UFABC

Computação Gráfica

André Brandão

Aula 15

Aula prática

Matrizes

Última aula

- Utilização de diferentes funções do OpenGL.
  - GL_TRIANGLES
  - GL_QUADS
  - GL_POINTS
  - GL_LINES
- Para cada uma das funções que serão aplicadas, crie um projeto novo, assim, cada uso das funções ficará em um projeto diferente.

Aula 15

- Trabalhar com matrizes de transformação

Coordenadas homogêneas

- Até então, consideramos apenas vértices 3D como (x, y, z). Vamos introduzir w. Agora teremos o vetor (x, y, z, w).
- Por enquanto, lembre-se:
  - Se w == 1, então o vetor (x, y, z, 1) é uma posição no espaço.
  - Se w == 0, então o vetor (x, y, z, 0) é uma direção.

Coordenadas homogêneas

- Para uma rotação, não muda nada. Quando você gira um ponto ou uma direção, você obtém o mesmo resultado.
- No entanto, para uma translação, as coisas são diferentes. O que poderia significar "transladar numa direção"? Não muito.
- As coordenadas homogêneas permitem usar uma única fórmula matemática para lidar com esses dois casos.

Matrizes

- Na computação gráfica em 3D, vamos usar principalmente matrizes 4x4. Elas nos permitirão transformar nossos vértices (x, y, z, w). Isto é feito multiplicando o vértice com a matriz:

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

Matrizes

- O trabalho com matrizes pode ser facilitado com o uso de bibliotecas específicas, como é o caso da GLM.

```
glm::mat4 myMatrix;
glm::vec4 myVector;
// fill myMatrix and myVector somehow
glm::vec4 transformedVector = myMatrix * myVector;
```

Matrizes de translação

- Estas são as matrizes de transformação mais simples de entender. Uma matriz de translação tem esta aparência:

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

onde, X, Y, Z são os valores que você deseja adicionar à sua posição.

Matrizes de translação

- Portanto, se queremos transladar o vetor (10,10,10,1) de 10 unidades na direção X, obtemos:

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

Matrizes de translação

- E obtemos um vetor homogêneo (20,10,10,1).
- Lembre-se, o 1 significa que é uma posição, não uma direção. Assim, nossa transformação não mudou o fato de que estávamos lidando com uma posição.

Matrizes de translação

- Vejamos agora o que acontece com um vetor que representa uma direção para o eixo -z: (0,0, -1,0):

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

Matrizes de translação

- Então, como isso se traduz em código?

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

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

Matriz identidade

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

Matriz identidade

- Em código:

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

Matrizes de escala

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

Matrizes de escala

- Então, se você quiser escalar um vetor (posição ou direção) por 2, em todas as direções:

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

- Em C++, o código fica da seguinte forma:

UFABC

Matrizes de escala

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

UFABC

Matrizes de rotação

- Em C++:

```
// Use #include <glm/gtc/matrix_transform.hpp> and #include <glm/gtx/transform.hpp>
glm::vec3 myRotationAxis( ??, ??, ??);
glm::rotate( angle_in_degrees, myRotationAxis );
```

Transformações cumulativas

- Sabemos como rotacionar, trasladar e escalar nossos vetores.
- É possível combinar essas transformações. Isto é feito multiplicando as matrizes juntas, por exemplo:

```
TransformedVector = TranslationMatrix * RotationMatrix * ScaleMatrix * OriginalVector;
```

Transformações cumulativas

- Esta linha executa a **escala**, primeiro, e então a **rotação**, e então a **translação**. É assim que funciona a multiplicação de matrizes.

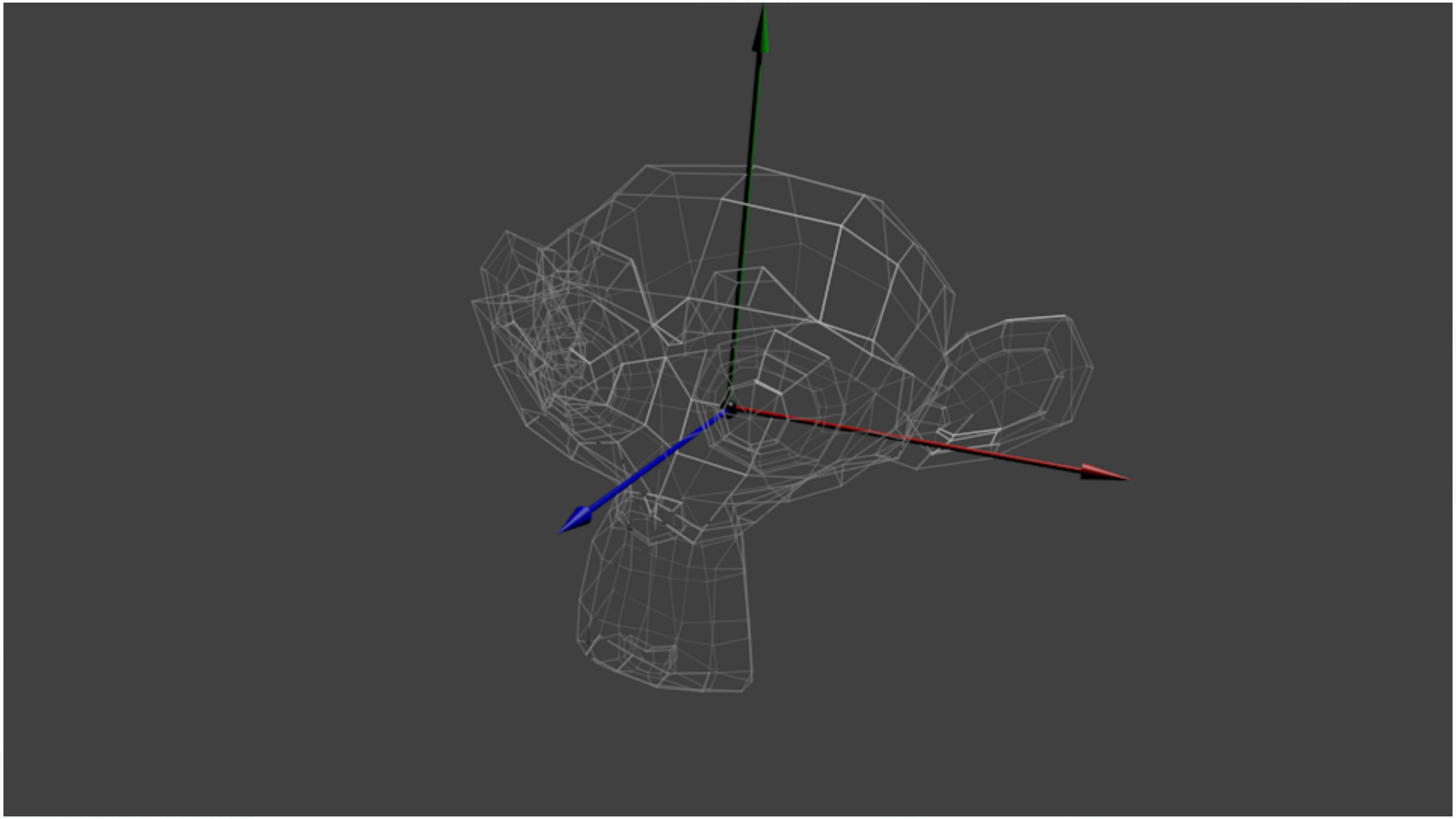
Matrizes de modelo, visualização e projeção

