Matrices, Transformation and Projection

Homogeneous Coordinates

- We will now have (x, y, z, w) vectors
  - If w == 1, then the vector (x, y, z, 1) is a position in space
  - If w == 0, then the vector (x,y,z,0) is a direction

- Why positions and direction differ?

glm

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} ax + by + cz + dw \\ ex + fy + gz + hw \\ ix + jy + kz + lw \\ mx + ny + oz + pw \end{bmatrix}$$

```
glm::mat4 myMatrix;
glm::vec4 myVector;
// fill myMatrix and myVector somehow
glm::vec4 transformedVector = myMatrix * myVector;
// Again, in this order ! this is important.
```

https://glm.g-truc.net/0.9.8/index.html

Translation

$$\begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 10 \\ 10 \\ 10 \\ 1 \end{bmatrix} = \begin{bmatrix} 1*10+0*10+0*10+10*1 \\ 0*10+1*10+0*10+0*1 \\ 0*10+0*10+1*10+0*1 \\ 0*10+0*10+0*10+1*1 \end{bmatrix} = \begin{bmatrix} 10+0+0+10 \\ 0+10+0+0 \\ 0+0+10+0 \\ 0+0+0+1 \end{bmatrix} = \begin{bmatrix} 20 \\ 10 \\ 10 \\ 1 \end{bmatrix}$$

```
#include <glm/gtx/transform.hpp> // after <glm/glm.hpp>

glm::mat4 myMatrix = glm::translate(glm::mat4(), glm::vec3(10.0f, 0.0f, 0.0f));
glm::vec4 myVector(10.0f, 10.0f, 10.0f, 1.0f);
glm::vec4 transformedVector = myMatrix * myVector; // guess the result
```

Scaling

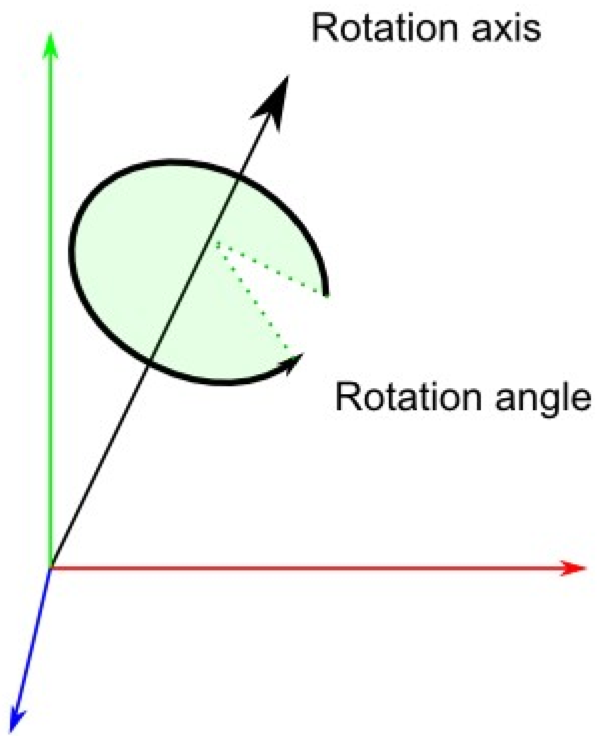
$$\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 2*x+0*y+0*z+0*w \\ 0*x+2*y+0*z+0*w \\ 0*x+0*y+2*z+0*w \\ 0*x+0*y+0*z+1*w \end{bmatrix} = \begin{bmatrix} 2*x+0+0+0 \\ 0+2*y+0+0 \\ 0+0+2*z+0 \\ 0+0+0+1*w \end{bmatrix} = \begin{bmatrix} 2*x \\ 2*y \\ 2*z \\ w \end{bmatrix}$$

```
// Use #include <glm/gtc/matrix_transform.hpp> and #include <glm/gtx/transform.hpp>
glm::mat4 myScalingMatrix = glm::scale(glm::mat4(), glm::vec3(2.0f, 2.0f ,2.0f));
```

