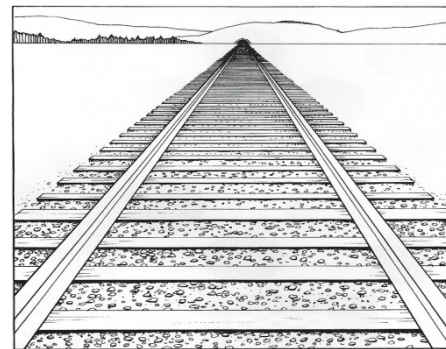


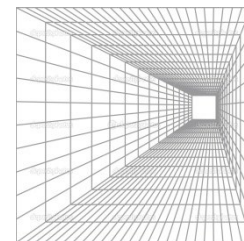
Câmera Sintética

por Rossana B Queiroz

Câmera Virtual

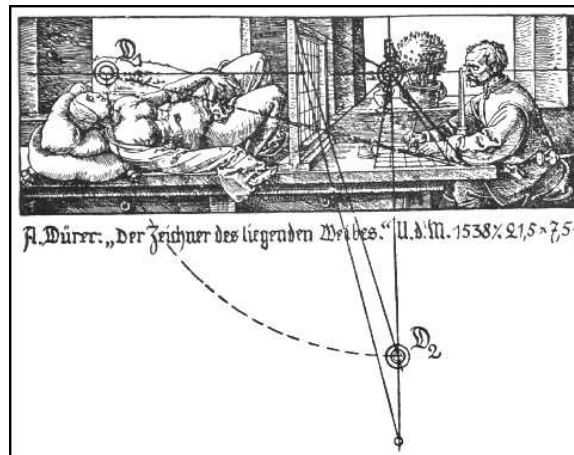


- Gostaríamos de poder navegar na nossa cena 3D
- Necessário configurar uma câmera virtual
 - Ajustar os pontos em relação à posição e orientação da câmera virtual (observador)
- Projetar a cena com perspectiva, ao invés da projeção ortográfica
 - Muito mais realista com ilusão de profundidade



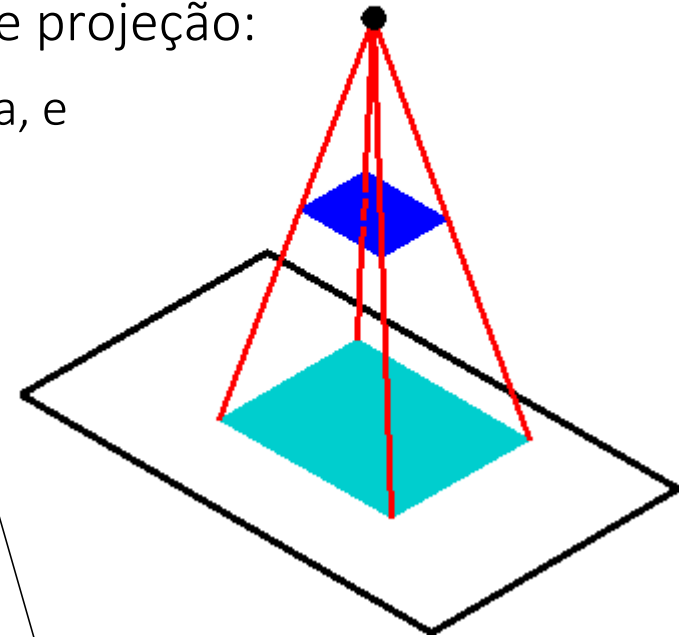
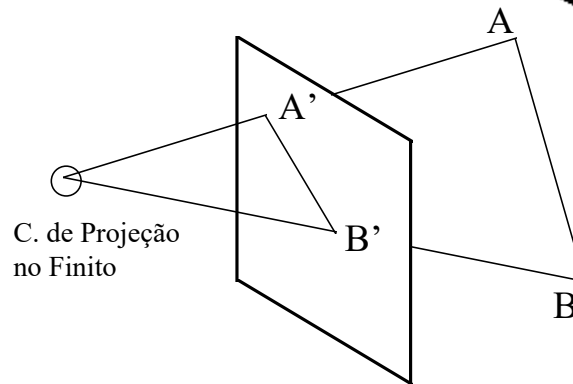
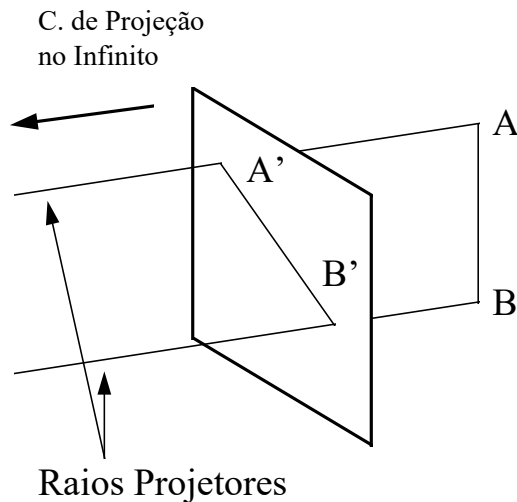
Perspectivas

- A palavra perspectiva vem do latim - *Perspicere* (ver através de)
- Se você se colocar atrás de uma janela envidraçada e, sem se mover do lugar, riscar no vidro o que está "vendo através da janela", terá feito uma perspectiva
- Perspectiva é a representação gráfica que mostra os objetos como eles aparecem a nossa vista, com três dimensões.