- As matrizes modelo, visualização e projeção são ferramentas úteis para separar as transformações de forma limpa.
- Esta é a maneira que fazemos, porque é a maneira mais fácil.

Matriz de modelo (*Model*)

- Modelos, tal como o exemplo do triângulo, é definido por um conjunto de vértices.
- As coordenadas X, Y, Z desses vértices são definidas em relação ao centro do objeto: isto é, se um vértice estiver em (0,0,0), ele está no centro do **objeto**.

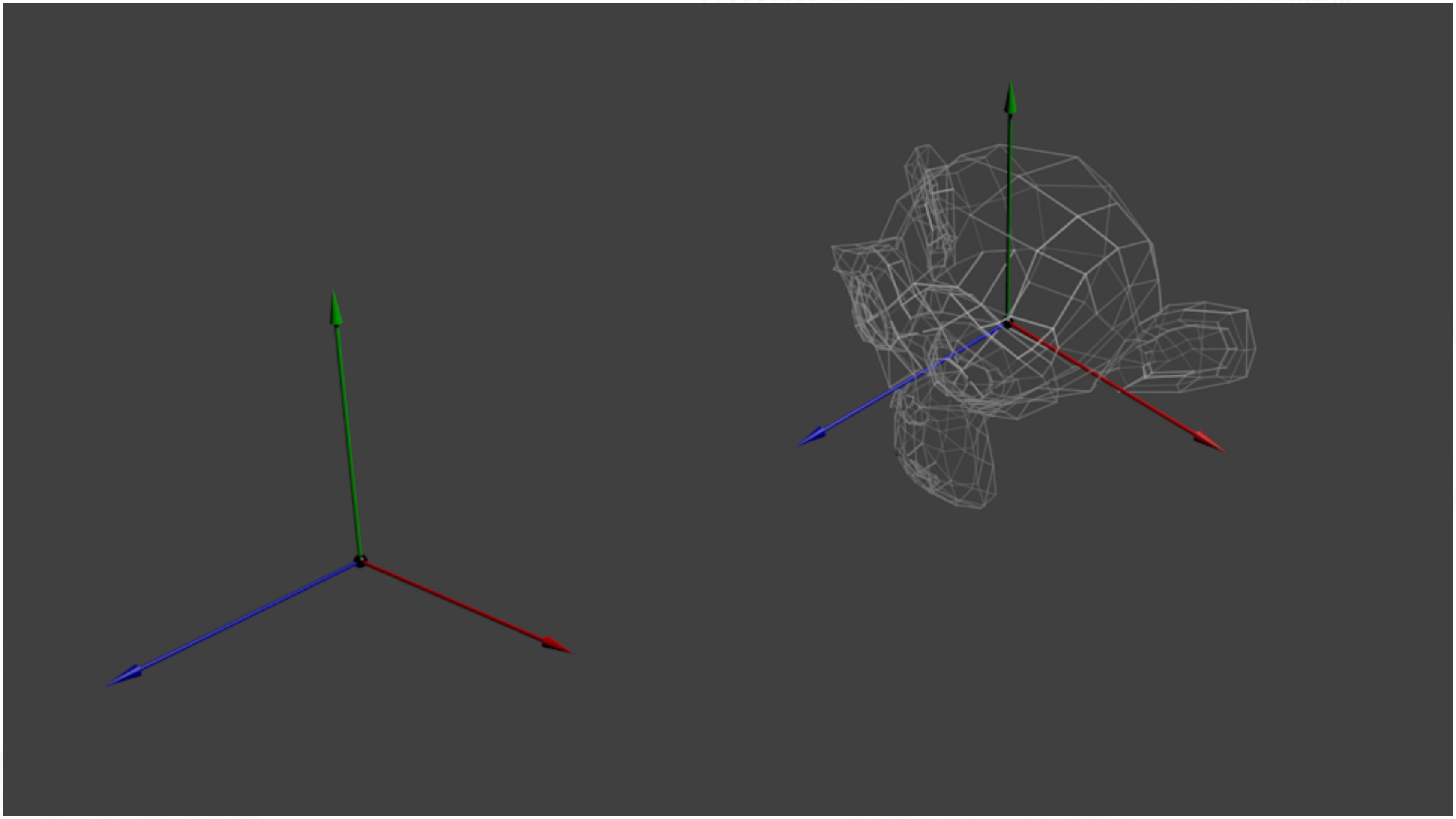
Matriz de modelo (*Model*)