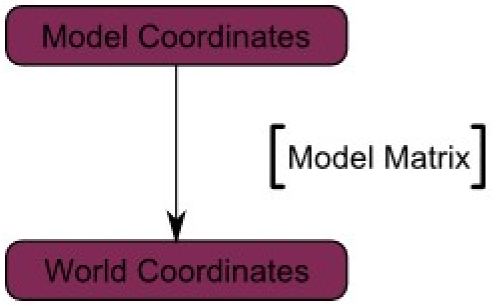
Rotations

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_y = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_z = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



```
using namespace glm;
mat4 rotation = rotate(mat4(1.0), 3.14f / 4.0f, vec3(0.0f, 0.0f, 1.0f));
```

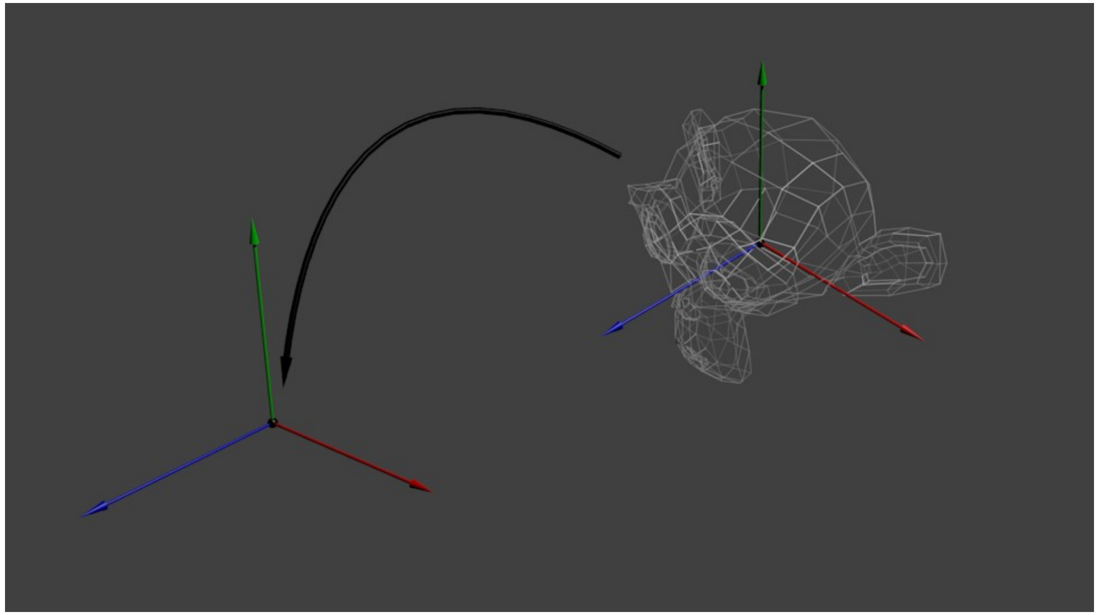