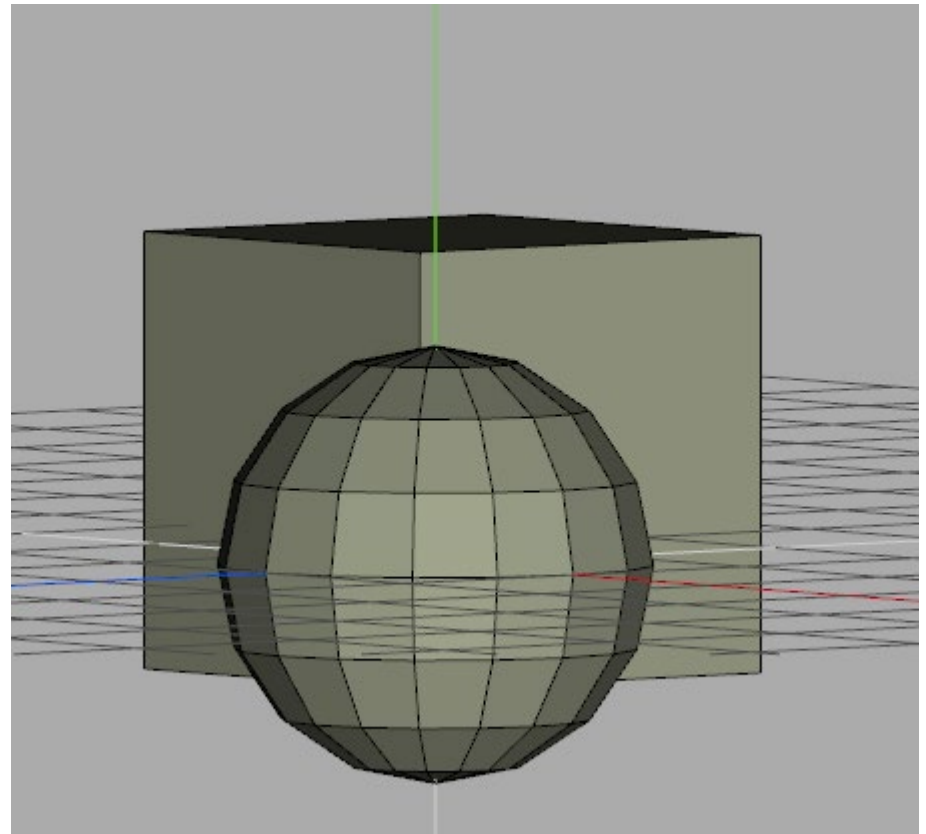
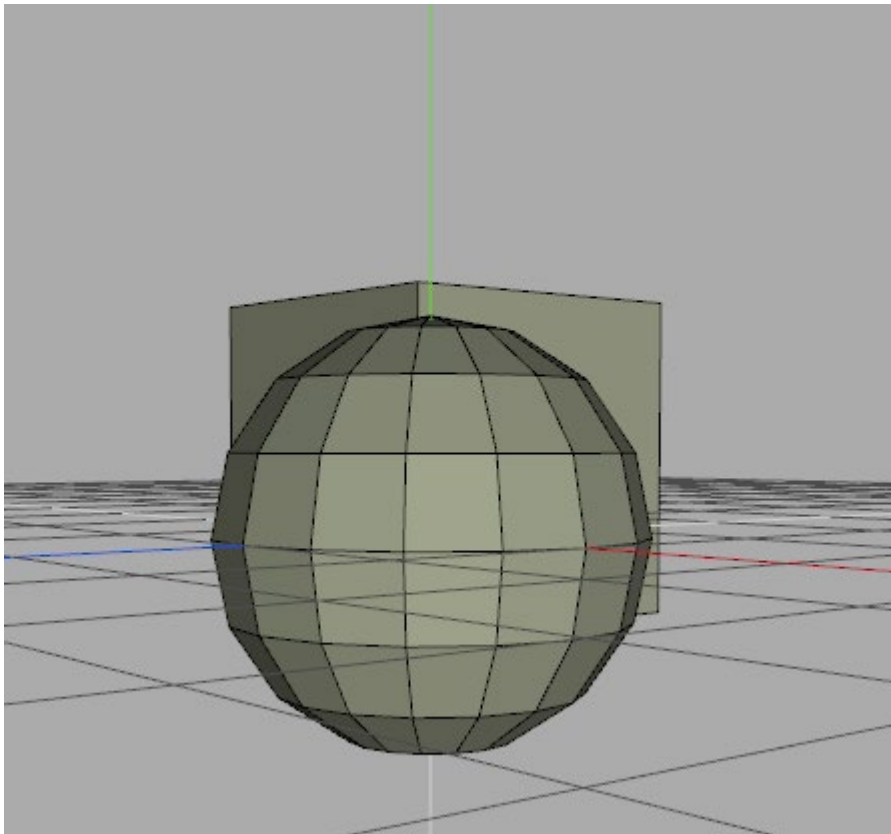
Projeções Paralelas e Perspectivas

- As projeções planares paralelas e perspectivas diferem com relação a distância do plano de projeção ao centro de projeção:
 - se a distância é finita, a projeção é perspectiva, e
 - se a distância é infinita, a projeção é paralela



