

DCC192

2025/1



Desenvolvimento de Jogos Digitais

A27: Gráficos 3D — Transformações

Prof. Lucas N. Ferreira

Plano de Aula

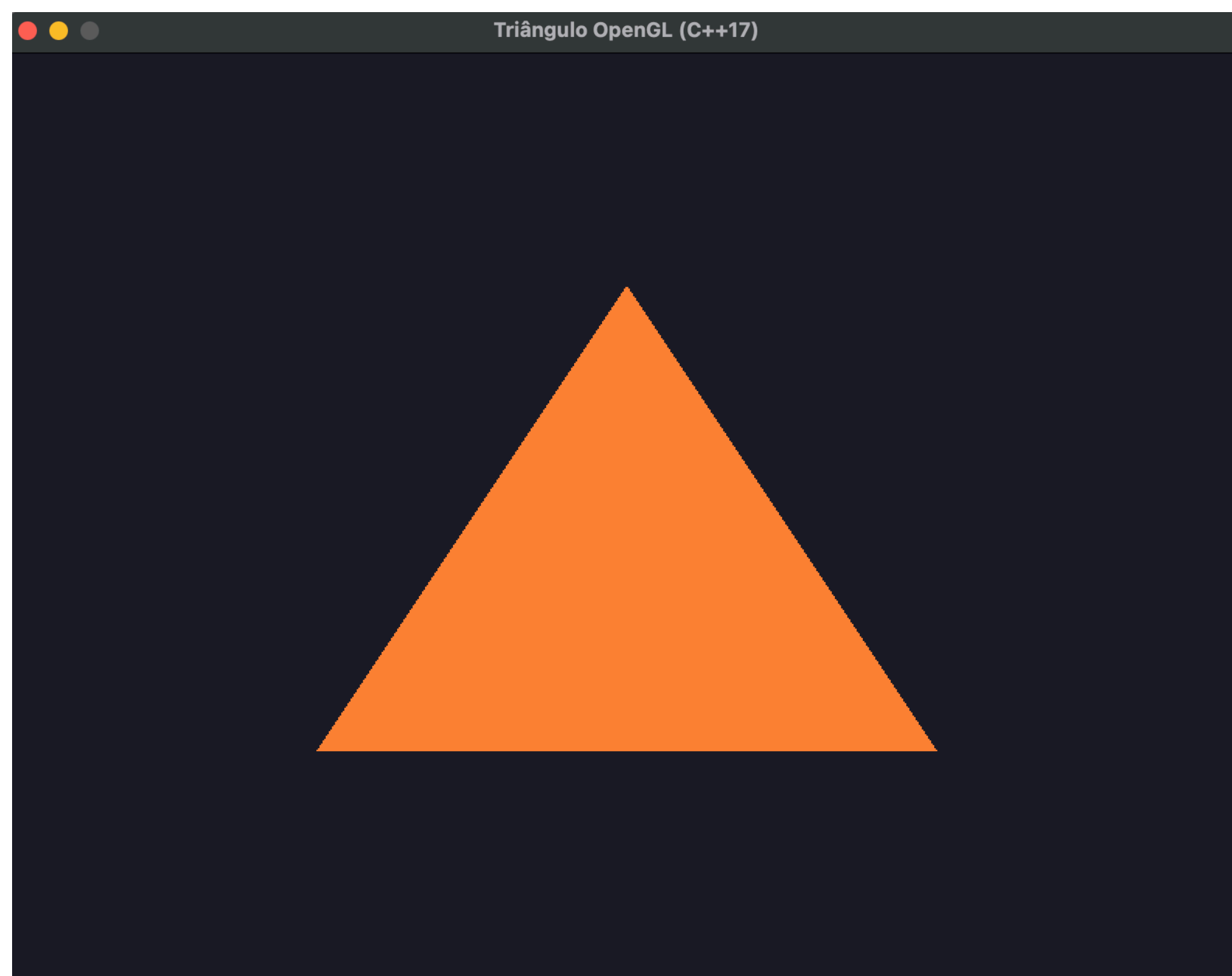


- ▶ Mais detalhes de OpenGL e GLSL
- ▶ Especificando Múltiplos Objetos
- ▶ Transformações
 - ▶ Translação, Rotação e Escala
 - ▶ Coordenadas Homogêneas
- ▶ Matrizes de Transformação em OpenGL/GLSL