The Model Matrix



Model Coordinates

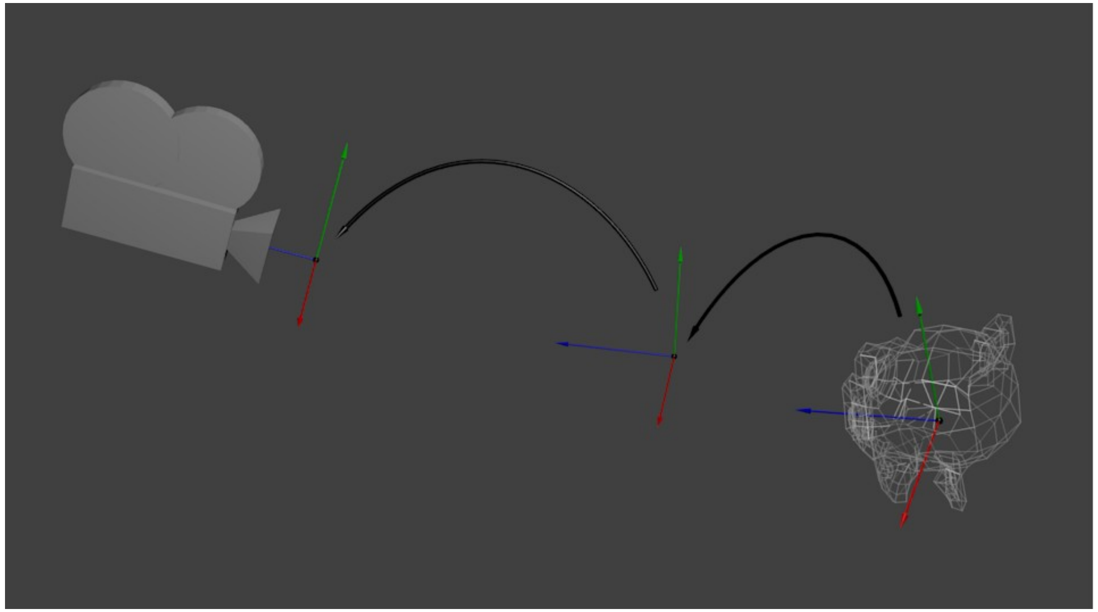
[Model Matrix]

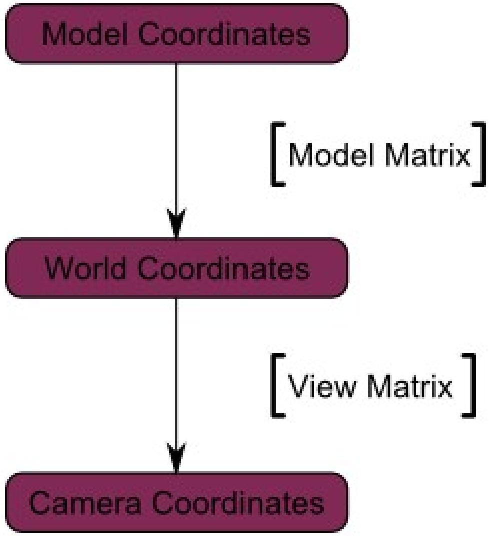
World Coordinates



View Matrix

“The engines don’t move the ship at all. The ship stays where it is and the engines move the universe around it.”, Futurama





Model Coordinates

[Model Matrix]