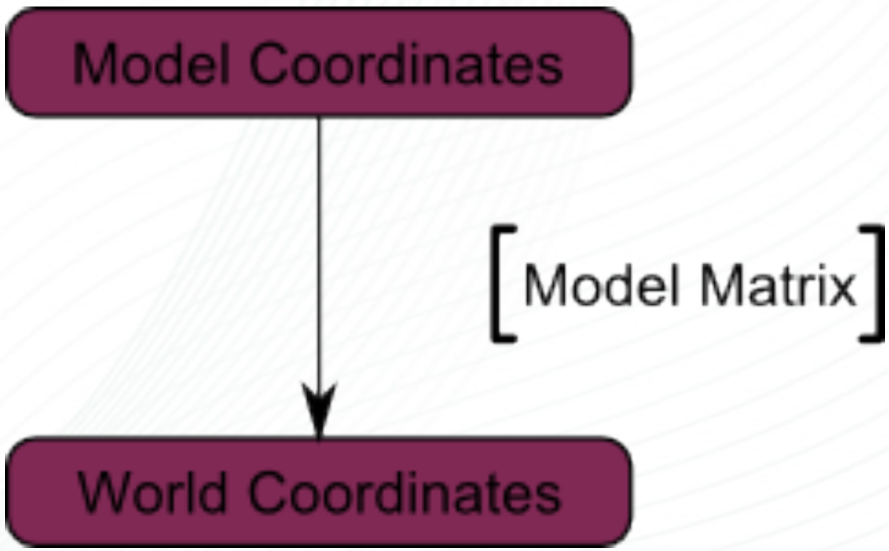
Matriz de modelo (*Model*)

- É possível de se mover este modelo (controla-lo com o teclado e o mouse).
- Isso é possível por meio da multiplicação translação * rotação * escala. Você aplica essa matriz a todos os seus vértices em cada quadro e tudo se move.
- Se algo não se mover, então esse algo estará no centro *do mundo (coordenadas de mundo)*.

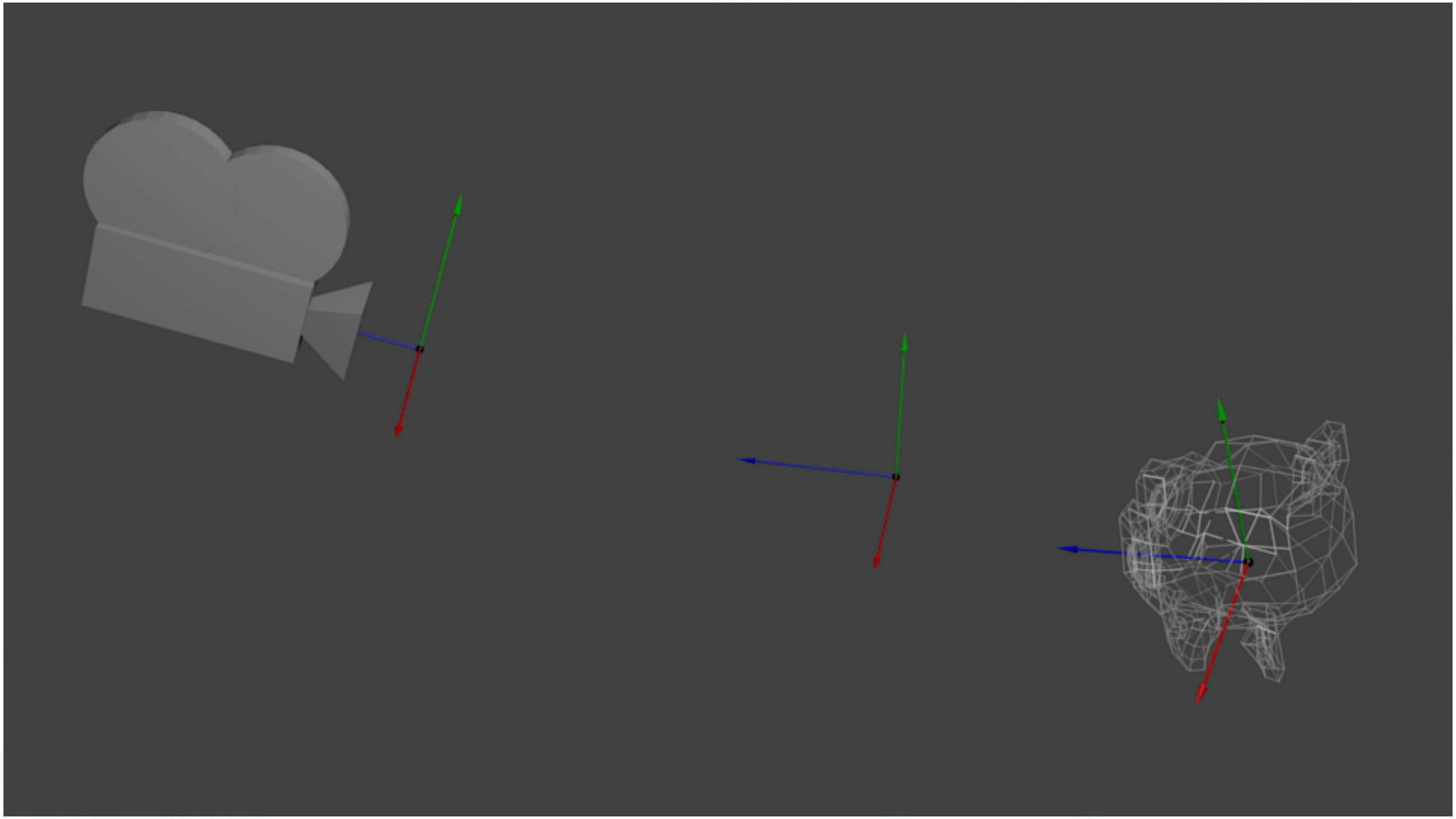
Matriz de modelo (*Model*)



