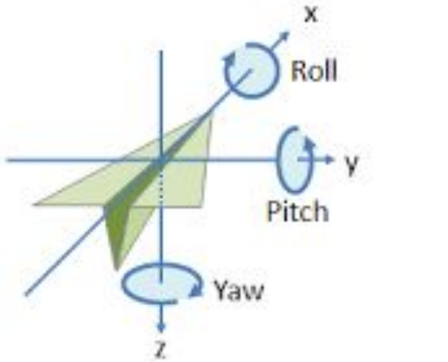
- Most editors implement their own controls
- Our goal is to mimic Unity:
    - Left/middle click: drag camera around 
    - Orbit: Alt + left click 
    - Zoom: Alt + right click 
    - Right button: rotate
    - Right button pressed: move with WASD + QE for up/down (+SHIFT to move faster)
- Start with a basic implementation of WASD+QE movement only

Implementation

Start by capturing keyboard event in the Camera module

1. Have Q/E go up/down in absolute values (try local if you prefer)
2. Have W/S move forward and backward relative to camera orientation
   a. Remember: you need to move along the **frustum.front** vector
3. Have A/D move left and right relative to camera orientation
   a. Move along the vector from **frustum.WorldRight()**
4. Each pair should perfectly cancel each other

Implementation



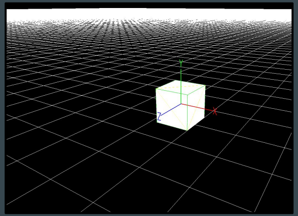
Axis of Rotation	Euler Angle Name	Euler Angle Symbol
x	Roll	u
y	Pitch	v
z	Yaw	w

Let’s simulate rotation with **arrow keys**

1. Have up/down arrow keys rotate the camera’s Pitch
    a. Generate a rotation Matrix or Quaternion that expresses the rotation you want
    b. Use it to transform (multiply) frustum.up and frustum.front vectors
2. Same for left/right affecting camera’s Yaw
    a. Recommendation: rotate in world Z axis (try camera’s Z axis and compare)
3. Finally use [mouse motion] to rotate the camera in the same way

Notes

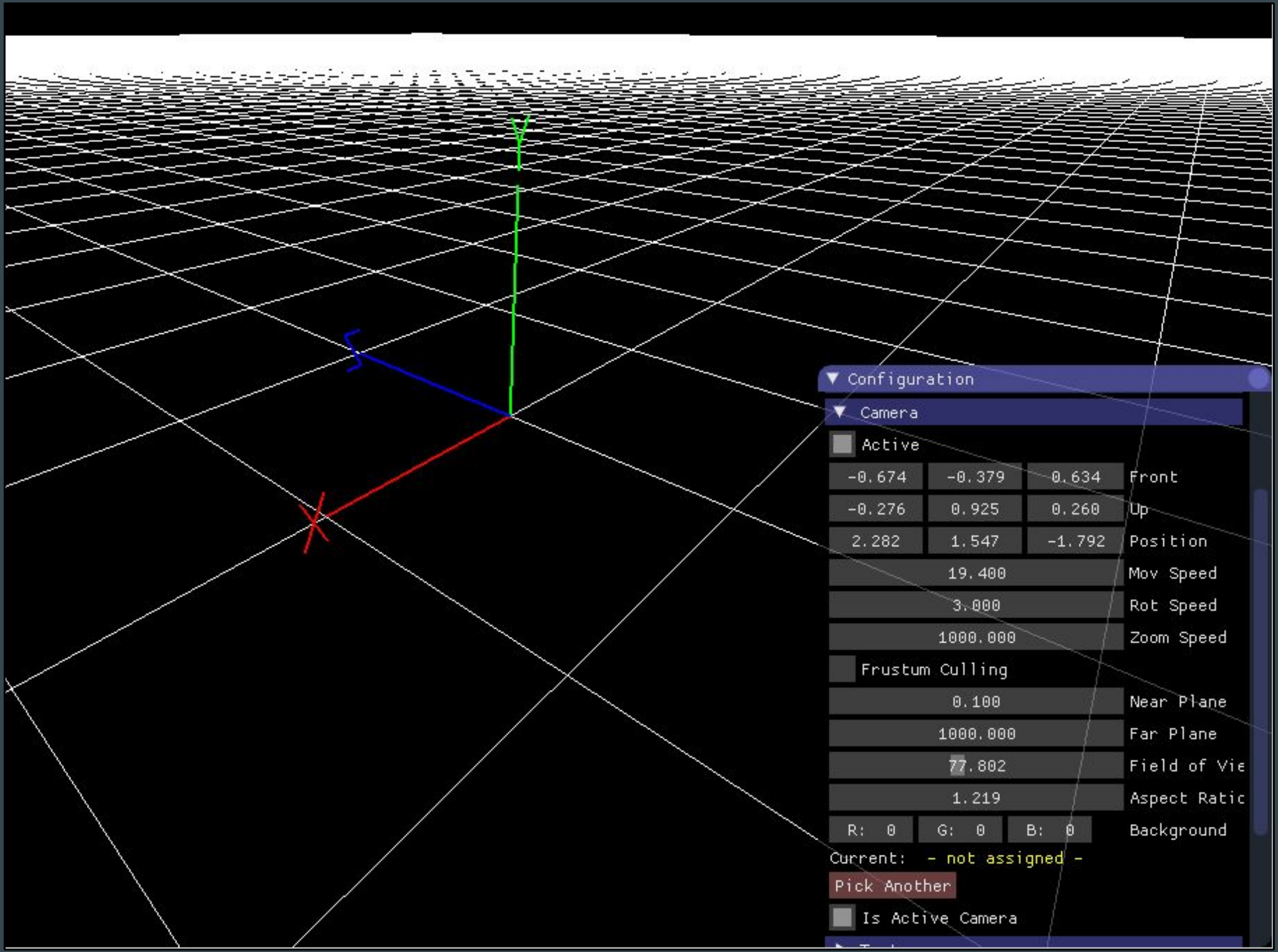
- The camera movement should be normalized using **delta time**
- Have the camera speed double/triple if SHIFT is being pressed
- You can read mouse wheel data to move forward/backward quickly
- Limit angles up/down to avoid disorienting the user
- Camera Orbit is complex and more suitable after we implement object picking



Homework

- Have a Camera Module that generates the perspective and view matrices
- Draw a grid on the ground for orientation
- Have all the camera controls from Unity (except orbit for now)
- React to window resize, calculate Aspect Ratio and modify FOVs accordingly
- Prepare the module to receive different settings through an Editor Window
  - FOV horizontal and vertical, near/far plane distance / . . .
  - Movement / Rotation Speed / background color / . . .

Homework



Y
Z
X
Configuration
Camera
Active
-0.674 -0.379 0.634 Front
-0.276 0.925 0.260 Up
2.282 1.547 -1.792 Position
19.400 Mov Speed
3.000 Rot Speed
1000.000 Zoom Speed
Frustum Culling
0.100 Near Plane
1000.000 Far Plane
77.802 Field of Vie
1.219 Aspect Ratic
R: 0 G: 0 B: 0 Background
Current: - not assigned -
Pick Another
Is Active Camera