Transformations

Evangelou Iordanis

Presentation co-author Anastasios Gkaravelis

Recap

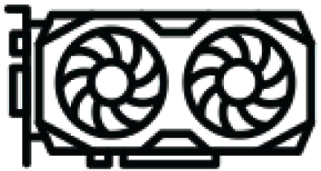
CPU side

GPU side

```
createwindow();
createOpenGL_context();
glewInit();
init_engine();

while(true)
{
    e = pollevents();
    renderer.update(e);
    renderer.draw();
}
```

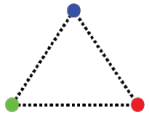
Engine

OpenGL



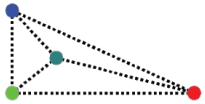
Recap

Input

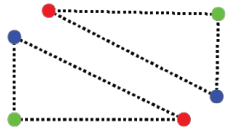


Vertex Shader

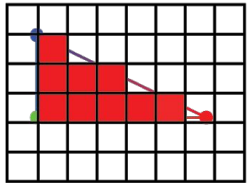
Tessellation Shader



Geometry Shader



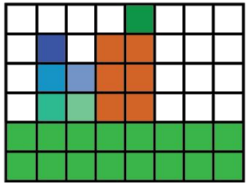
Rasterization



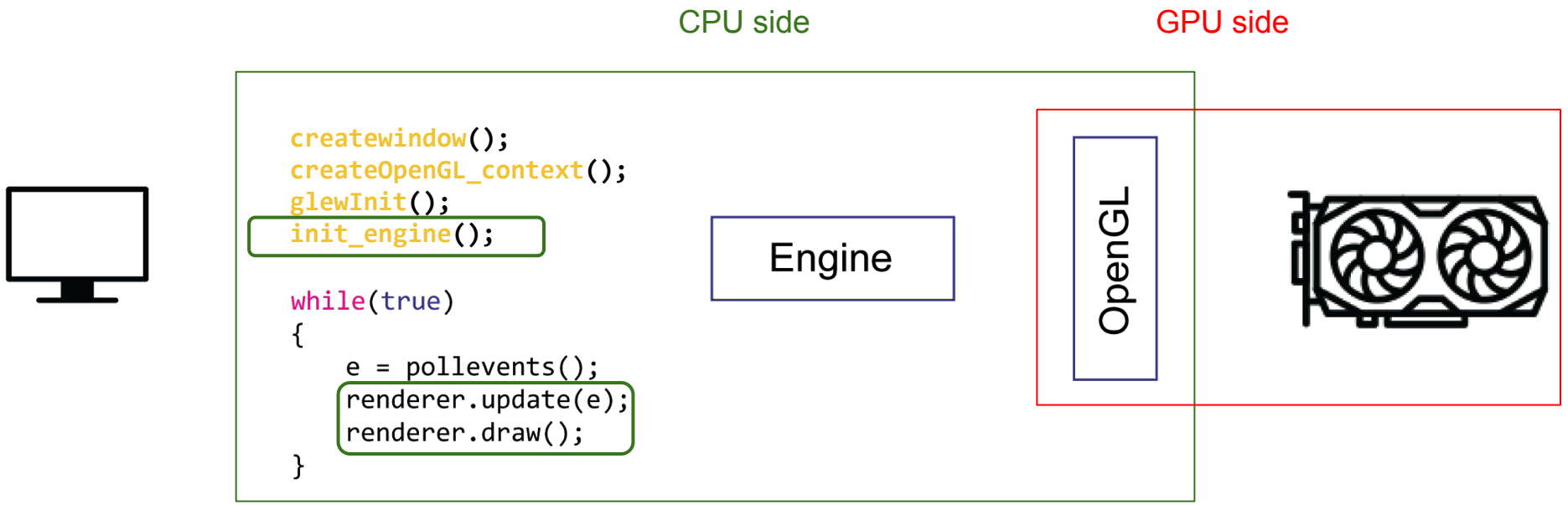
Fragment Shader

Fragment Test

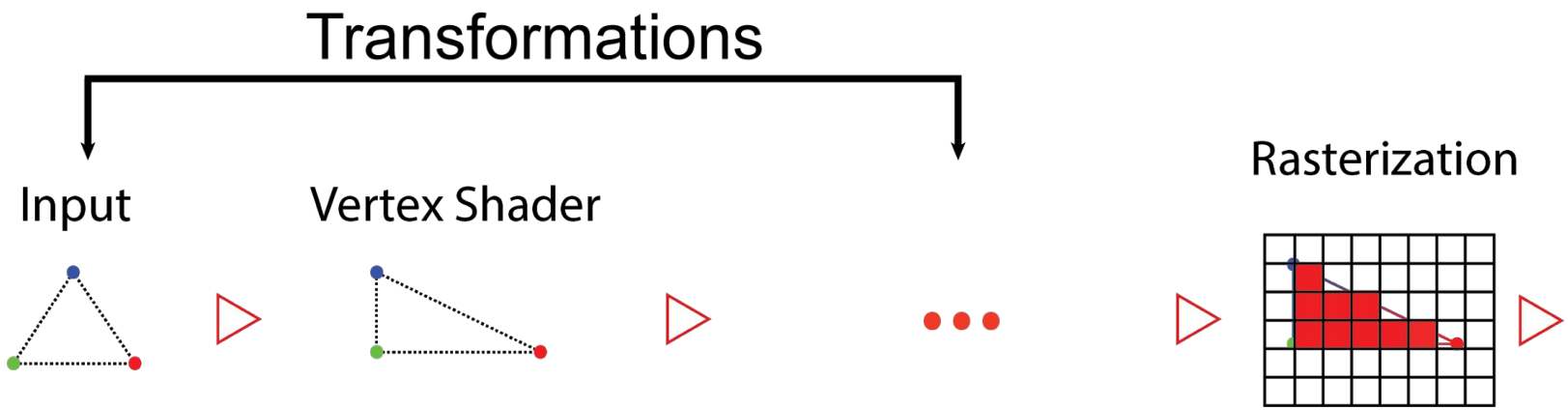
Output Buffer



Transformations pipeline



Transformations pipeline



Transformations

- Two basic types of transformations
  - Manipulate 3D-coordinates
    - Translation, Rotation, Scaling
  - Alternate between coordinate systems
- We describe transformations with matrices (column major)
  - `glm::mat4, mat4` (GLSL)

```
glm::mat4 T(1.f);                          // identity matrix
glm::vec4 column0 = T[0];                  // index first column
glm::vec4 column1 = glm::column(T, 1);     // index second column
glm::vec4 row1 = glm::row(T, 1);           // index second row
float elem = T[0][1];                      // first column, second row
```