Matriz de modelo (*Model*)



Matriz de visualização



Matriz de visualização

- Se você quiser ver um objeto por outro ângulo, você pode mover a câmera ou mover o objeto, em relação à câmera. Embora isso não aconteça no mundo físico, isso é útil em Computação Gráfica.
- Assim, sua câmera está na origem do espaço de mundo. Para mover o mundo, você simplesmente introduz uma outra matriz. Digamos que você deseja mover sua câmera de 3 unidades para a direita (+ X).

Matriz de visualização

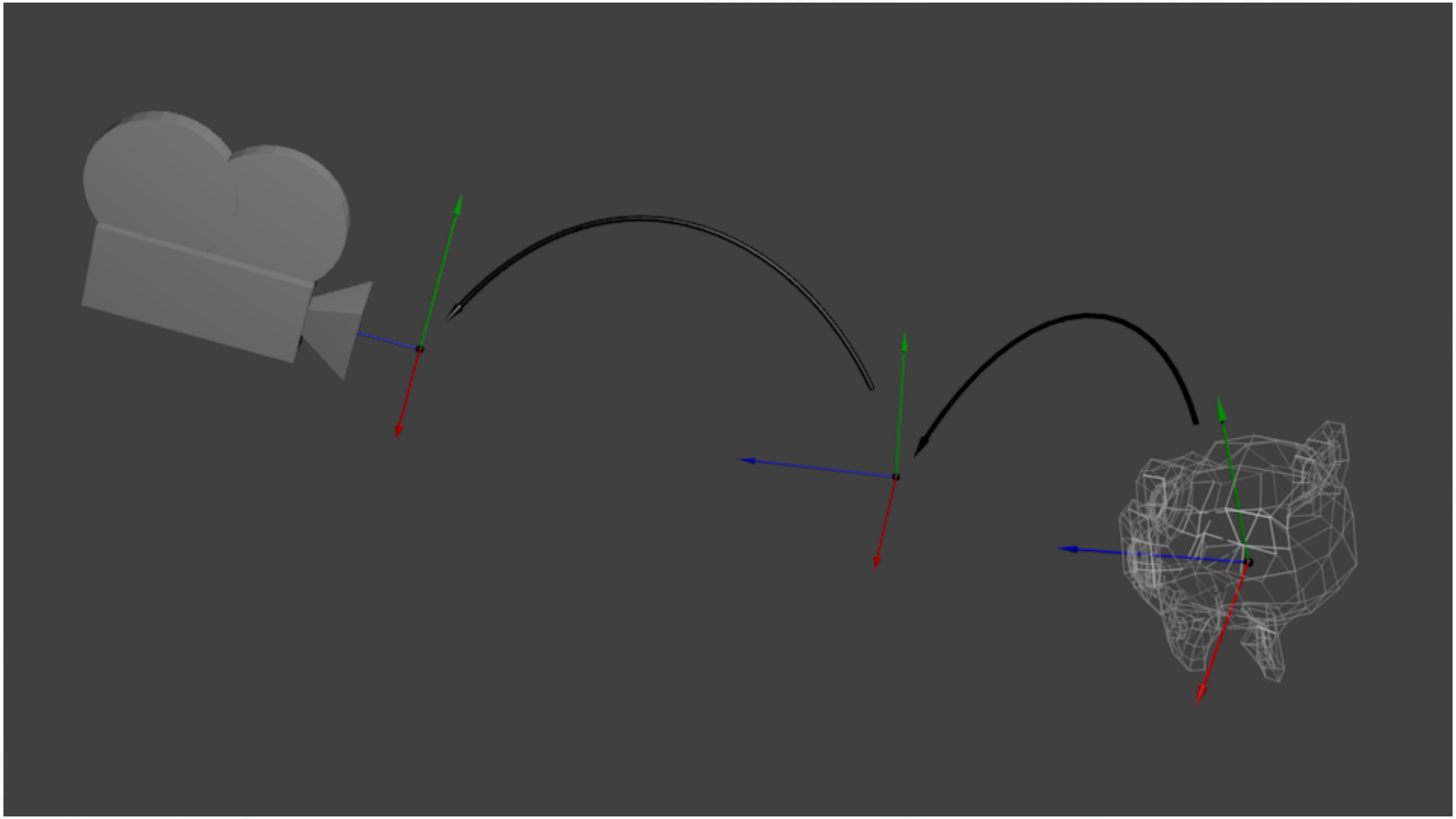
- Isto é equivalente a mover o seu mundo inteiro em 3 unidades para a esquerda (-X).

```
// Use #include <glm/gtc/matrix_transform.hpp> and #include <glm/gtx/transform.hpp>
glm::mat4 ViewMatrix = glm::translate(-3.0f, 0.0f ,0.0f);
```

Matriz de visualização

- A próxima imagem ilustra isto: vamos do espaço de mundo (todos os vértices definidos relativamente ao centro do mundo) ao Espaço da Câmera (todos os vértices são definidos relativamente à câmera).