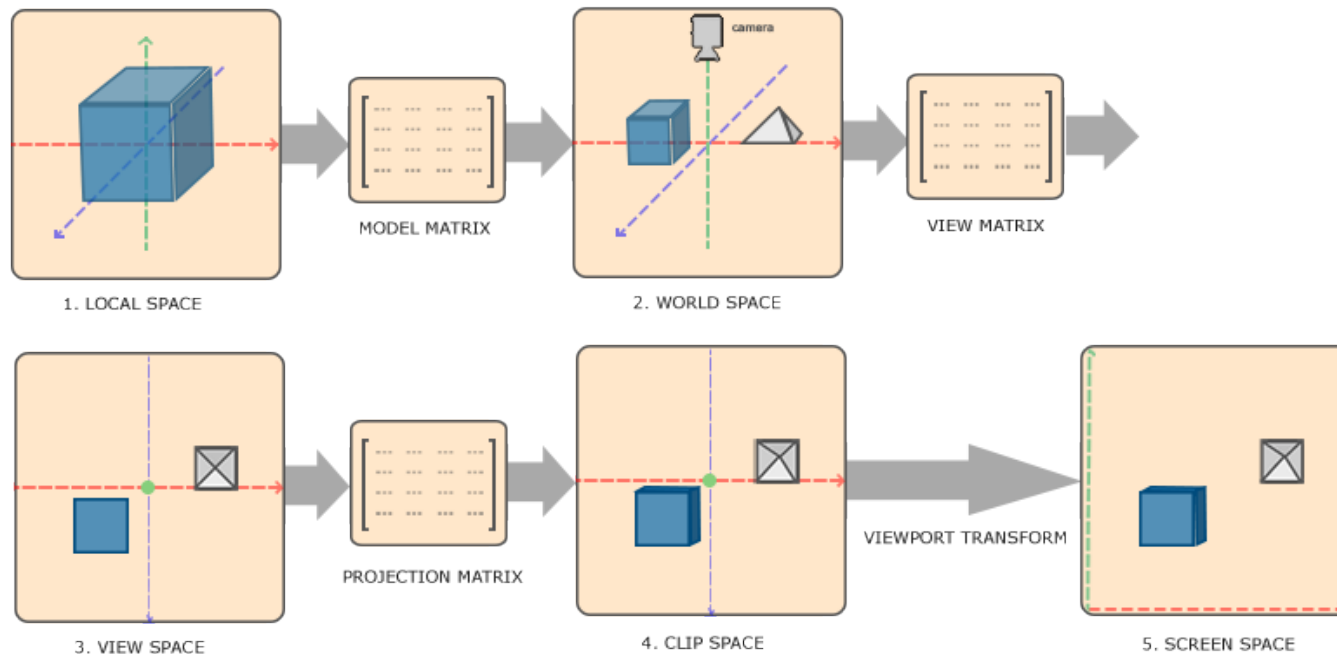
Projeções Paralelas e Perspectivas

Perspectiva (1 ponto de fuga) Paralela (ortográfica)



Pipeline 3D

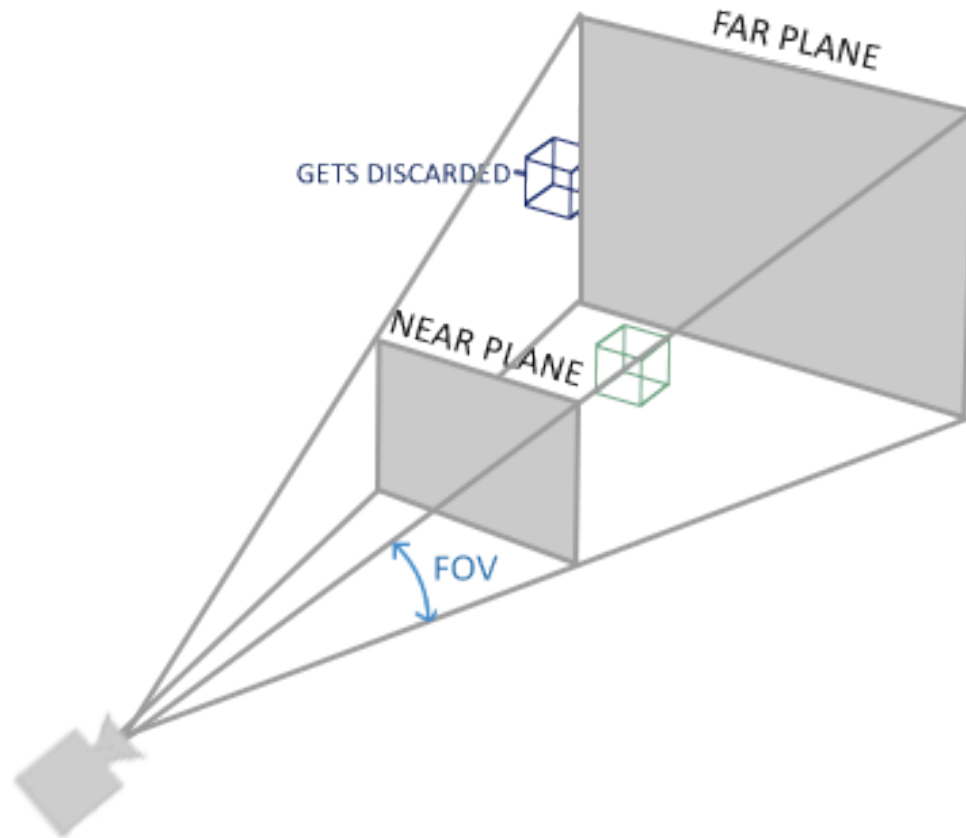
- A importância da câmera com projeção perspectiva: o que muda no pipeline?