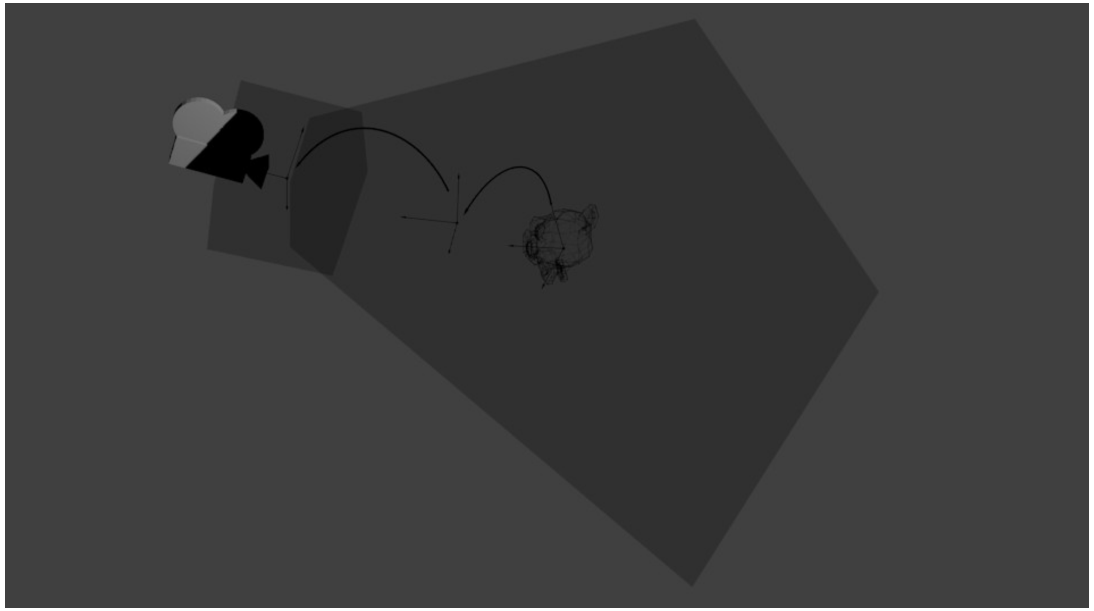
World Coordinates

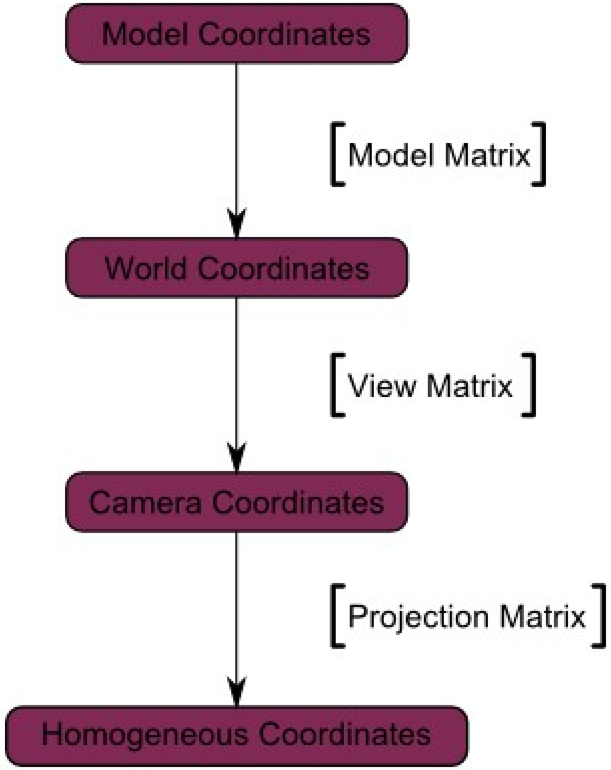
[View Matrix]

Camera Coordinates

```
glm::mat4 CameraMatrix = glm::lookAt(
    cameraPosition, // the position of your camera, in world space
    cameraTarget,   // where you want to look at, in world space
    upVector        // probably glm::vec3(0,1,0), but (0,-1,0) would
make you looking upside-down, which can be great too
);
```