Matriz de visualização

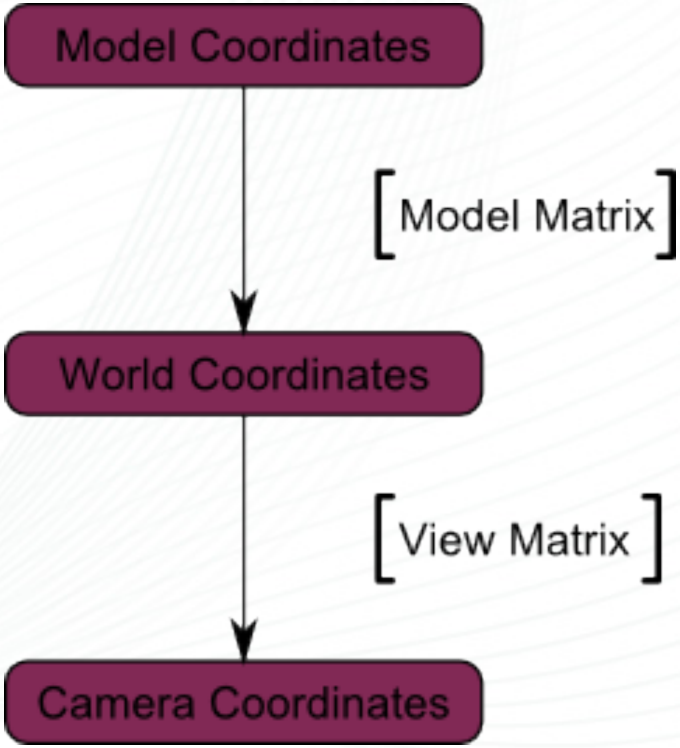


Matriz de visualização

- Para tanto, usa-se uma função, que realiza esse tipo de transformação.

```
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
);
```

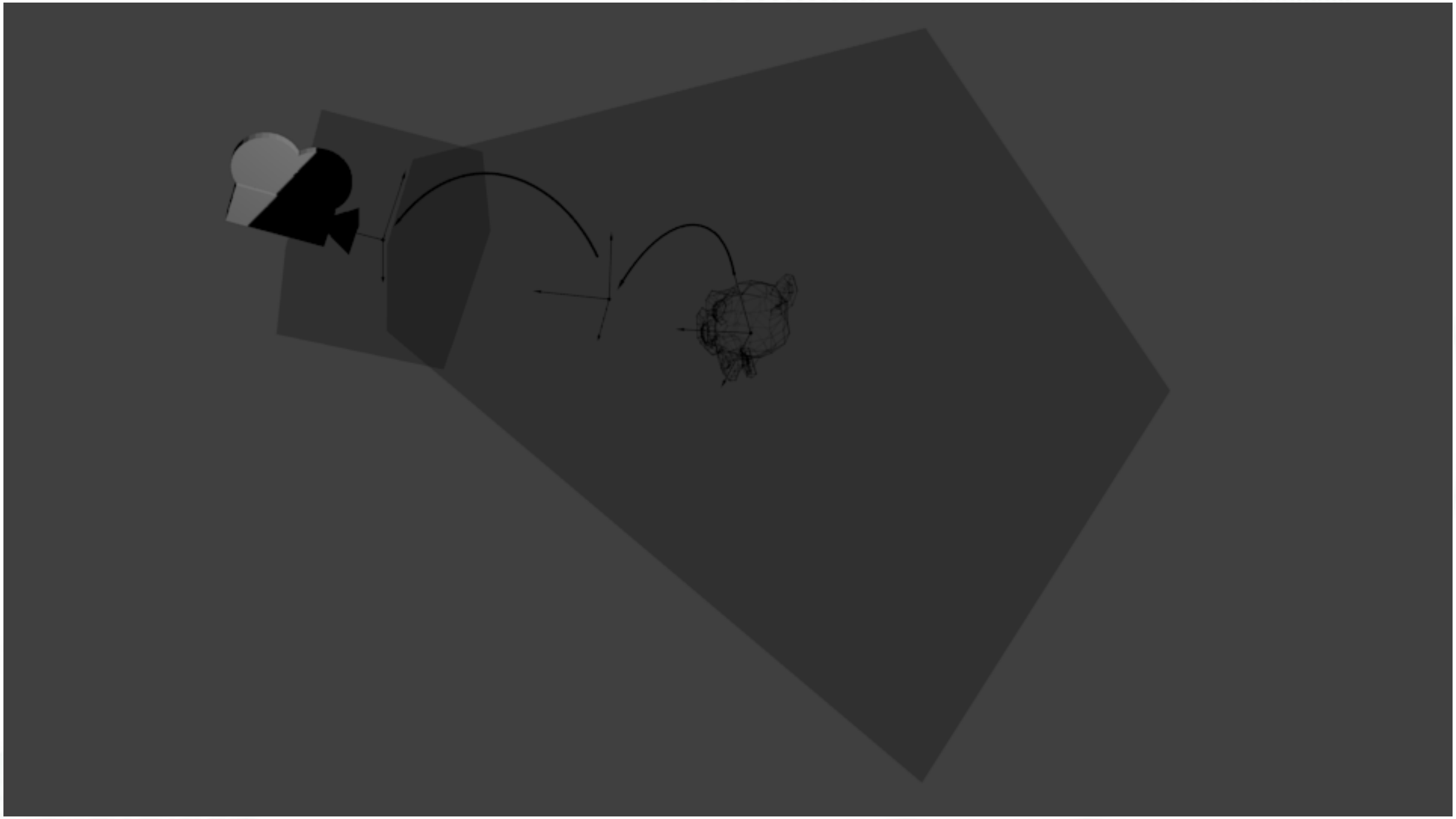
Matriz de visualização



Matriz de projeção

- Até agora, trabalhamos no Espaço da Câmera.
- Após todas as transformações, um vértice que tem valores de x == 0 e y == 0 deve ser renderizado no centro da tela.
- Mas, não podemos usar apenas as coordenadas x e y para determinar onde um objeto deve ser colocado na tela: sua distância para a câmera (z) também é considerada.