Transformations

- Two basic types of transformations
  - Manipulate 3D-coordinates
    - Translation, Rotation, Scaling
  - Alternate between coordinate systems
- We describe transformations with matrices (column major)
  - `glm::mat4, mat4` (GLSL)
- We apply transformations with matrix-vector multiplications

```
glm::mat4 T = glm::translate(glm::mat4(1.f), glm::vec3(1.f, 0.f, 0.f));
glm::vec4 pos(2.f, 0.f, 0.f, 1.f);
glm::vec4 newPos = T * pos;
```

Transformations

- Two basic types of transformations
  - Manipulate 3D-coordinates
    - Translation, Rotation, Scaling
  - Alternate between coordinate systems
- We describe transformations with matrices (column major)
  - `glm::mat4, mat4` (GLSL)
- We apply transformations with matrix-vector multiplications
- Stack multiple transformations

```
glm::mat4 T1 = glm::translate(glm::mat4(1.f), glm::vec3(1.f, 0.f, 0.f));
glm::mat4 T2 = glm::translate(glm::mat4(1.f), glm::vec3(0.f, 1.f, 0.f));
glm::mat4 T3 = glm::translate(glm::mat4(1.f), glm::vec3(0.f, 0.f, 1.f));
glm::vec4 pos(2.f, 0.f, 0.f, 1.f);
glm::vec4 newPos = T3 * T2 * T1 * pos;
```

Transformations

- Two basic types of transformations
  - Manipulate 3D-coordinates
    - Translation, Rotation, Scaling
  - Alternate between coordinate systems
- We describe transformations with matrices (column major)
  - `glm::mat4, mat4` (GLSL)
- We apply transformations with matrix-vector multiplications
- Stack multiple transformations
- Invert to prior states

```
glm::mat4 T1 = glm::translate(glm::mat4(1.f), glm::vec3(1.f, 0.f, 0.f));
glm::mat4 T2 = glm::translate(glm::mat4(1.f), glm::vec3(0.f, 1.f, 0.f));
glm::mat4 T3 = glm::translate(glm::mat4(1.f), glm::vec3(0.f, 0.f, 1.f));
glm::vec4 pos(2.f, 0.f, 0.f, 1.f);
glm::vec4 newPos = T3 * T2 * T1 * pos;
glm::vec4 oldPos1 = glm::inverse(T1) * glm::inverse(T2) * glm::inverse(T3) * newPos;
glm::vec4 oldPos2 = glm::inverse(T3 * T2 * T1) * newPos;
```

Transformations

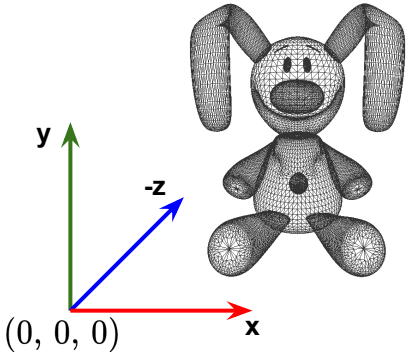
- Two basic types of transformations
  - Manipulate 3D-coordinates
    - Translation, Rotation, Scaling
  - Alternate between coordinate systems
- We describe transformations with matrices (column major)
  - `glm::mat4, mat4` (GLSL)
- We apply transformations with matrix-vector multiplications
- Stack multiple transformations
- Invert to prior states

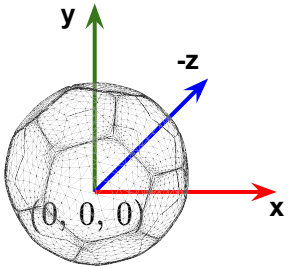
```
glm::mat4 T1 = glm::translate(glm::mat4(1.f), glm::vec3(1.f, 0.f, 0.f));
glm::mat4 T2 = glm::translate(glm::mat4(1.f), glm::vec3(0.f, 1.f, 0.f));
glm::mat4 T3 = glm::translate(glm::mat4(1.f), glm::vec3(0.f, 0.f, 1.f));
glm::vec4 pos(2.f, 0.f, 0.f, 1.f);
glm::vec4 newPos = T3 * T2 * T1 * pos;
glm::vec4 oldPos1 = glm::inverse(T1) * glm::inverse(T2) * glm::inverse(T3) * newPos;
glm::vec4 oldPos2 = glm::inverse(T3 * T2 * T1) * newPos;

glUniformMatrix4fv(glGetUniformLocation(program, "uniform_matrix"); // the uniform location
      1,                        // number of matrices 4x4
      GL_FALSE,                 // apply transpose op.
      glm::value_ptr(T3));      // pointer to the first element
```

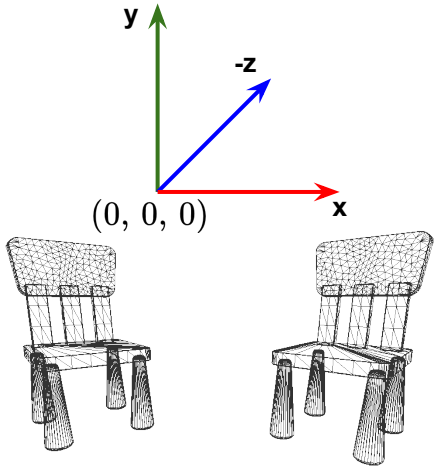
Transformations

Object space

- A geometric object consist of primitives (triangles)
  - Primitives consist of vertices
- Every object has each own local coordinate system