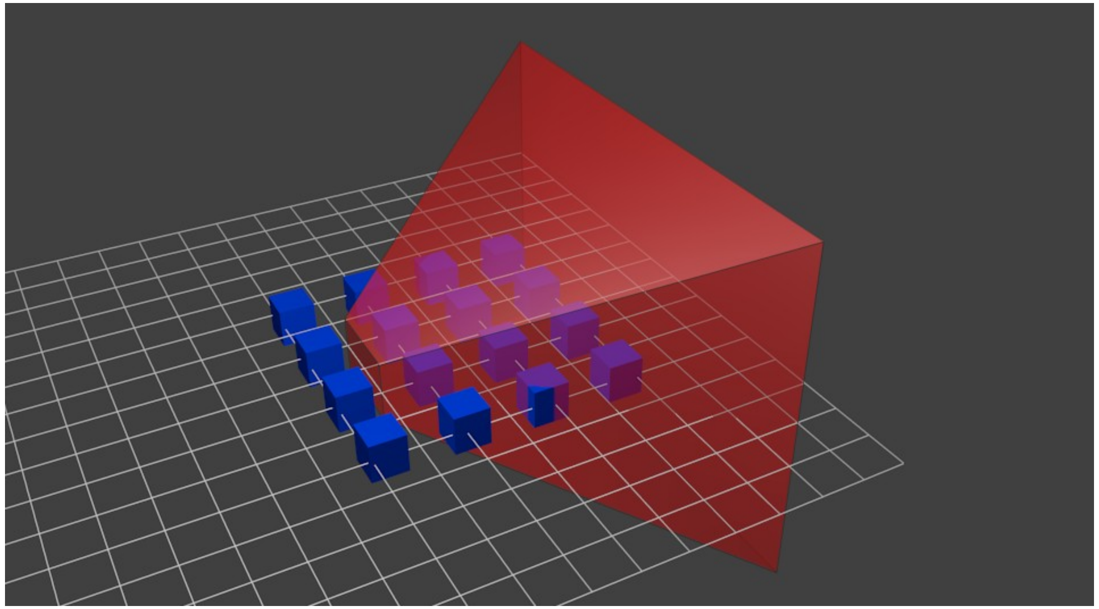
The Projection Matrix

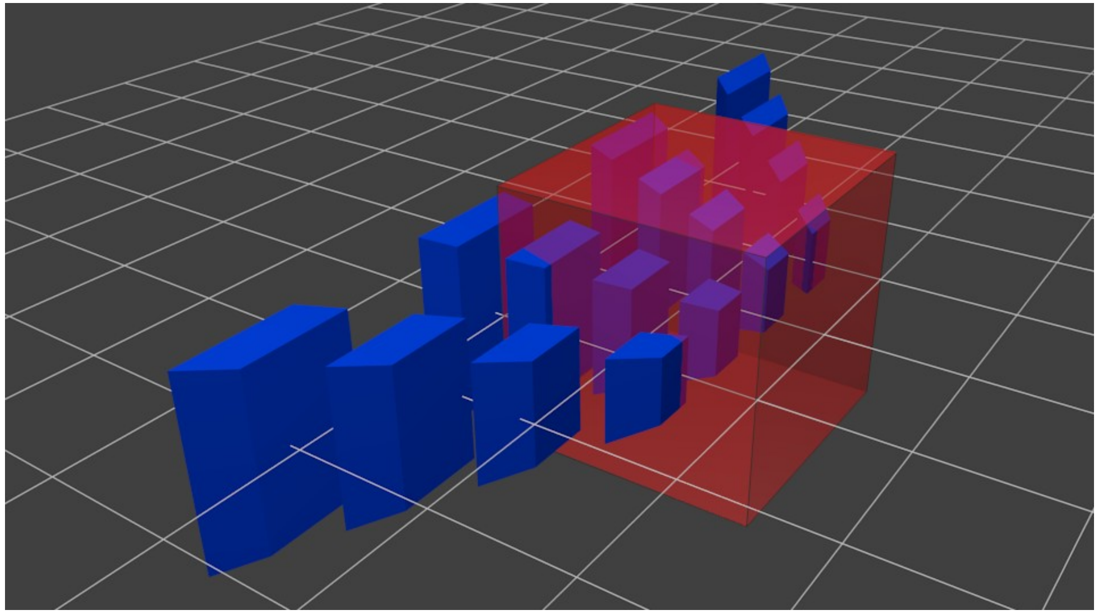




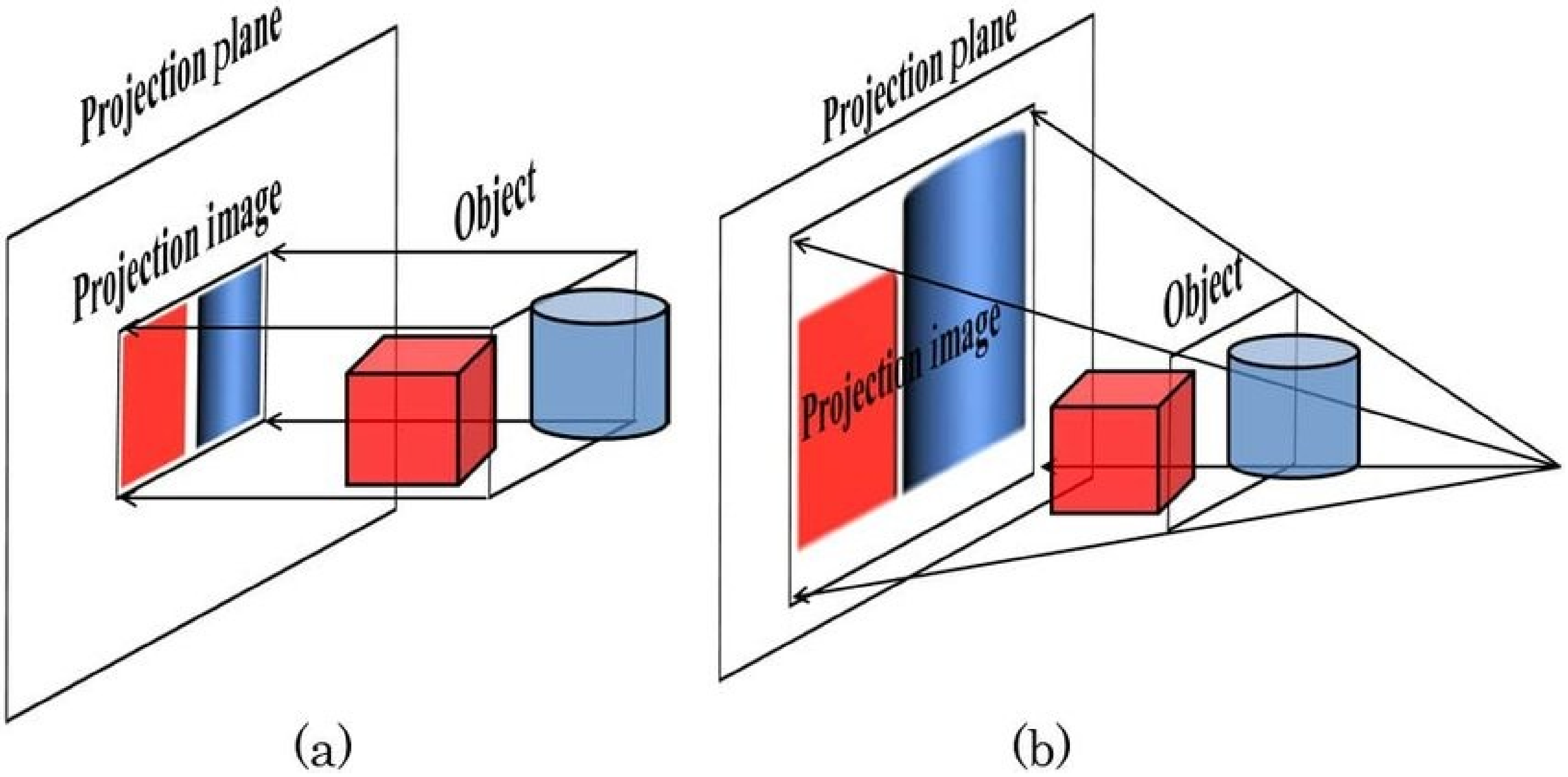
```
glm::mat4 projectionMatrix = glm::perspective(
    glm::radians(FoV), // The vertical Field of View, in radians: the amount of "zoom". Think "camera lens". Usually between 90° (extra wide) and 30° (quite zoomed in)
    4.0f / 3.0f,       // Aspect Ratio. Depends on the size of your window. Notice that 4/3 == 800/600 == 1280/960, sounds familiar ?
    0.1f,              // Near clipping plane. Keep as big as possible, or you'll get precision issues.
    100.0f             // Far clipping plane. Keep as little as possible.
);
```