Matriz de projeção



Matriz de projeção

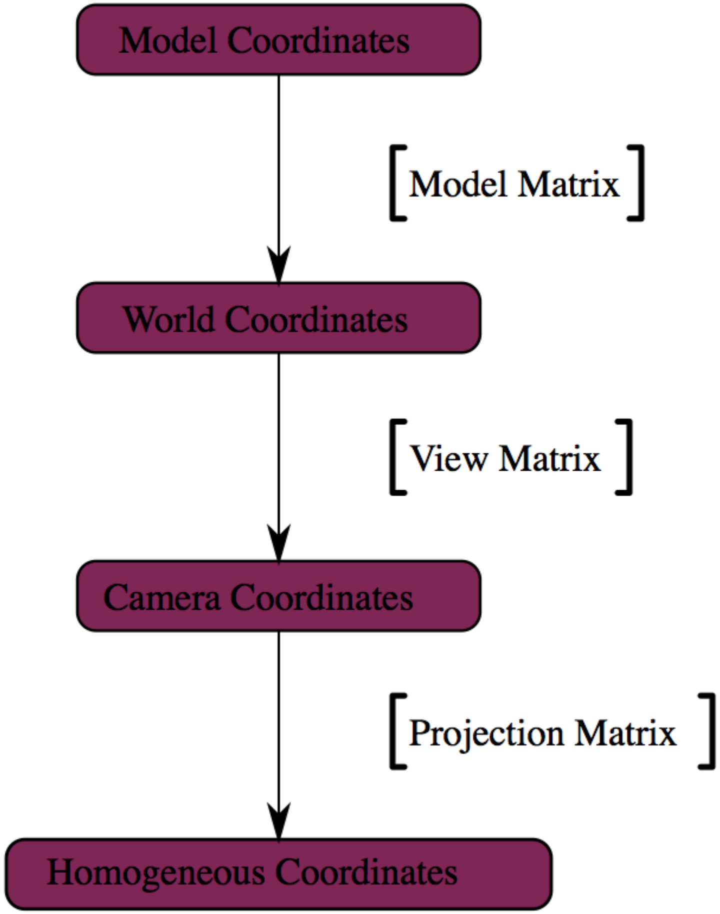
- Uma matriz 4x4 pode representar essa projeção.

```
// Generates a really hard-to-read matrix, but a normal, standard 4x4 matrix nonetheless
glm::mat4 projectionMatrix = glm::perspective(
    FoV,         // The horizontal Field of View, in degrees : 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.
);
```

Matrizes de transformação

- Passamos do Espaço da Câmera (todos os vértices definidos relativamente à câmera) para Espaço Homogêneo (todos os vértices definidos em um pequeno cubo. Tudo dentro do cubo está na tela).