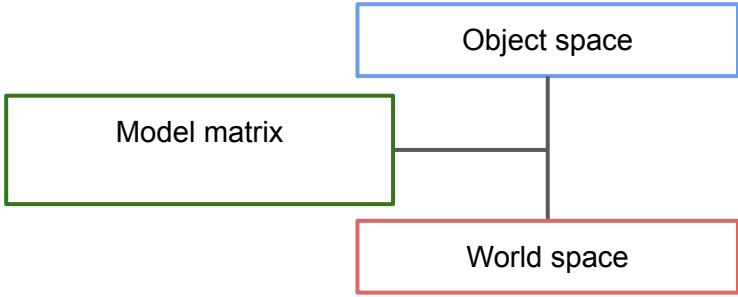


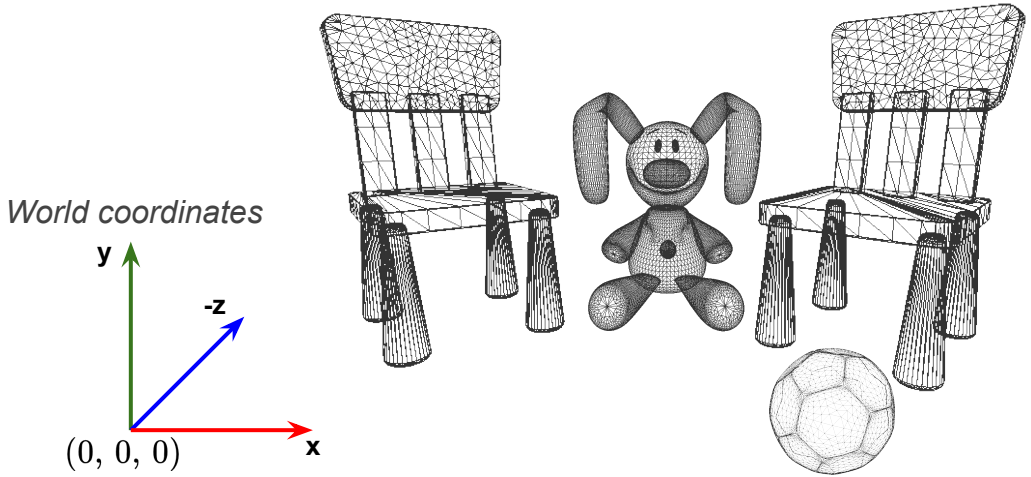




Transformations

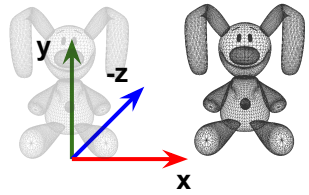
- Each object defines a model matrix
  - Object -> World transformation



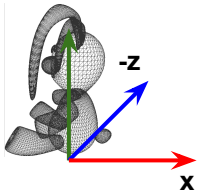


Transformations

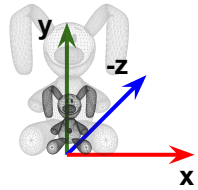
- Each object defines a model matrix
  - Object -> World transformation
- Translate, Rotate, Scale each object



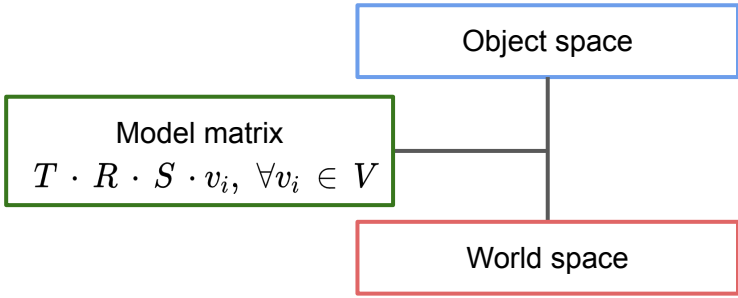

```
glm::mat4 T = glm::translate(glm::mat4(1.f),
       vec3(4.f, 0.f, 0.f)); // units to translate
```




```
glm::mat4 R = glm::rotate(glm::mat4(1.f),
       glm::radians(-90.f),   // radians to rotate [0, 2pi]
       vec3(0.f, 1.f, 0.f));  // rotation axis
```

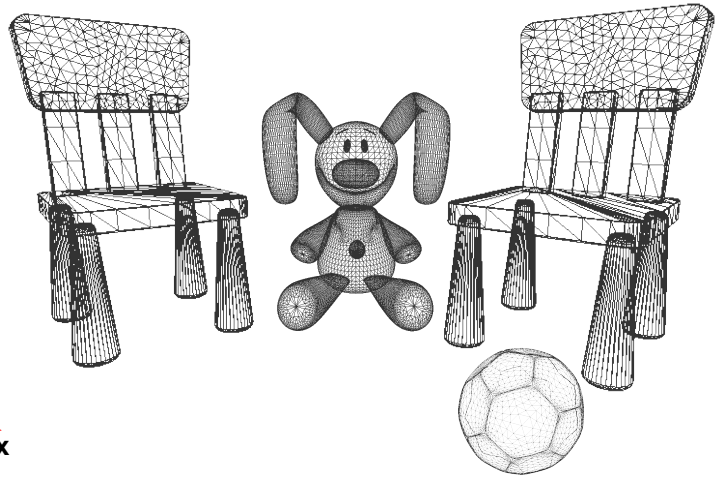


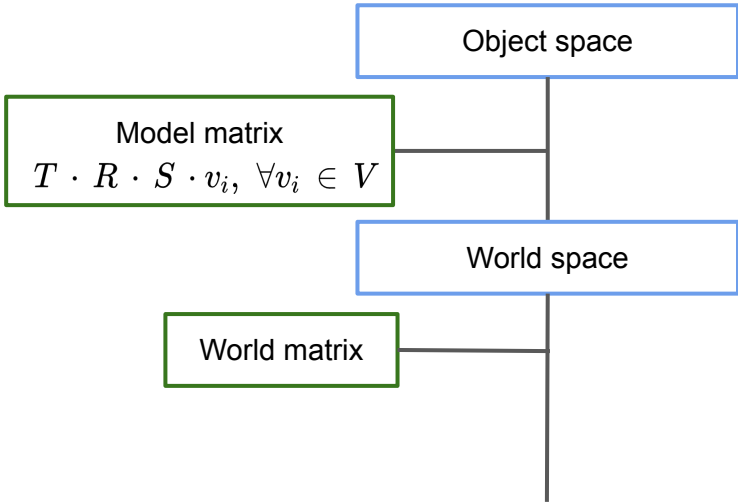

```
glm::mat4 S = glm::scale(glm::mat4(1.f),
       vec3(0.5f, 0.5f, 0.5f)); // scaling factor for each axis
```



Transformations

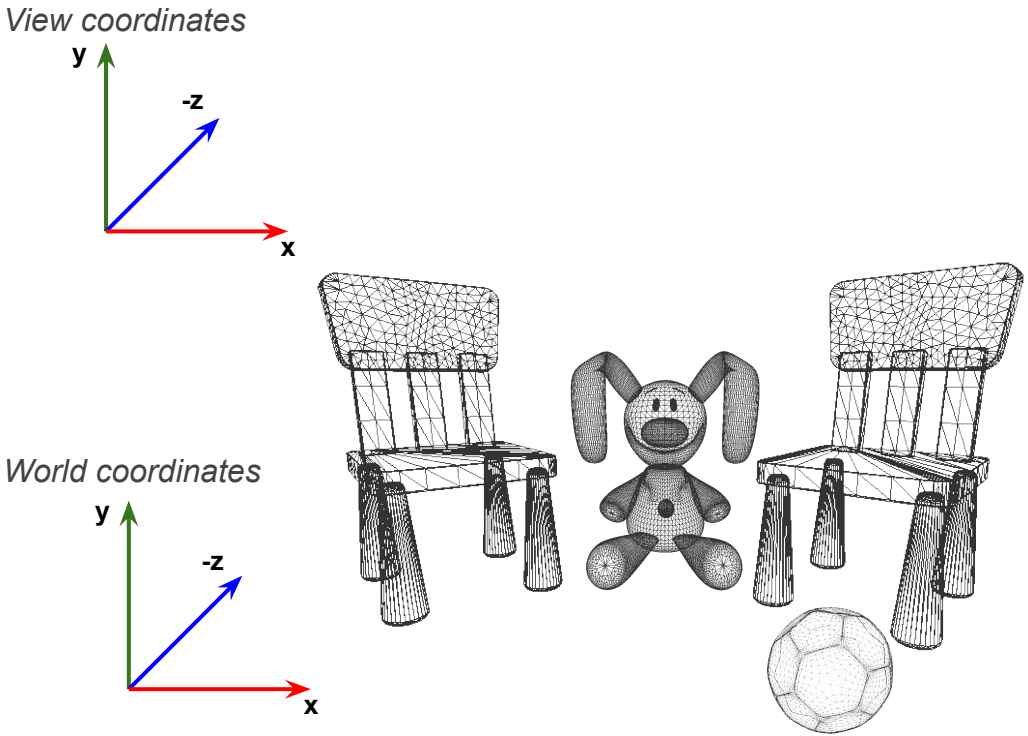
- Each object defines a model matrix
  - Object -> World transformation
- Translate, Rotate, Scale each object
- A global coordinate system world space
  - Manipulate scene as a whole