Primeiro programa em OpenGL/GLSL



Na última aula, vimos rapidamente um programa em OpenGL/GLSL com SDL para desenhar um triângulo na tela. Vamos entender quais são as partes que compõe esse programa:



1. Inicialização da SDL com OpenGL
2. Definir os vértices dos modelos e configurar buffers:
 - ▶ Vertex Array Object (VAO)
 - ▶ Vertex Buffer Object (VBO)
 - ▶ Index Buffer Objects (IBO)
3. Escrever, compilar e linkar os vertex e fragment shaders
4. Limpar a tela e desenhar vértices

Definir os vértices dos modelos e configurar buffers



- ▶ Vertex Array Object (VAO): armazena o estado de configuração dos atributos de vértice
- ▶ Vertex Buffer Object (VBO): armazena os dados brutos dos vértices
- ▶ Index Buffer Objects (IBO): armazena índices que definem quais vértices desenhar e em que ordem

```
// Define os vértices de um triângulo em coordenadas normalizadas
float vertices[] = {-0.5f, -0.5f, 0.5f, -0.5f, 0.0f, 0.5f };

// Cria e configura VAO (Vertex Array Object) e VBO (Vertex Buffer Object)
GLuint VAO, VBO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);

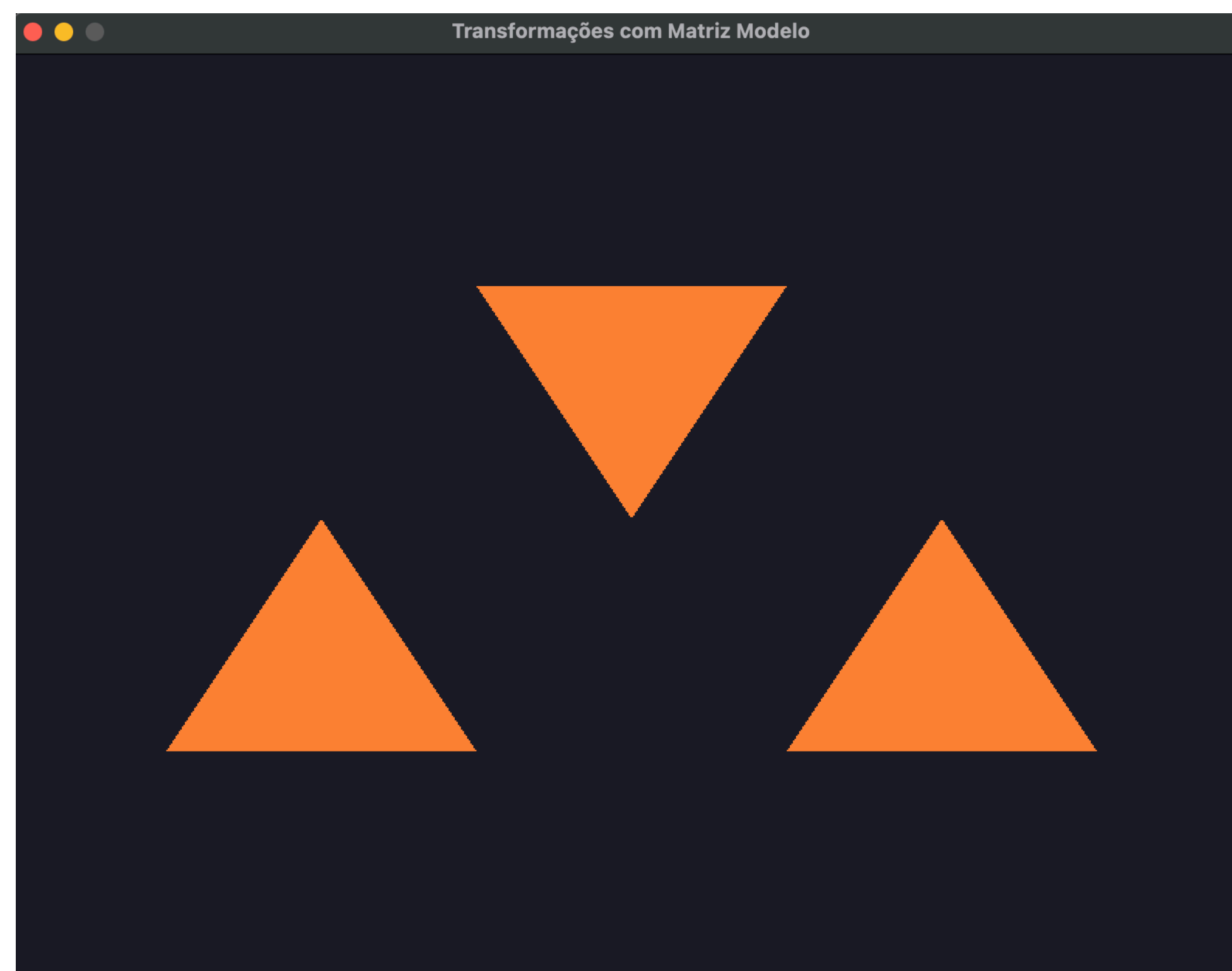
glBindVertexArray(VAO); // Ativa VAO
glBindBuffer(GL_ARRAY_BUFFER, VBO); // Liga VBO
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW); // Envia dados

// Informa como interpretar os dados do VBO (layout)
glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0); // Ativa atributo de vértice 0
```