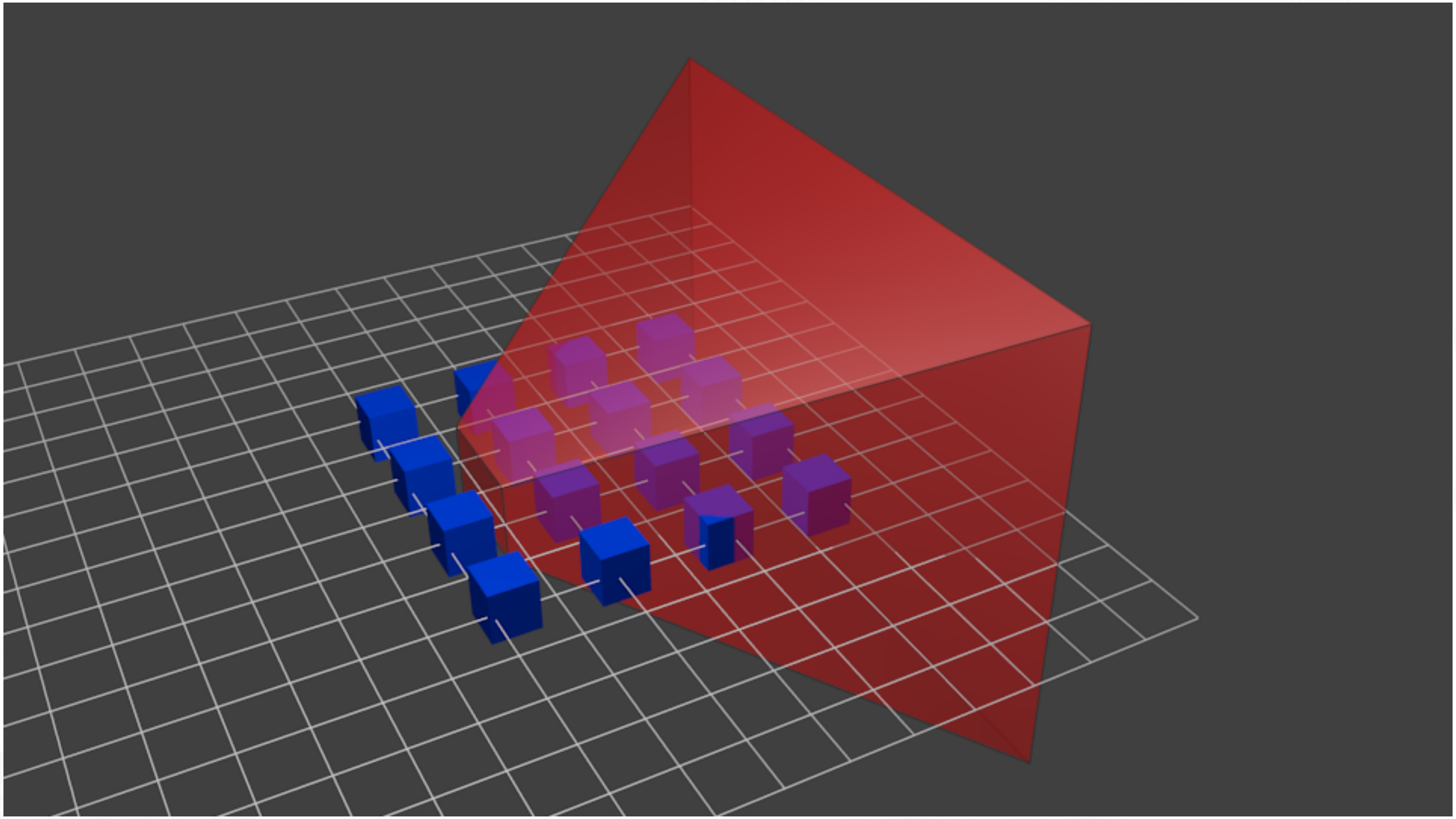
Matrizes de transformação



Matrizes de transformação

- No próximo slide, é apresentada outra forma para que você entenda melhor o que acontece com este material de Projeção.

Matrizes de transformação



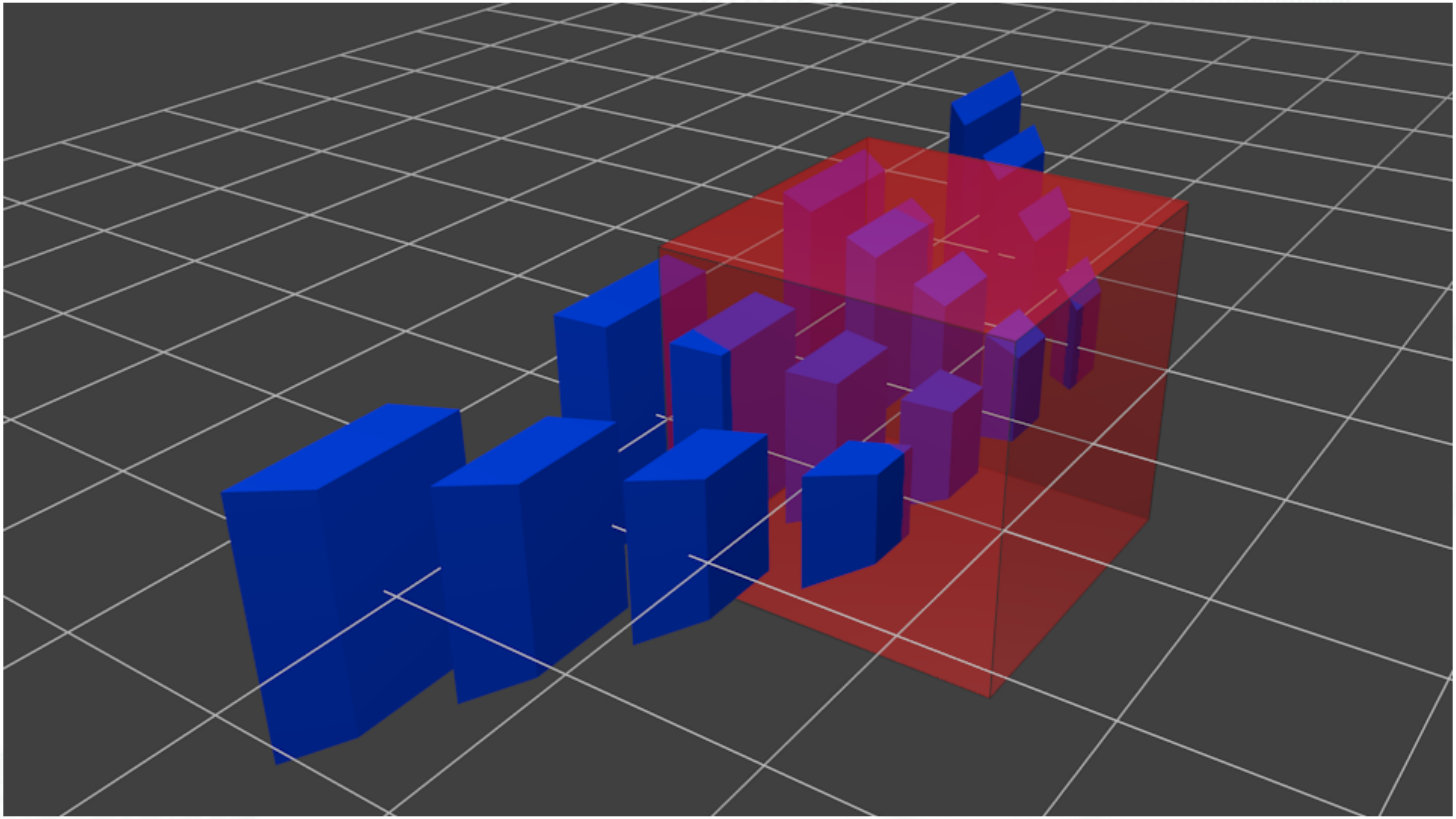
Matrizes de transformação

- Antes da projeção, temos os objetos azuis, no Espaço da Câmera, e a forma vermelha representa o "tronco" da câmera: a parte da cena que a câmera realmente pode ver.