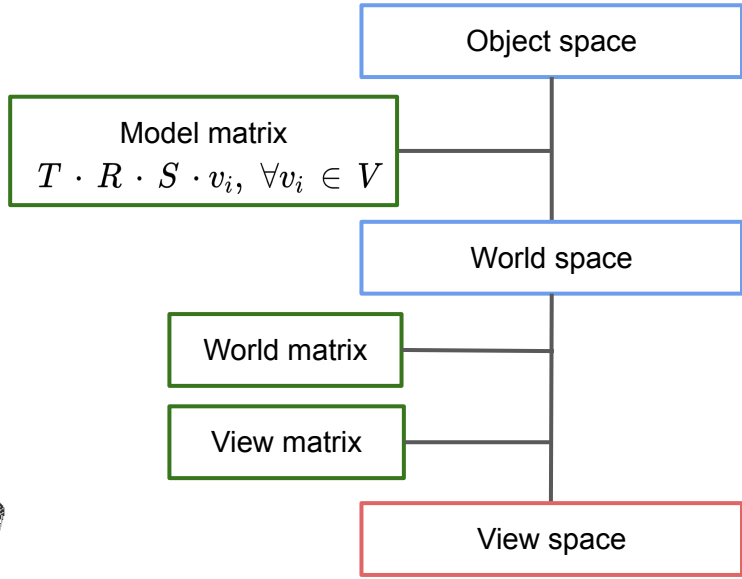




Transformations

- The observer has each own coordinate system
  - World -> View transformation

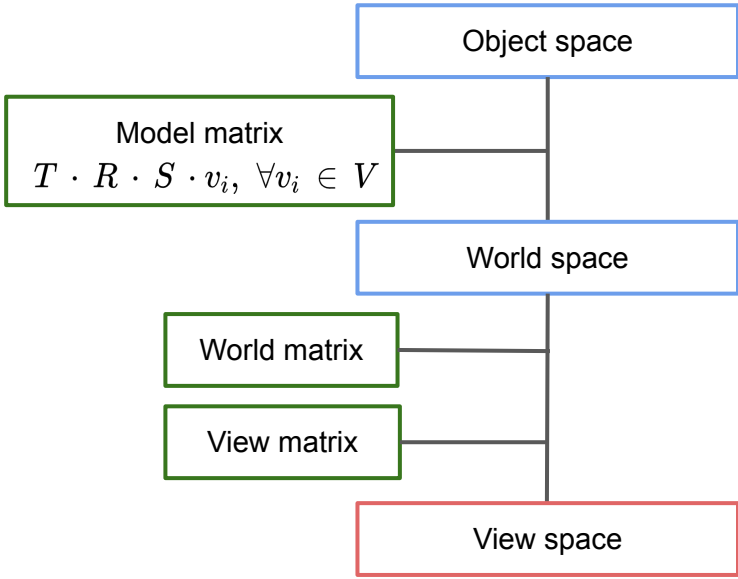




Transformations

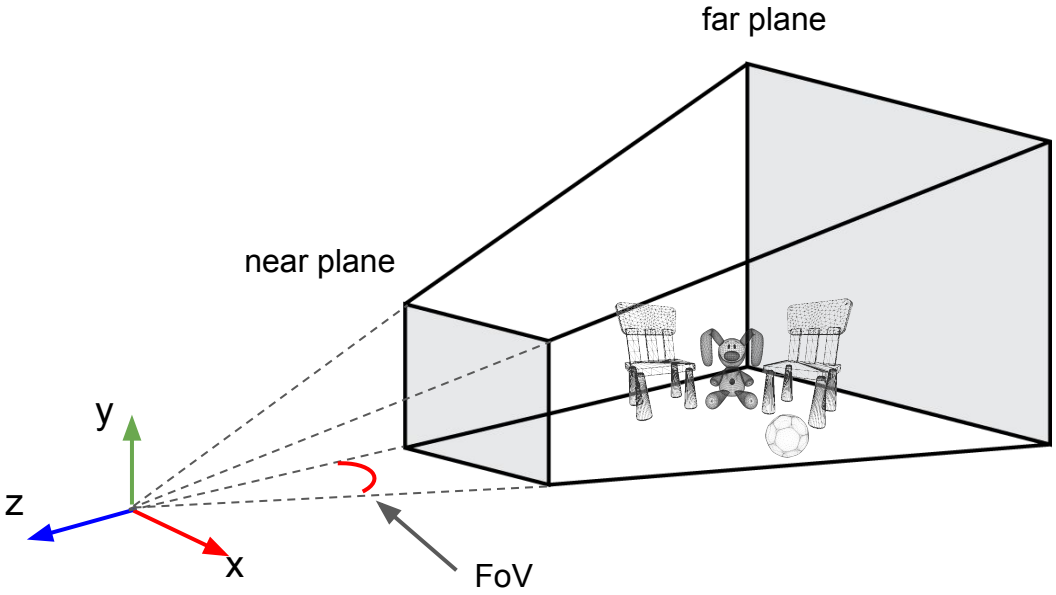
- The observer has each own coordinate system
  - World -> View transformation