Transformações Geométricas



Em jogos, é muito comum termos que criar múltiplas instâncias de um mesmo objeto. Para fazer isso, podemos utilizar a mesma lista de vértices e aplicar transformações geométricas:

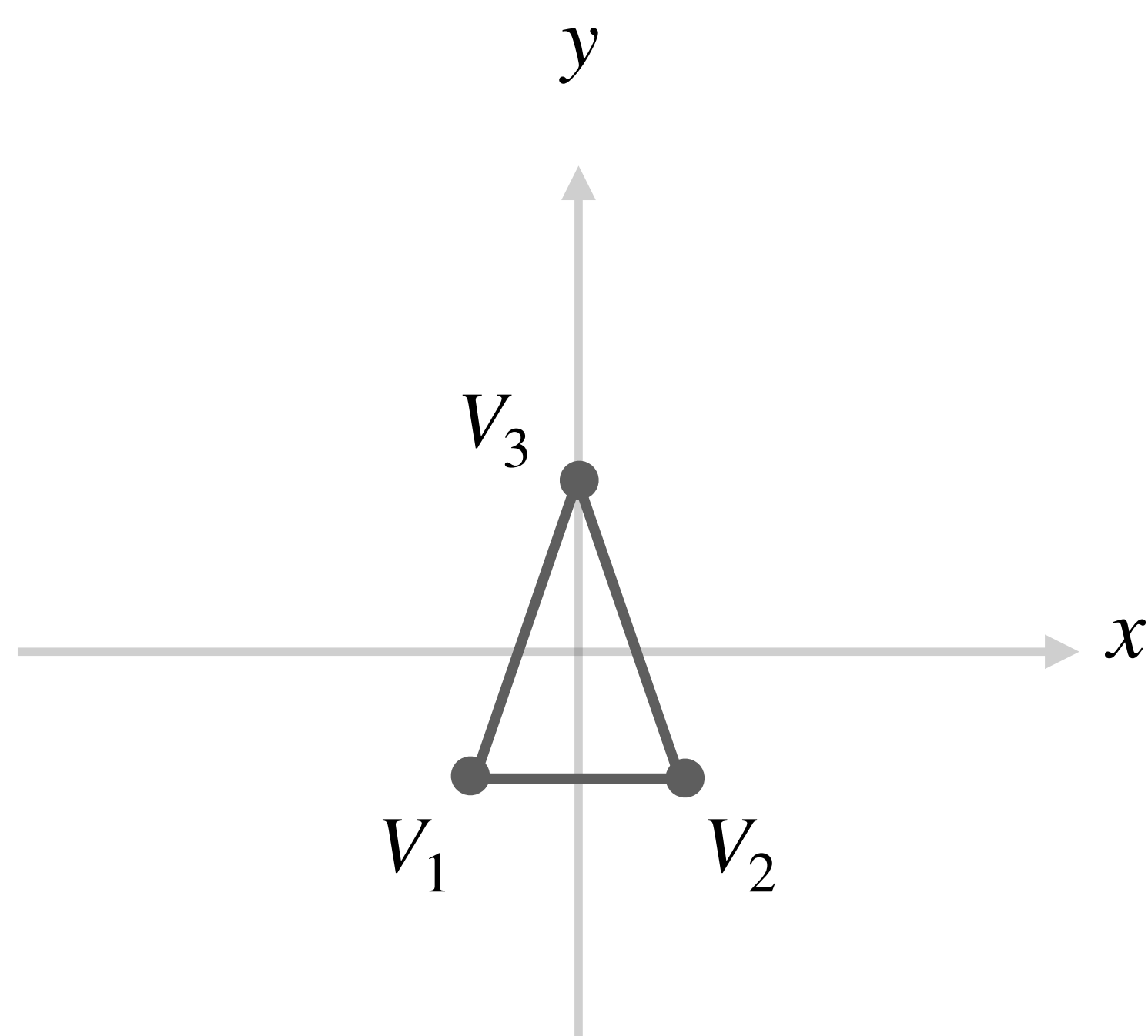


- ▶ Em computação gráfica, existem três tipos de transformações geométricas mais importantes:
 - ▶ Translação
 - ▶ Rotação
 - ▶ Escala
- ▶ Cada transformação é representada por uma matriz, que são combinadas em uma única matriz final chamada de *model matrix*
- ▶ A *model matrix* é enviada a ao vertex shader, que realiza a multiplicação em todos os vértices em paralelo.