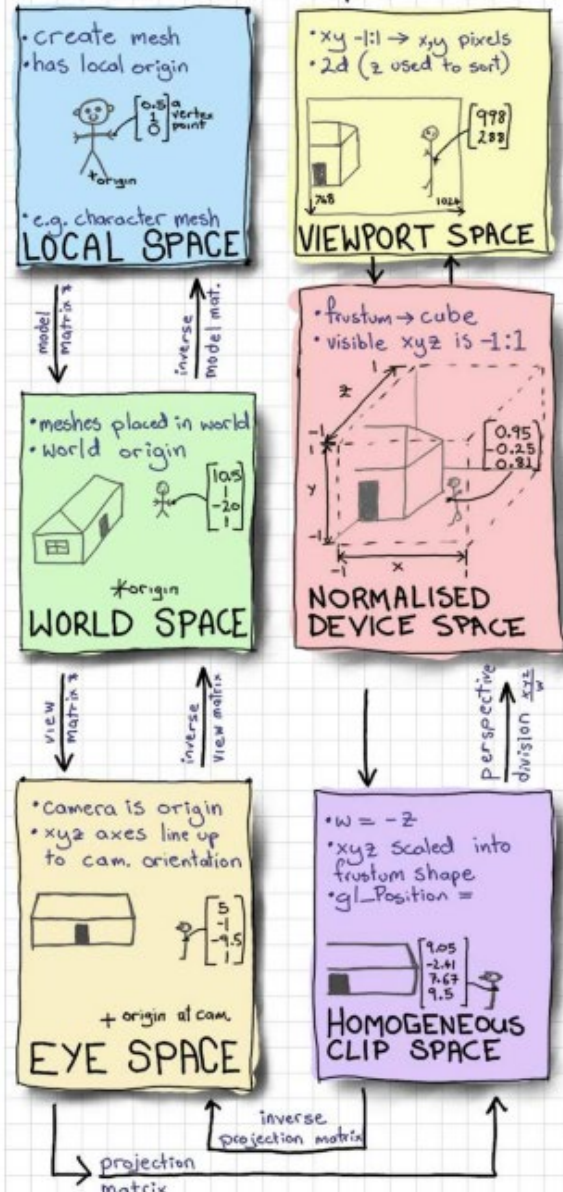
<https://learnopengl.com/#!Getting-started/Coordinate-Systems>

Câmera Sintética

- Frustum



3d Transformation Pipeline



[Anton's 2014]

Manipulado no **vertex shader** usando 3 matrizes:

1. **Model Matrix:** posiciona e orienta os vértices de um objeto na cena
2. **View Matrix:** posiciona e orienta os vértices em relação à câmera
3. **Perspective Matrix:** adiciona perspectiva; sensação de profundidade

World Space \rightarrow Eye Space

Esses passos são combinados na View Matrix:

$$VM = R * T$$

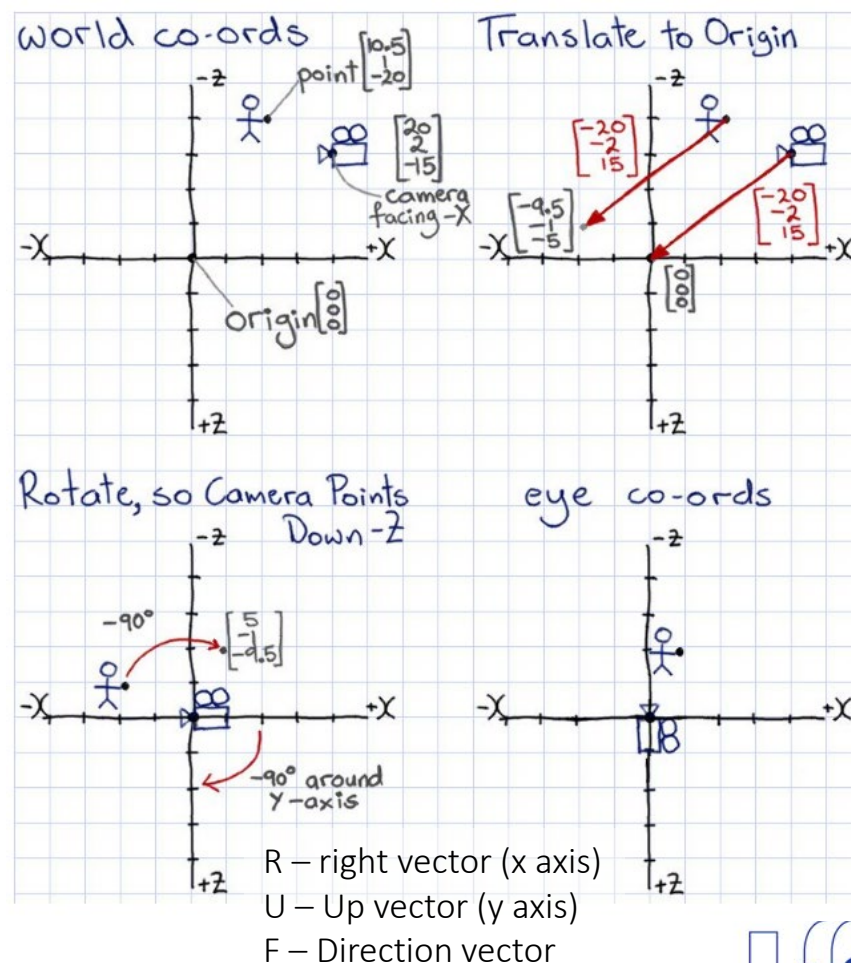
$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T = \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$V = R * T$

Bird's Eye View Matrix

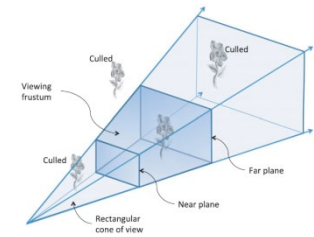
$$V = \begin{bmatrix} R_x & R_y & R_z & -P_x \\ U_x & U_y & U_z & -P_y \\ F_x & F_y & F_z & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Column-Major View Matrix