Matrizes de transformação

- Multiplicar tudo pela Matriz de Projeção tem o seguinte efeito:

Matrizes de transformação



Matrizes de transformação

- O "tronco" é um cubo perfeito (entre -1 e 1 em todos os eixos), e todos os objetos azuis foram deformados da mesma maneira.
- Os objetos que estão perto da câmera (perto da face do cubo que não podemos ver) são grandes, os outros são menores.

Matrizes

Parte prática

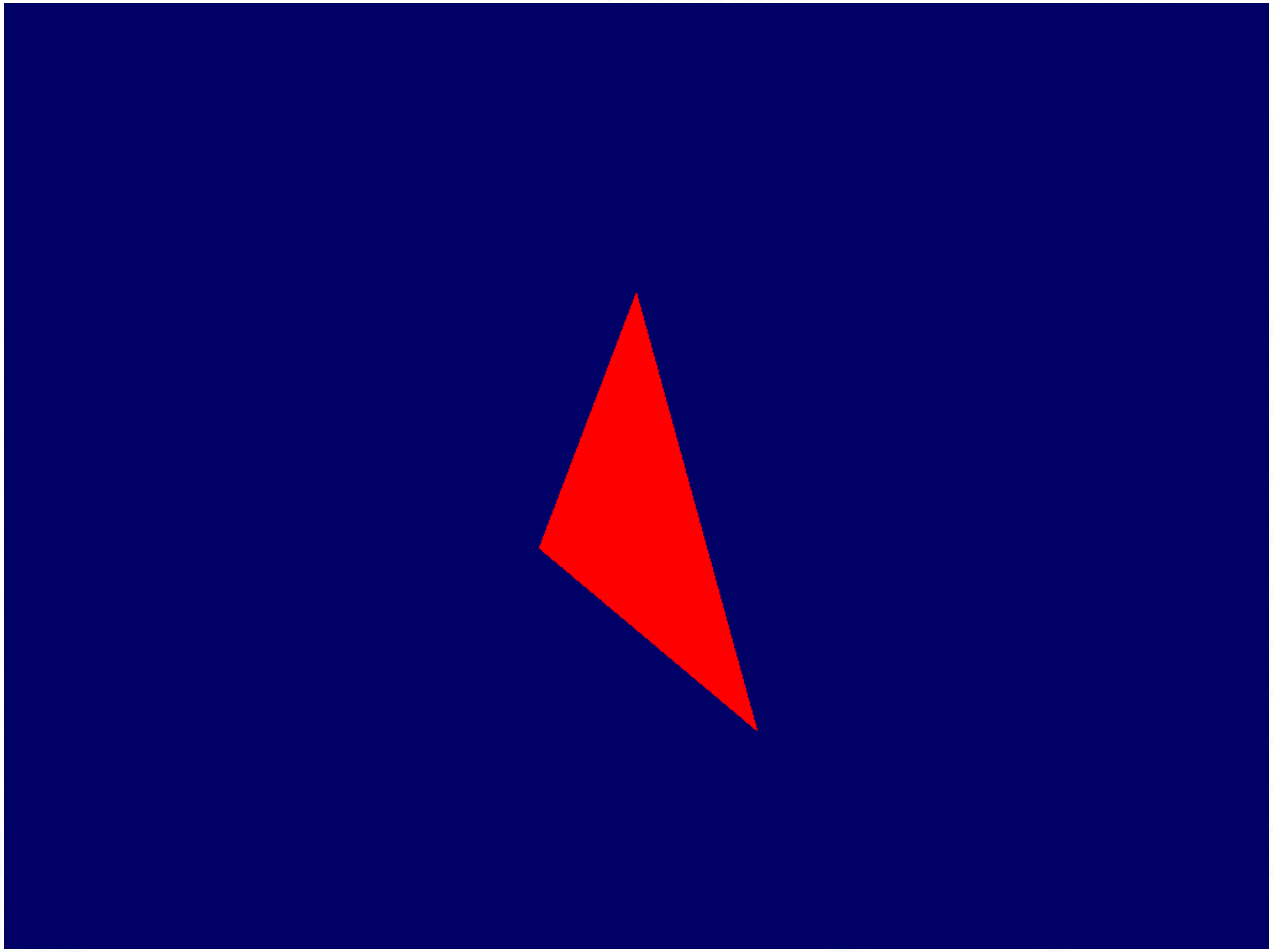
Matrizes

- Faça os Tutoriais 1, 2 e 3;
- Baixe os seguintes arquivos que estão na ferramenta Repositório, do TIDIA, na pasta CG-Aula15:
  - main.cpp, glm.zip, shader.cpp, shader.hpp, SimpleTransform.vertexshader e SingleColor.fragmentshader
- No projeto onde você fez os tutoriais 1, 2 e 3, substitua o arquivo main.cpp pelo arquivo main.cpp que você baixou da pasta CG-Aula15;

Matrizes

- Descompacte o arquivo glm.zip;
- Insera a pasta glm no diretório include do projeto;
- Assim como foi feito na Aula 13, inclua os arquivos "shader.hpp" e “shader.cpp” no projeto;
- Compile e execute o projeto.

Matrizes



Atividade 03

- Altere glm::perspective
- Modifique a matriz do modelo *(Model)* para transladar, rotacionar e escalar o triângulo. Do jeito que está, no código, foi criada uma matriz identidade. Trabalhe com as matrizes de transformação em *Model.*
- Faça a mesma coisa, mas em ordens diferentes. O que você percebe? Qual é a melhor ordem que você deseja usar para um personagem?

Atividade 03

- Se necessário, crie funções que o auxiliem a resolver o problema (Por exemplo: *translate*)

Atividade 03

- Realizar a Atividade 03 até o dia 25/11, às 23h55min.

Referências

- OpenGL Tutorial

  http://www.opengl-tutorial.org/beginners-tutorials/

UFABC

Fim da Aula 15

André Luiz Brandão

UFABC