Translação

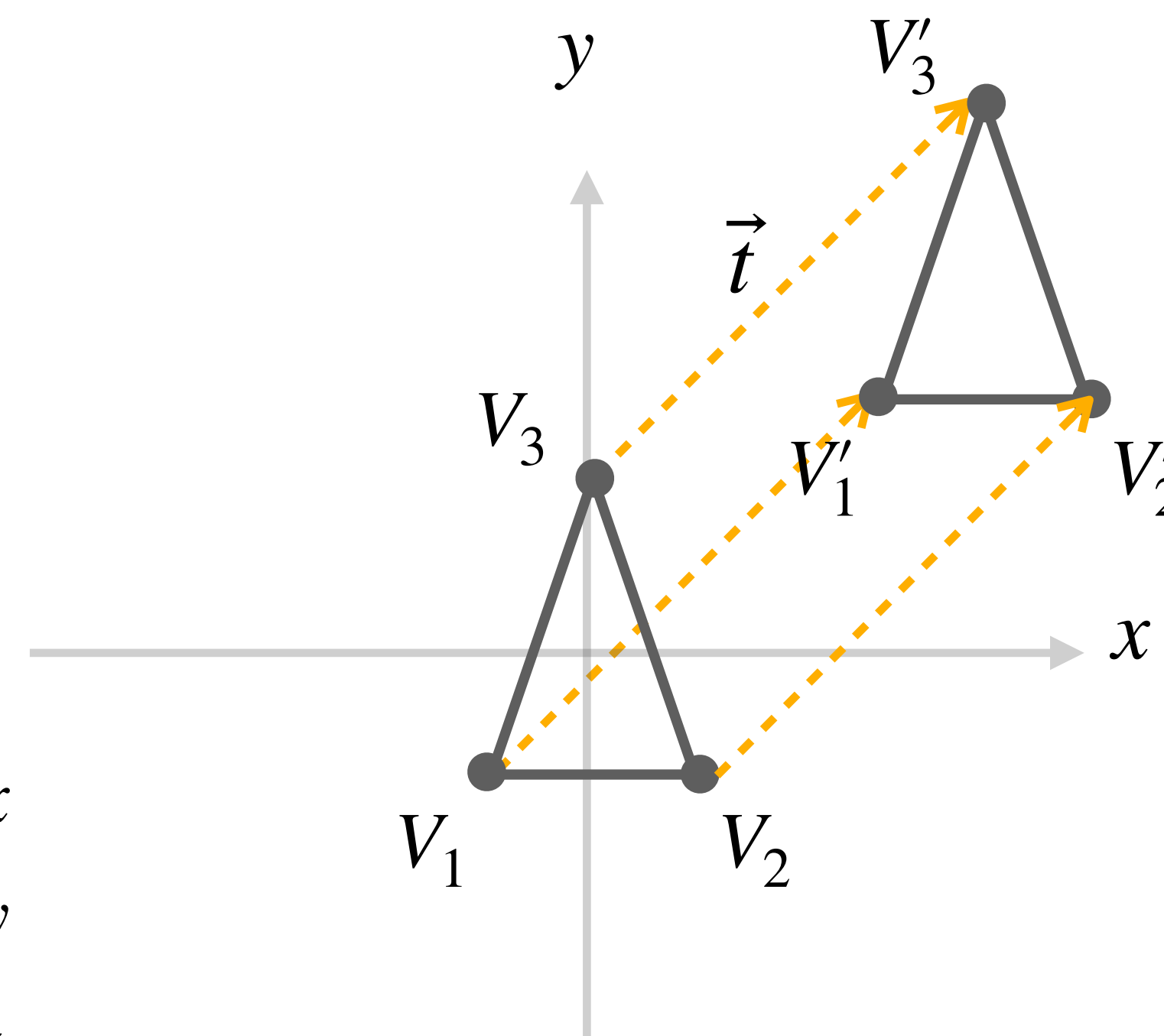


Para realizar uma transformação de translação, basta somar um dado vetor de deslocamento \vec{t} a todos os vértices V_i do modelo:



$$V' = V + \vec{t}$$
$$V' = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

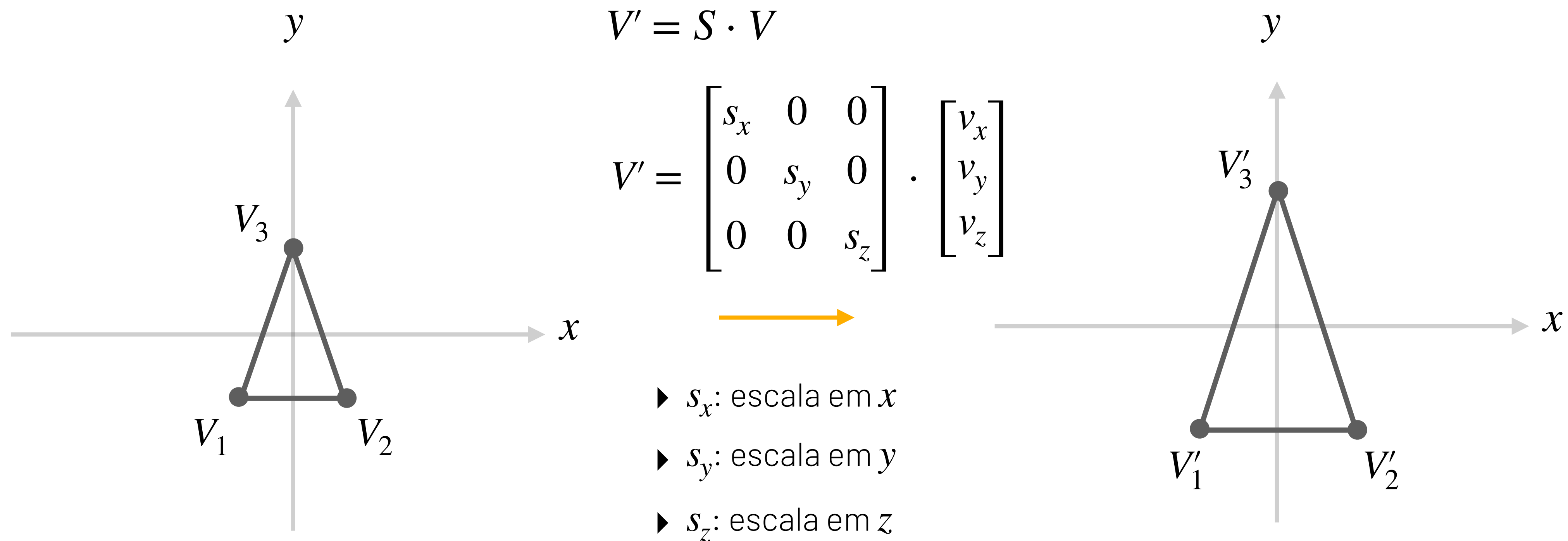
- ▶ t_x : deslocamento em x
- ▶ t_y : deslocamento em y
- ▶ t_z : deslocamento em z