[Anton's 2014]

Matriz de Projeção



- A matriz de Projeção faz 3 coisas:
 1. Define os cortes **Near** e **Far**: ou seja, a faixa visível ao longo do eixo Z. Objetos fora dessas faixas são removidos do pipeline de renderização.
Ex: `near = 0.1; far = 100.0;`
 2. Define o **Field Of View(FOV)**: ângulo de visão conforme aspect ratio (width/height). Ex: aspect ratio 4:3 $\rightarrow 67^\circ * 4/3 = 89.33^\circ$
 3. Define o **Frustum**: volume de visualização, formado a partir do Near, Far e FOV

$$S_x = (2 * near) / (range * aspect + range * aspect)$$

$$S_y = near / range$$

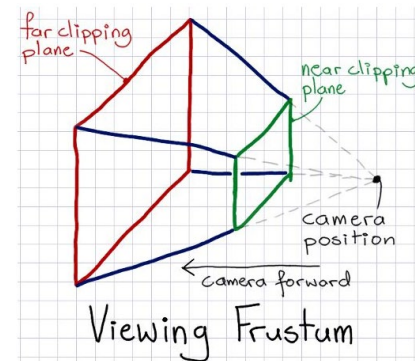
$$S_z = -(far + near) / (far - near)$$

$$P_z = -(2 * far * near) / (far - near)$$

$$range = \tan(fov * 0.5) * near$$

$$P = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & P_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

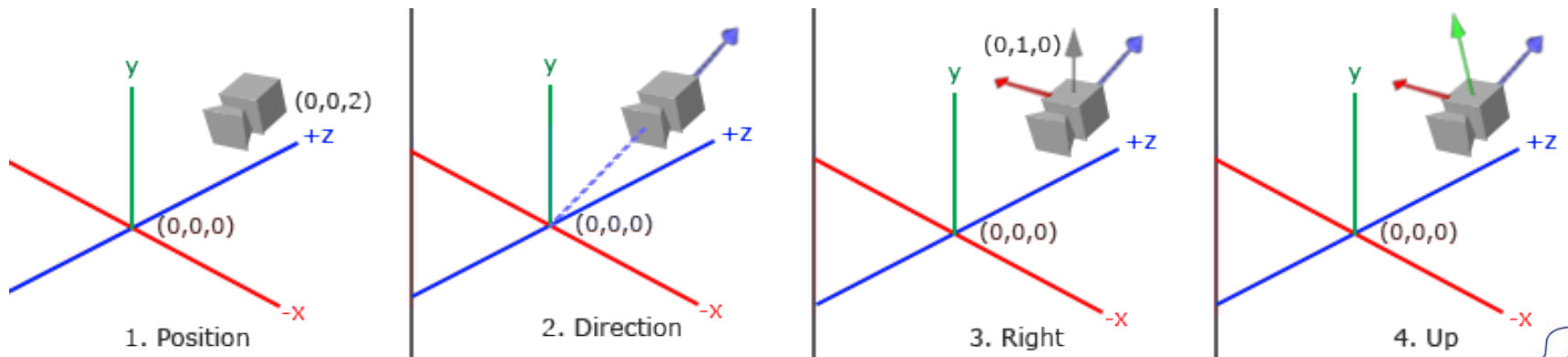
Projection Matrix
(column-major)



PASSO-A-PASSO: MOVIMENTAÇÃO

Parâmetros para *setar* a Câmera

- View matrix:
 - Posição, Orientação
 - Posição, Vetor Up e Right (ou Up e Front)



Parâmetros para *setar* a Câmera

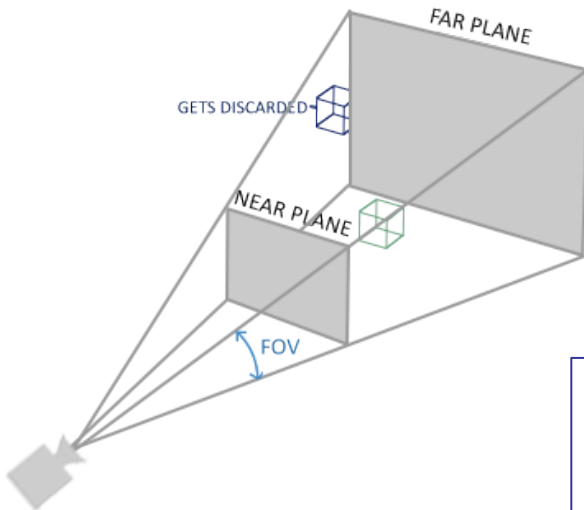
- View matrix:
 - Posição, Orientação
 - Posição, Vetor Up e Right (ou Up e Direction) → LookAt

$$LookAt = \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
glm::mat4 view;  
view = glm::lookAt(glm::vec3(0.0f, 0.0f, 3.0f), // Posição (ponto)  
glm::vec3(0.0f, 0.0f, 0.0f), // Target (ponto, não vetor) → dir = target - pos  
glm::vec3(0.0f, 1.0f, 0.0f)); // Up (vetor)
```

Parâmetros para *setar* a Câmera

- Projection matrix
 - Frustum
 - FOV (ângulo), Aspect ratio (width/height), zNear e zFar



$$P = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & P_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Projection Matrix
(column-major)

$$S_x = (2 * near) / (range * aspect + range * aspect)$$

$$S_y = near / range$$

$$S_z = -(far + near) / (far - near)$$

$$P_z = -(2 * far * near) / (far - near)$$


$$range = \tan(fov * 0.5) * near$$