Perspective Projection



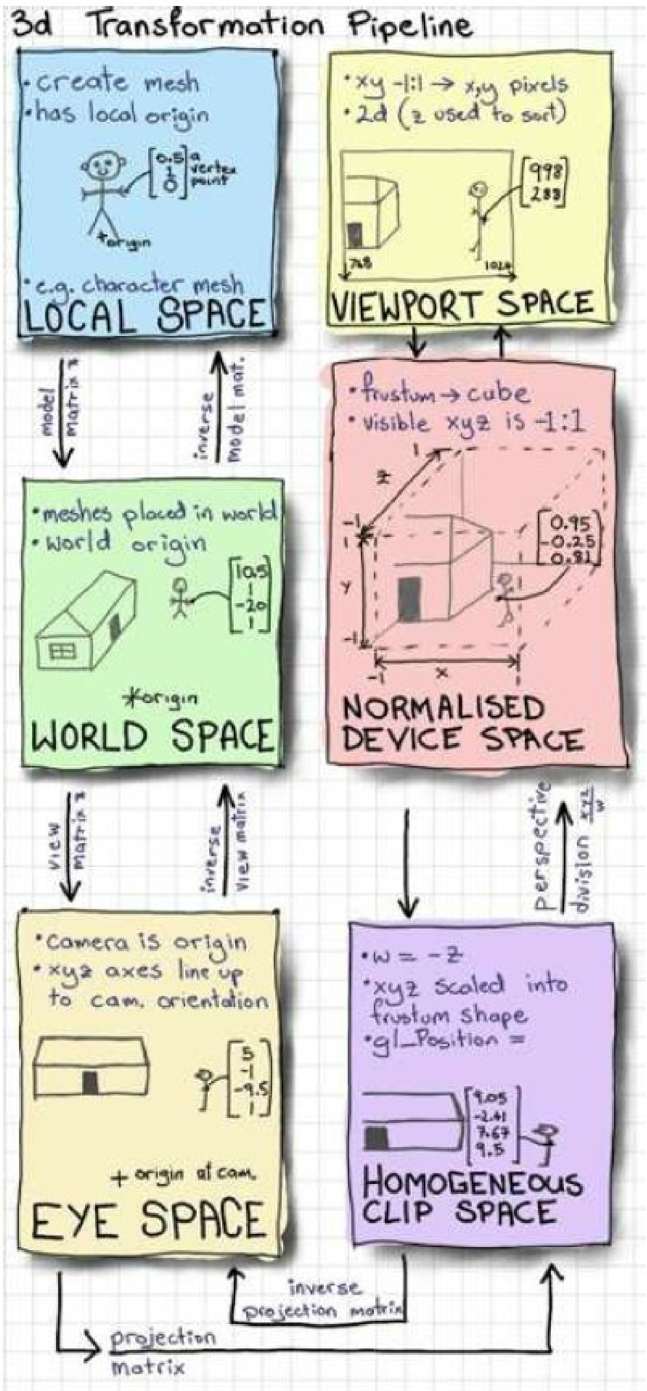


Perspective vs Orthographic projection



3D Transformation Pipeline

- Local coordinates
- Word coordinates (**model**)
- Camera space (**view**)
- Homogeneous space (shader)
- Normalized device space (**projection**)
- Viewport space (screen)



GLSL uniform variables

lab02.cpp

```
// Get a handle for our "MVP" uniform
// Only during the initialisation
GLuint mvpID =
glGetUniformLocation(program_id, "MVP");

// Send our transformation to the currently
bound shader, in the "MVP" uniform
// This is done in the main loop since each
model will have a different MVP matrix (At
least for the M part)
glUniformMatrix4fv(mvpID, 1, GL_FALSE,
&mvp[0][0]);
```

Vertex shader

```
// Input vertex data, different for all
executions of this shader.
layout(location = 0) in vec3
vertexPosition_modelspace;

// Values that stay constant for the whole mesh.
uniform mat4 MVP;

void main(){
  // Output position of the vertex, in clip
space : MVP * position
  gl_Position =  MVP *
vec4(vertexPosition_modelspace,1);
}
```

The Z-Buffer

- The Z-Buffer sorts the fragments according to their depth so that the GPU is able to draws them in the correct order