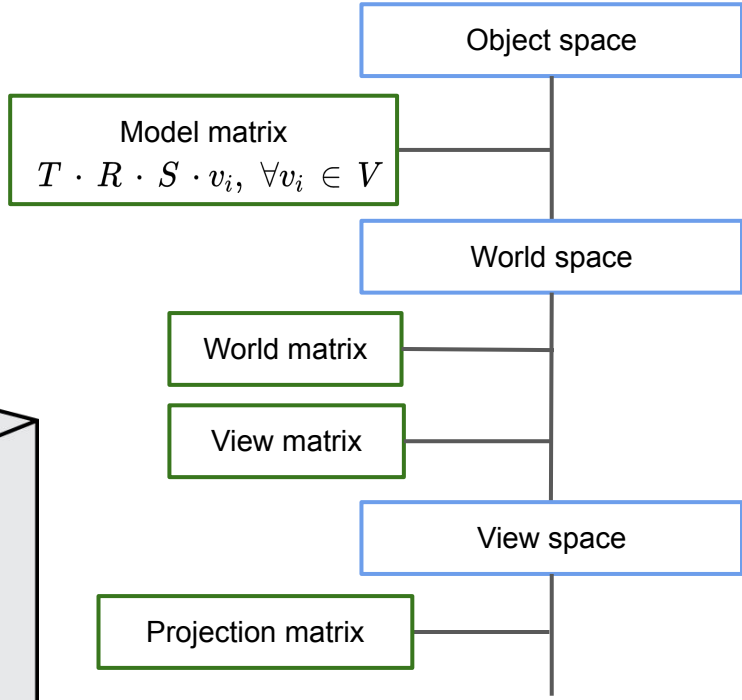
```
glm::vec3 camera_position = glm::vec3(0.f, 5.f, 6.f);
glm::vec3 camera_target = glm::vec3(0.f, 0.f, 0.f);
glm::vec3 camera_up_vector = glm::vec3(0.f, 1.f, 0.f);

glm::mat4 view_matrix = glm::lookAt(
       camera_position,
       camera_target,
       camera_up_vector);

glm::mat4 model_to_view_space = view_matrix * world_matrix * model_matrix;
```



Transformations

- The observer has each own coordinate system
  - World -> View transformation
- Defines a frustum

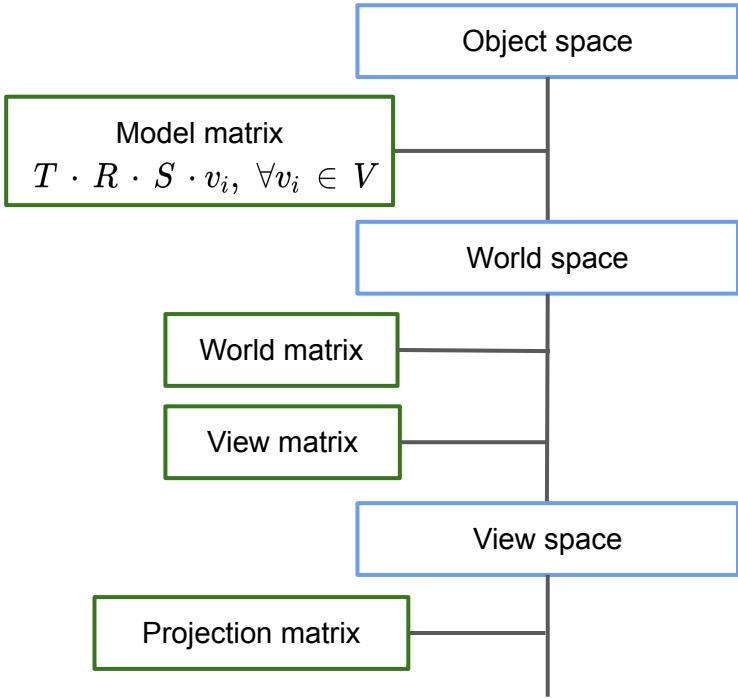




Transformations

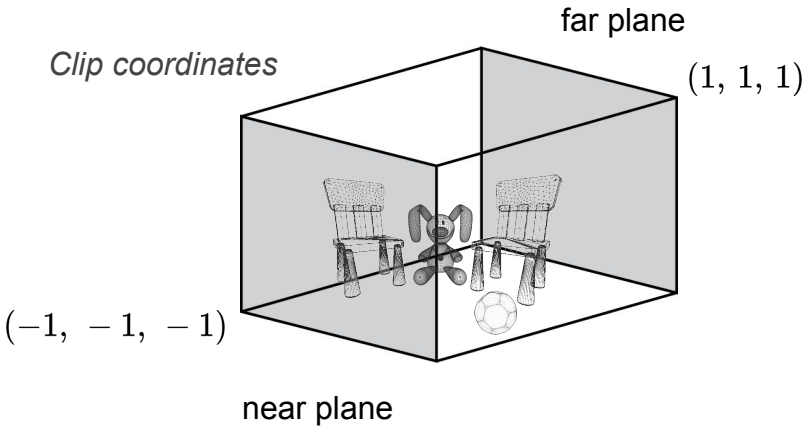
- The observer has each own coordinate system
  - World -> View transformation
- Defines a frustum

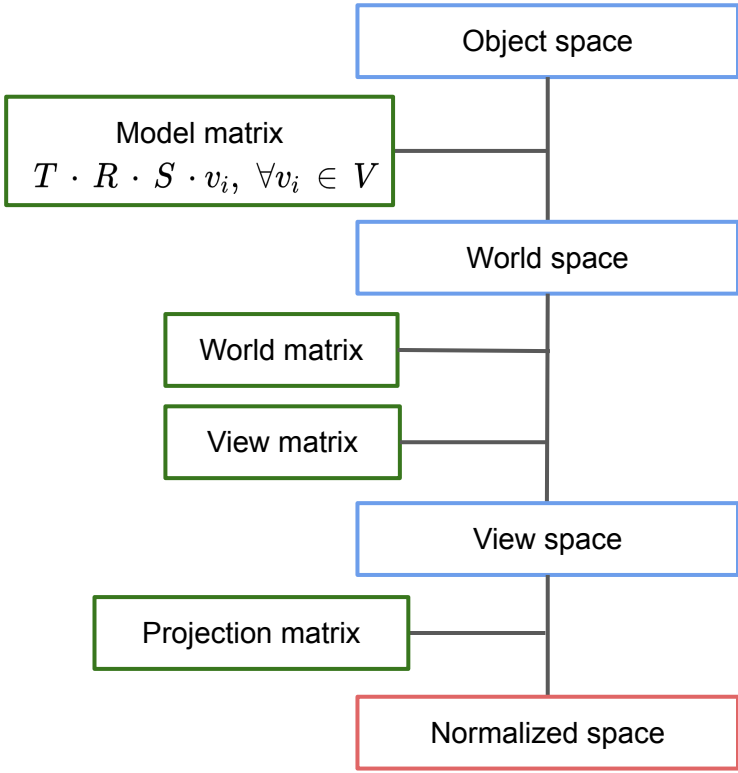
```
float FoV = glm::radians(45.f);
float aspect_ratio = width / (float) height;
float near_plane = 0.1f;
float far_plane = 10.f;

glm::mat4 proj_matrix = glm::perspective(
        foV,
        aspect_ratio,
        near_plane, far_plane);

glm::mat4 model_to_proj_space =
        proj_matrix * view_matrix * world_matrix * model_matrix;
```