```
glm::mat4 projection;  
projection = glm::perspective(45.0f, (GLfloat)WIDTH /  
(GLfloat)HEIGHT, 0.1f, 100.0f);
```


Parâmetros para *setar* a Câmera

- Como passar os dados para o shader
 - **uniform**: shader recebe como entrada uma variável vinda da CPU
 - Esta variável é global a todos os shaders

Mesmo
nome
que no
shader!!!

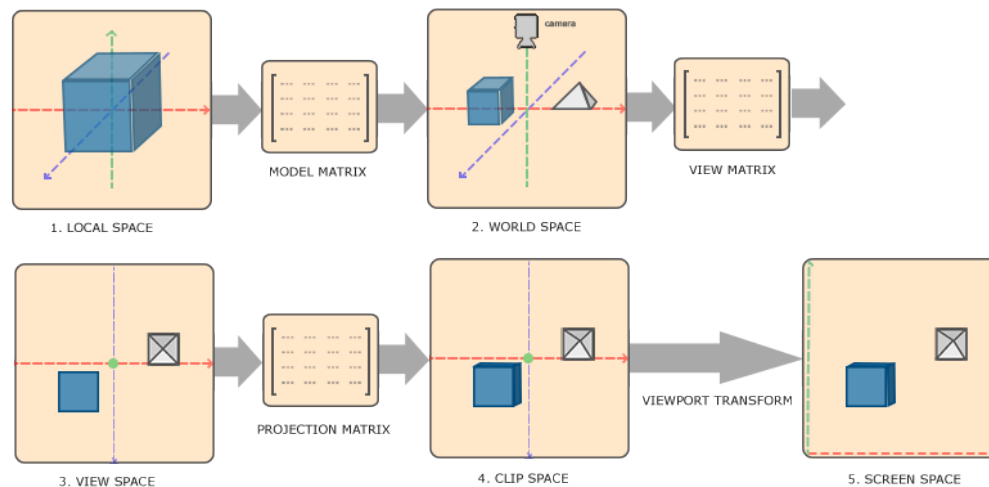


```
// Recupera sua localização
GLint modelLoc = glGetUniformLocation(shader.ID, "model");
GLint viewLoc = glGetUniformLocation(shader.ID, "view");
GLint projLoc = glGetUniformLocation(shader.ID, "projection");
// Passa seu conteúdo para o shader
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
// No caso da matriz de projeção, se não mudar não precisa passar a
// cada iteração
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
```

Parâmetros para *setar* a Câmera

- No vertex shader (GLSL):
 - Multiplicar obedecendo a seguinte ordem (column-based)

```
gl_Position = projection * view * model * vec4(position, 1.0f);
```



Movimentando nas 4 direções

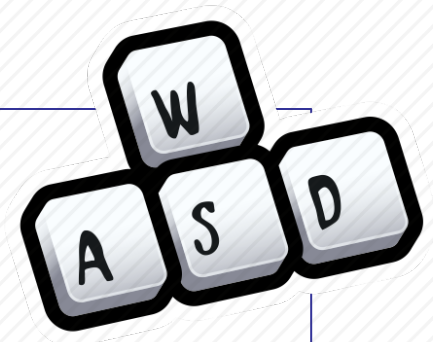
1. Armazenar a posição e orientação da camera

```
glm::vec3 cameraPos = glm::vec3(0.0f, 0.0f, 3.0f);  
glm::vec3 cameraFront = glm::vec3(0.0f, 0.0f, -1.0f);  
glm::vec3 cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);
```

Movimentando nas 4 direções

1. ~~Armazenar a posição e orientação da camera~~
2. Modificar nas *callbacks* de input (mouse, teclado, etc) ou proceduralmente

```
void updateCameraPos(GLFWwindow *window)
{ ...
    float cameraSpeed = 0.05f; // adjust accordingly
    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
        cameraPos += cameraSpeed * cameraFront;
    if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
        cameraPos -= cameraSpeed * cameraFront;
    if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
        cameraPos -= glm::normalize(glm::cross(cameraFront, cameraUp)) * cameraSpeed;
    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
        cameraPos += glm::normalize(glm::cross(cameraFront, cameraUp)) * cameraSpeed;
}
```



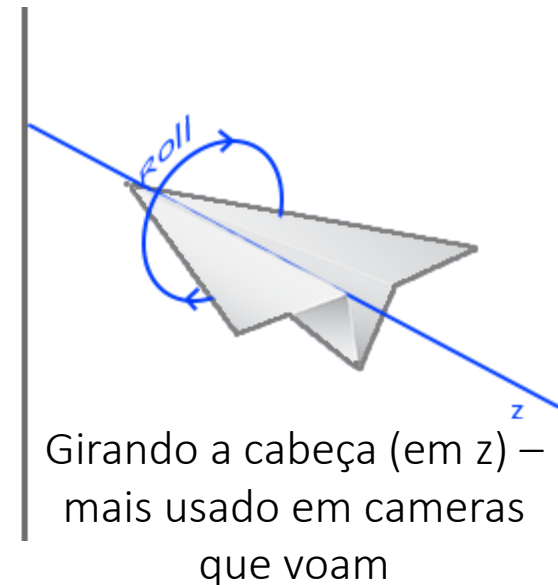
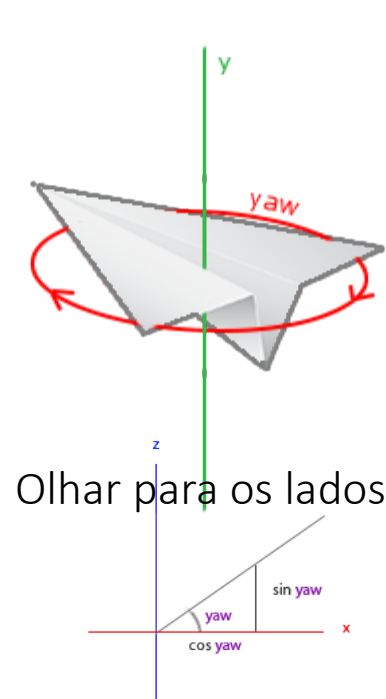
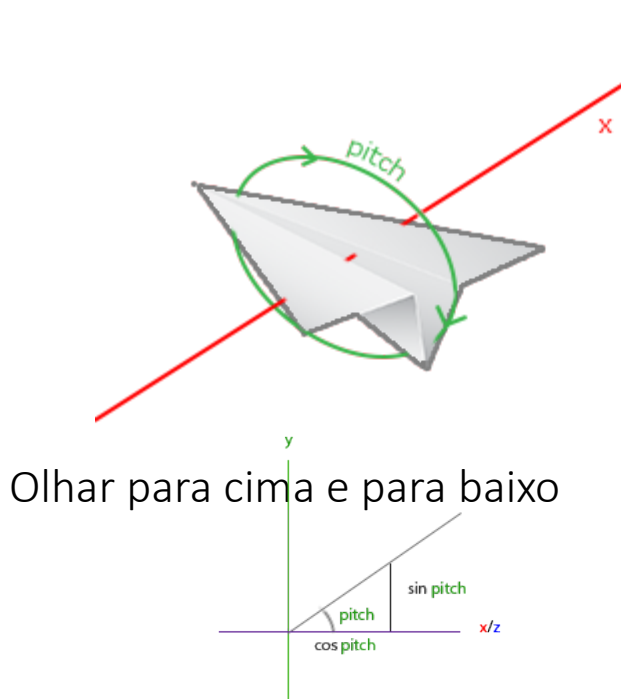
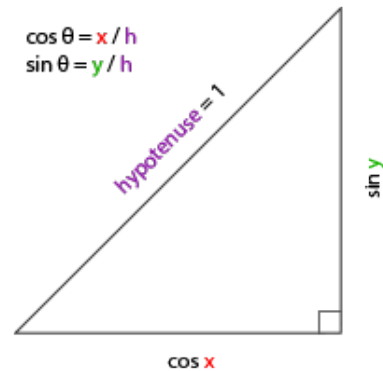
Controlando a velocidade

1. Verifica-se quanto tempo se passou de um ciclo para outro
 - deltaTime
2. Usá-lo para moderar a velocidade

```
float currentFrame = glfwGetTime();  
deltaTime = currentFrame - lastFrame;  
lastFrame = currentFrame;  
  
void processInput(GLFWwindow *window) {  
    float cameraSpeed = 2.5f * deltaTime;  
    ...  
}
```

Mudando a direção

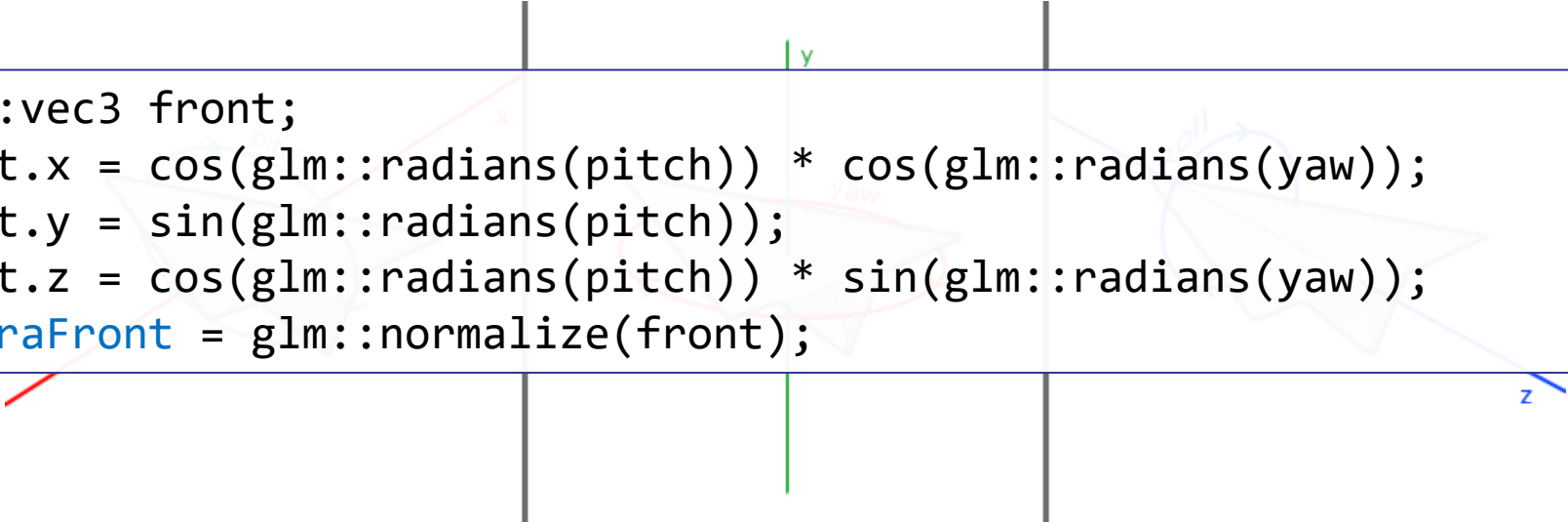
- Olhando ao redor
 - Conceito de Ângulos de Euler



Mudando a direção

- Alterando o vetor direção (*front*)