Transformations

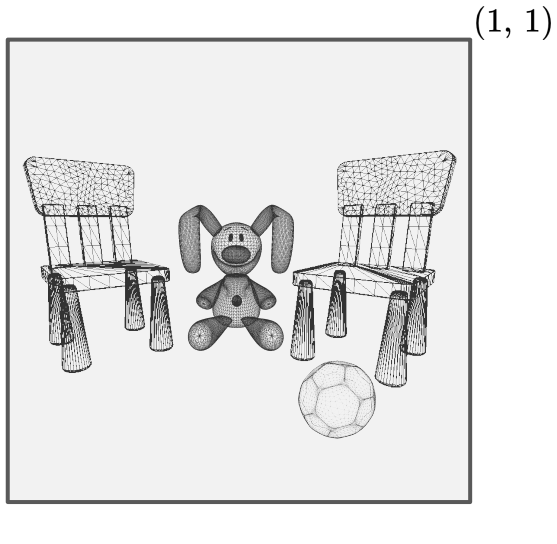
- The observer has each own coordinate system
  - World -> View transformation
- Defines a frustum
- Primitives are clipped
  - against the frustum planes
  - back facing triangles are discarded (optional)

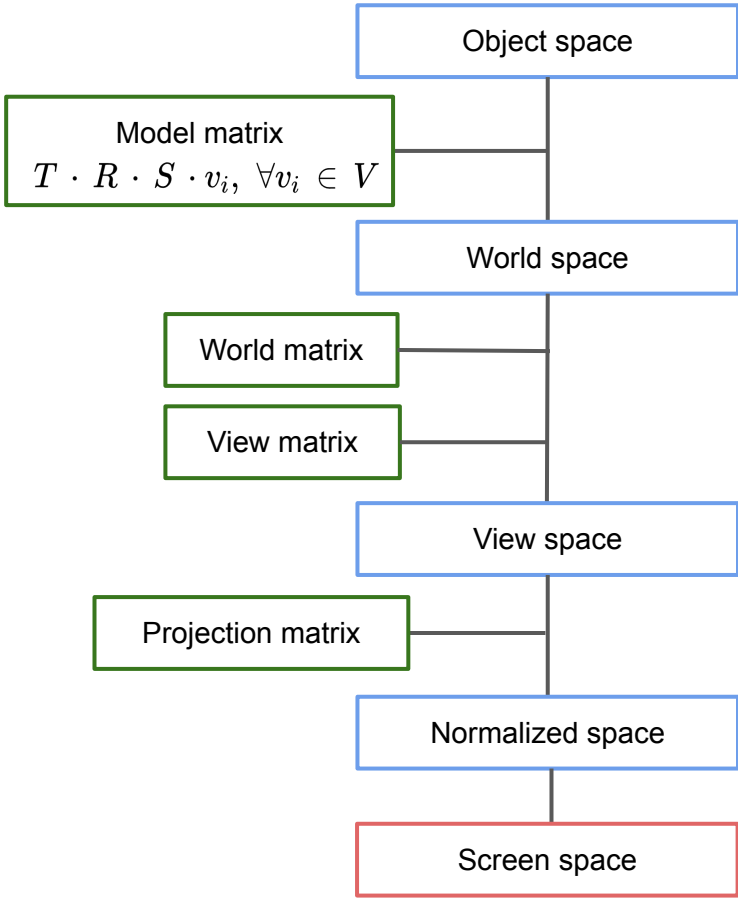




Transformations

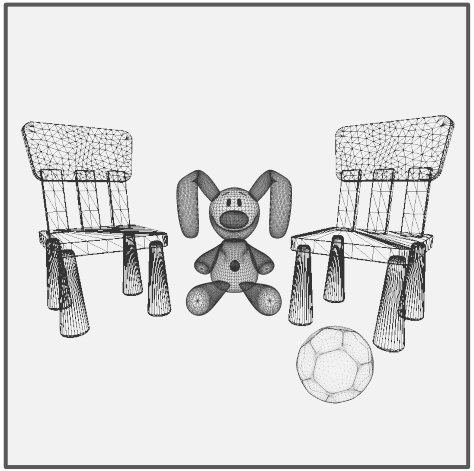
- Clipped primitives are projected onto the image plane
  - Space bounds [-1, -1] x [1, 1]

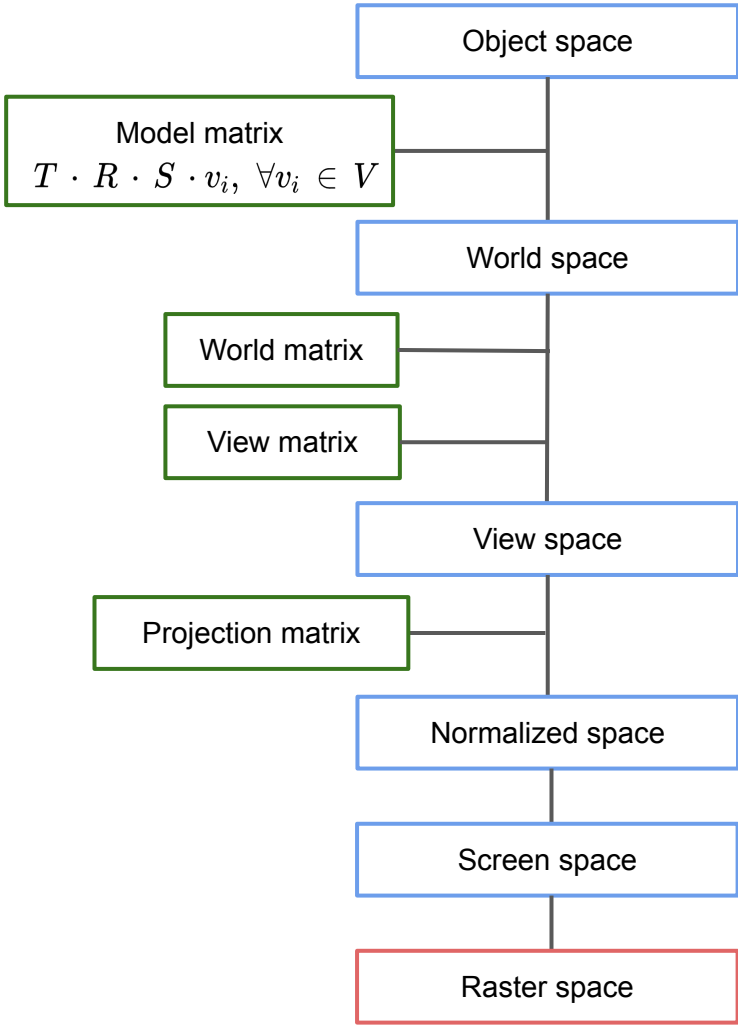




Transformations

- Rescaled to raster/image space
  - Space bounds [0, 0] x [width, height]





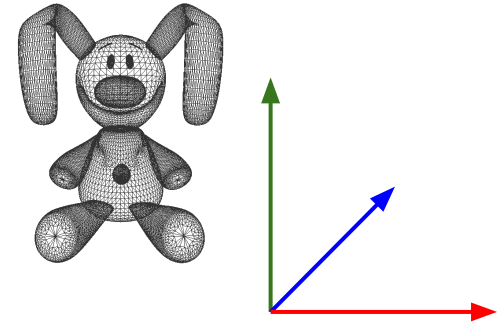
Examples

```
glm::mat4 T = glm::translate(glm::mat4(1.f), glm::vec3(3, 2, 1));
glm::mat4 S = glm::scale(glm::mat4(1.f), glm::vec3(4, 4, 4));
glm::mat4 ST = S * T;
glm::mat4 TS = T * S;
glm::vec4 pos(1, 1, 1, 1);
```

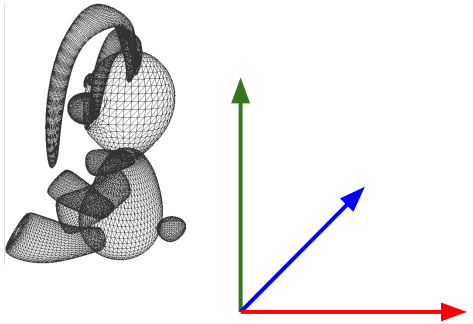
$$STv = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 16 \\ 12 \\ 8 \\ 1 \end{bmatrix}$$

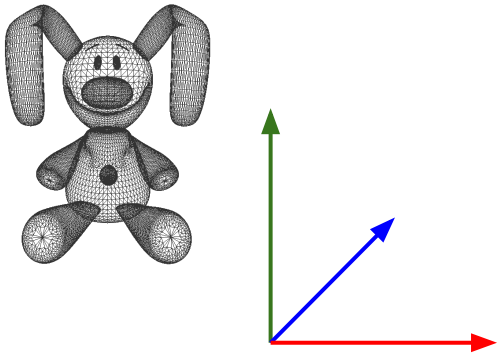
$$TSv = \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 4 \\ 4 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 7 \\ 6 \\ 5 \\ 1 \end{bmatrix}$$

Examples

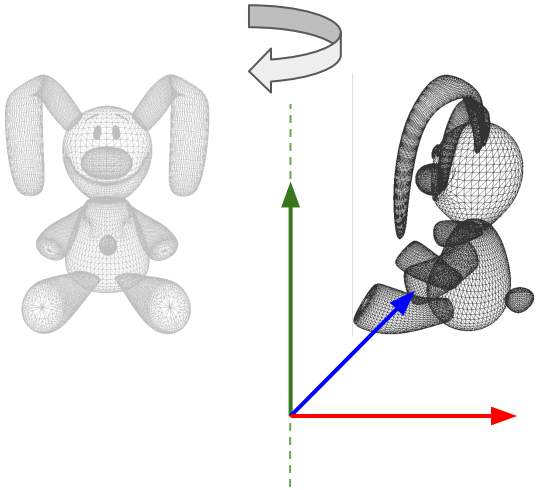


?



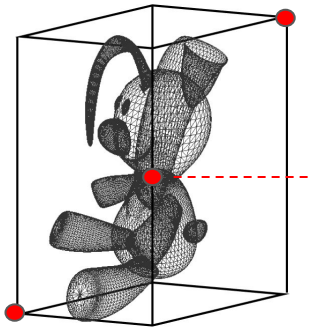


```
glm::mat4 R = glm::rotate(glm::mat4(1.f),
        glm::radians(-90.f),
        vec3(0.f, 1.f, 0.f));
```



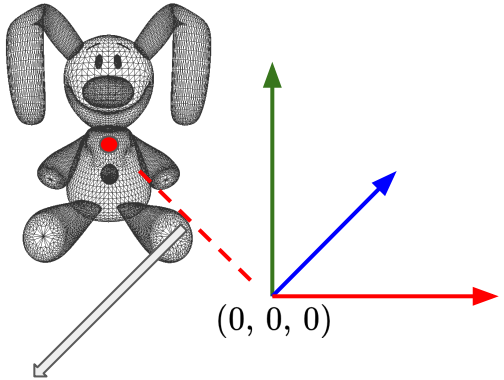
Examples

$box_{\max} = [x_{\max}, y_{\max}, z_{max}]$



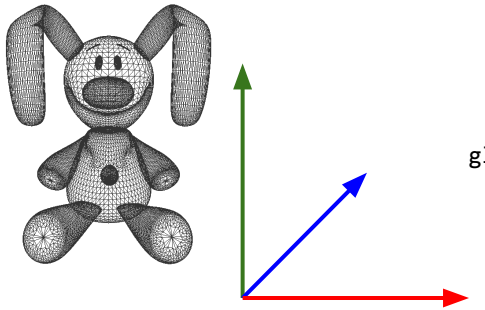
$center = \frac{1}{2}(box_{\min} + box_{\max})$

$box_{\min} = [x_{\min}, y_{\min}, z_{min}]$

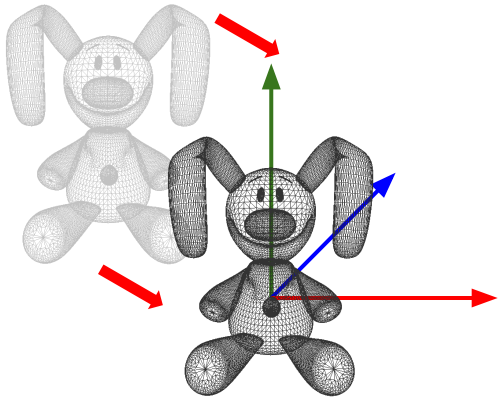


$d = [0, 0, 0] - center$

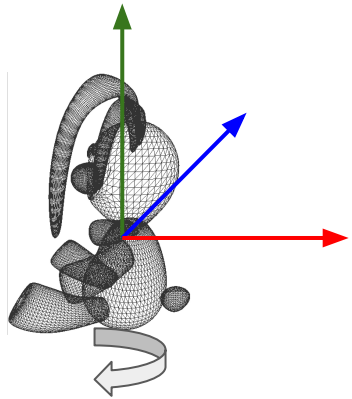
Examples



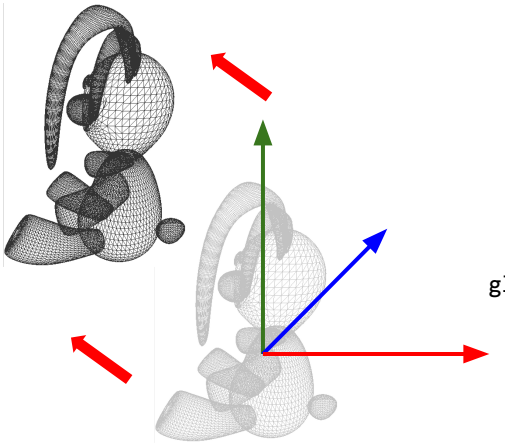
```
glm::mat4 T1 = glm::translate(glm::mat4(1.f), -aabb.center);
```



```
glm::mat4 R = glm::rotate(glm::mat4(1.f),
        glm::radians(-90.f),
        vec3(0.f, 1.f, 0.f));
```