```
glm::vec3 front;  
front.x = cos(glm::radians(pitch)) * cos(glm::radians(yaw));  
front.y = sin(glm::radians(pitch));  
front.z = cos(glm::radians(pitch)) * sin(glm::radians(yaw));  
cameraFront = glm::normalize(front);
```



Mudando a direção

- Variáveis para controle da câmera
 - Dica: seriam atributos de uma classe Camera

```
//Variáveis globais para o controle da câmera  
glm::vec3 cameraPos, cameraFront, cameraUp;  
bool firstMouse = true;  
float lastX = WIDTH / 2.0, lastY = HEIGHT / 2.0;  
//para calcular o quanto que o mouse deslocou  
float yaw = -90.0, pitch = 0.0; //rotação em x e y
```

Mudando a direção

- Usando o mouse para o input da direção
 - 1 Calcular o deslocamento do mouse de um frame para o outro
 - 2 Adicionar esses valores no yaw e no pitch
 - 3 Adicionar algumas restrições no yaw (se quiser, no pitch também)
 - 4 Recalcular o vetor direção e up

Mudando a direção

- Desabilitar o cursor para inicializar no centro da tela

```
glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
```

- Usar a função de callback de movimento do mouse

Mudando a direção

```
void mouse_callback(GLFWwindow* window, double xpos, double ypos)
{
    if(firstMouse)
    {
        lastX = xpos;
        lastY = ypos;
        firstMouse = false;
    }

    float xoffset = xpos - lastX;
    float yoffset = lastY - ypos;
    lastX = xpos;
    lastY = ypos;

    float sensitivity = 0.05;
    xoffset *= sensitivity;
    yoffset *= sensitivity;
```

1

Amortizando, deixando o movimento mais suave

Mudando a direção

```
yaw += xoffset;  
pitch += yoffset;
```

2

```
if(pitch > 89.0f)  
    pitch = 89.0f;  
if(pitch < -89.0f)  
    pitch = -89.0f;
```

3

```
glm::vec3 front;  
front.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));  
front.y = sin(glm::radians(pitch));  
front.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));  
cameraFront = glm::normalize(front);
```

4

```
//Precisamos também atualizar o cameraUp!! Pra isso, usamos o Up do  
//mundo (y), recalculamos Right e depois o Up  
glm::vec3 right = glm::normalize(glm::cross(cameraFront,  
glm::vec3(0.0,1.0,0.0)));  
cameraUp = glm::normalize(glm::cross(right, cameraFront));
```

```
}
```

Mudando a direção

- Dentro do loop da aplicação, no momento de atualizar a matriz de *view*, precisamos usar nossas variáveis de controle!

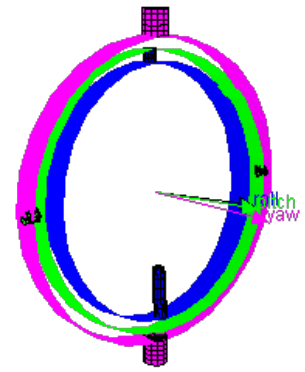
```
view = glm::lookAt(cameraPos, cameraPos+cameraFront, cameraUp);
```

Zoom

- Usar a função de *callback* de *scroll* do mouse para dar zoom in/out → alterando o FOV

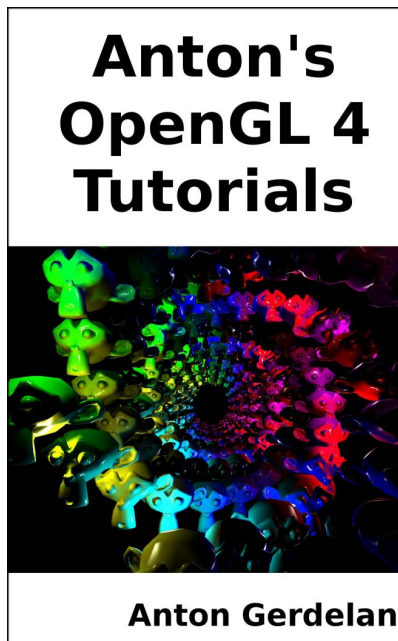
```
void scroll_callback(GLFWwindow* window,
double xoffset, double yoffset)
{
    if(fov >= 1.0f && fov <= 45.0f)
        fov -= yoffset;
    if(fov <= 1.0f)
        fov = 1.0f;
    if(fov >= 45.0f)
        fov = 45.0f;
}
```


Considerações



- Este é um sistema de câmera simples, e portanto imperfeito
 - Pode ocasionar o efeito de “Gimbal lock” (2 eixos ficarem em paralelo), perdendo um grau de Liberdade
 - Solução: adicionar um quarto eixo: uso de quaternions
 - Capítulo “Quaternion Quick Start”, do Anton’s OpenGL 4 Tutorials
 - <http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-17-quaternions/>

Referências



Ebook para Kindle

Muitos materiais online disponíveis em:

<http://antongerdelan.net/opengl/>

Referências

- Slides sobre CG dos professores: Christian Hofsetz, Cristiano Franco, Marcelo Walter, Soraia Musse, Leandro Tonietto e Rafael Hocevar.
- Leituras obrigatórias:
 - <https://learnopengl.com/Getting-started/Camera>