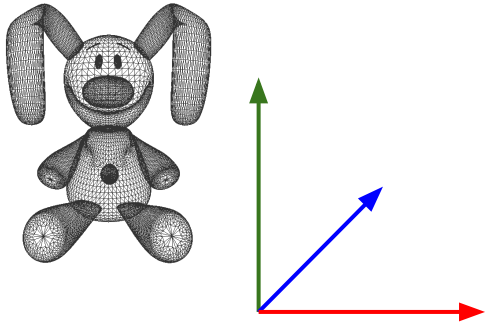


```
glm::mat4 T2 = glm::translate(glm::mat4(1.f), aabb.center);
```

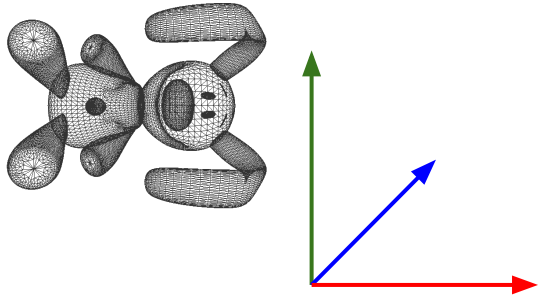


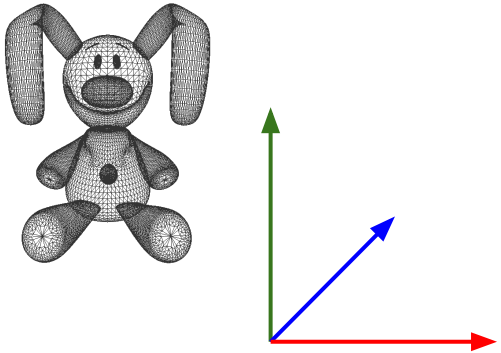
Presentation co-author Anastasios Gkaravelis

Examples

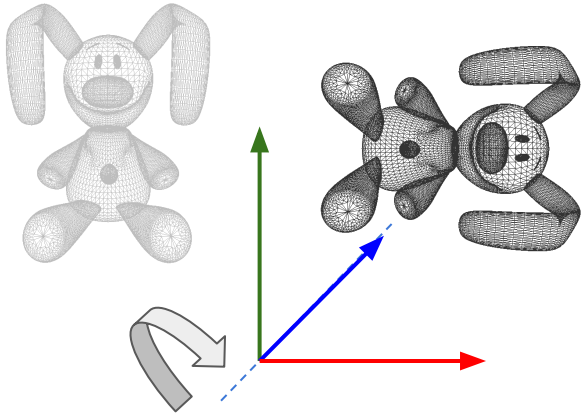


?

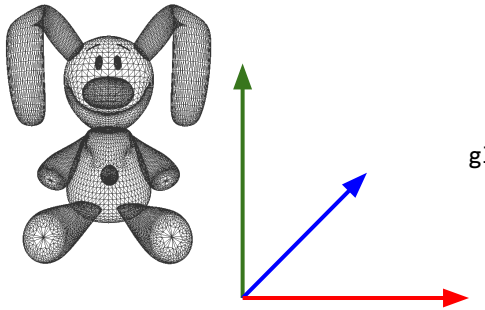




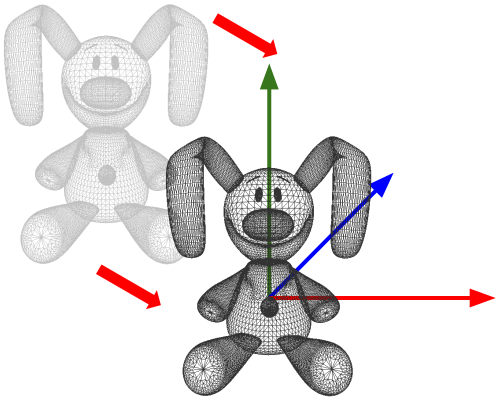
```
glm::mat4 R = glm::rotate(glm::mat4(1.f),
        glm::radians(-90.f),
        vec3(0.f, 0.f, 1.f));
```



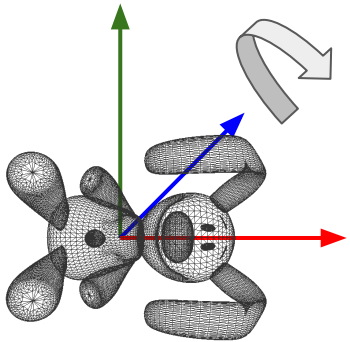
Examples



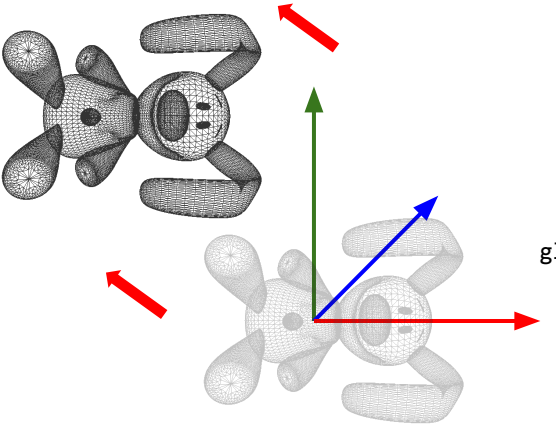
```
glm::mat4 T1 = glm::translate(glm::mat4(1.f), -aabb.center);
```



```
glm::mat4 R = glm::rotate(glm::mat4(1.f),
        glm::radians(-90.f),
        vec3(0.f, 0.f, 1.f));
```



```
glm::mat4 T2 = glm::translate(glm::mat4(1.f), aabb.center);
```



Presentation co-author Anastasios Gkaravelis

Task

- Construct a first-person camera using :
  - Translations transformations for button movements
  - Rotations transformations for mouse motion movements
    - Optionally, derive an exact calculation for the radians parameter