Escala



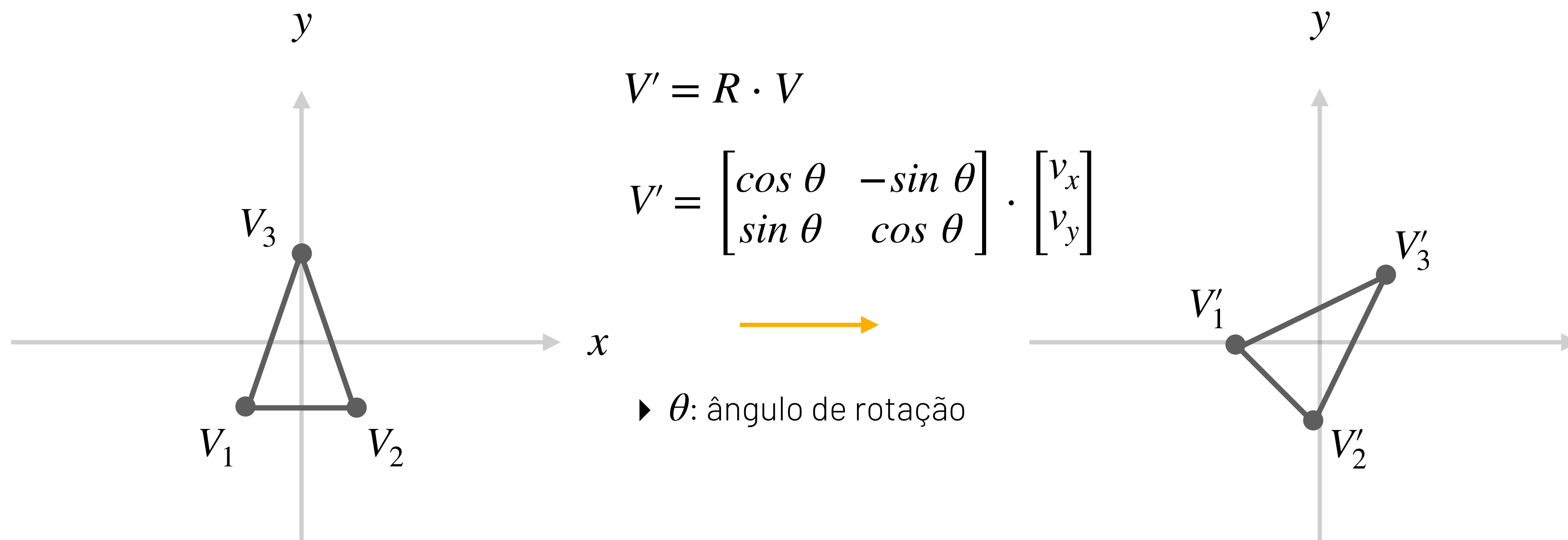
Para realizar uma transformação de escala, basta multiplicar uma matriz diagonal S por todos os vértices do modelo. As diagonais representam os fatores de escala:



Rotação em 2D



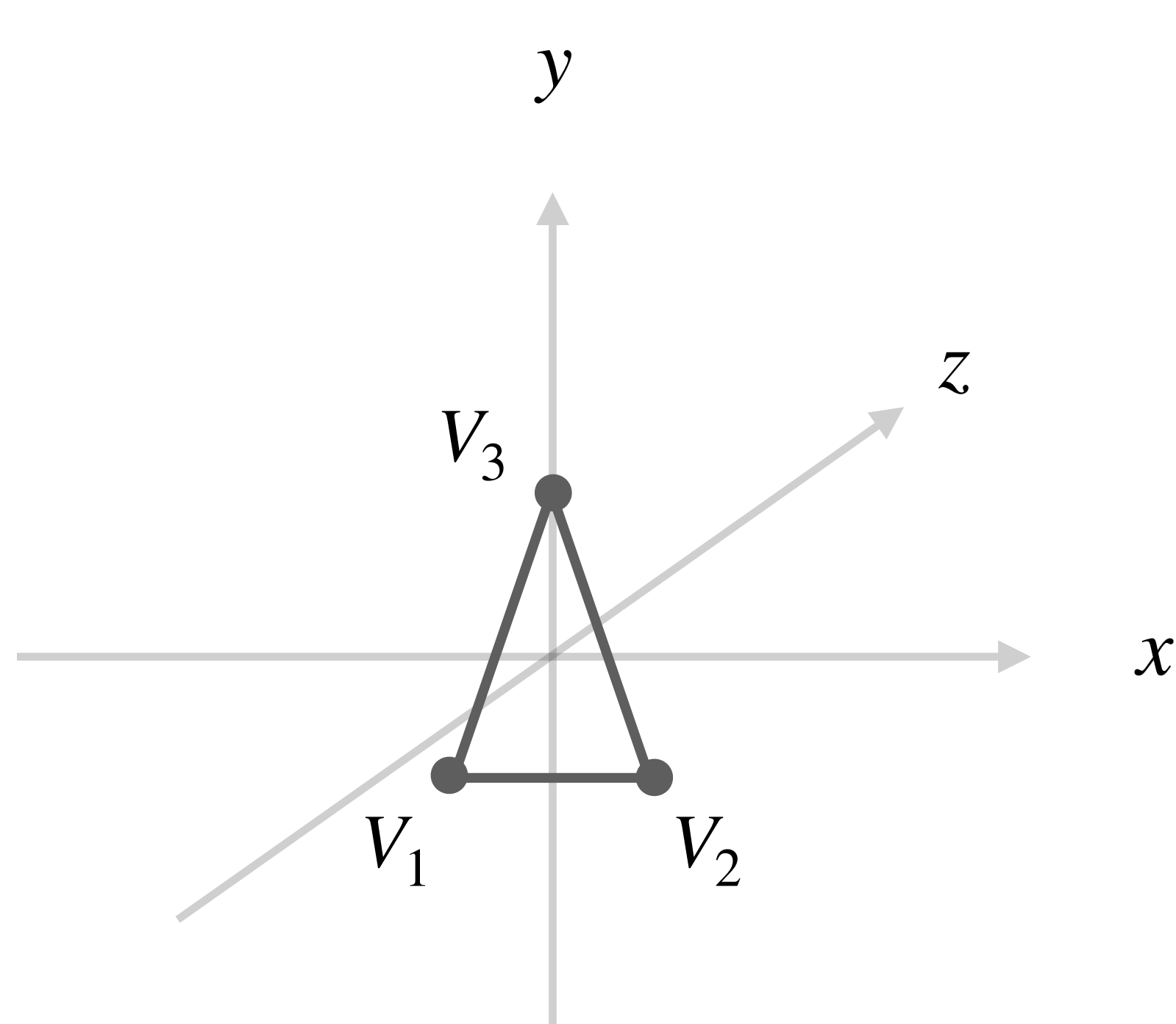
Como vimos em aulas anteriores, para realizar uma transformação de rotação em 2D, basta multiplicar uma matriz de rotação R por todos os vértices do modelo:



Rotação em 3D



Em um espaço tridimensional, temos 3 diferentes eixos para fazer a rotação: x , y e z . As matrizes de rotação são diferentes para cada eixo:



$$\xrightarrow{x} V' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

$$\xrightarrow{y} V' = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

$$\xrightarrow{z} V' = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

Coordenadas Homônêneas



É mais eficiente (evita tráfego CPU x GPU) combinar as transformações geométricas em uma única matriz de transformações. Para isso, podemos fazer as seguintes manipulações:

1. Adicionar uma dimensão $w = 1$ aos vértices:

$$\begin{array}{c} V_i \\ \begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \\ (3D) \end{array}$$

2. Representar as transformações com uma matriz 4×4 :

$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

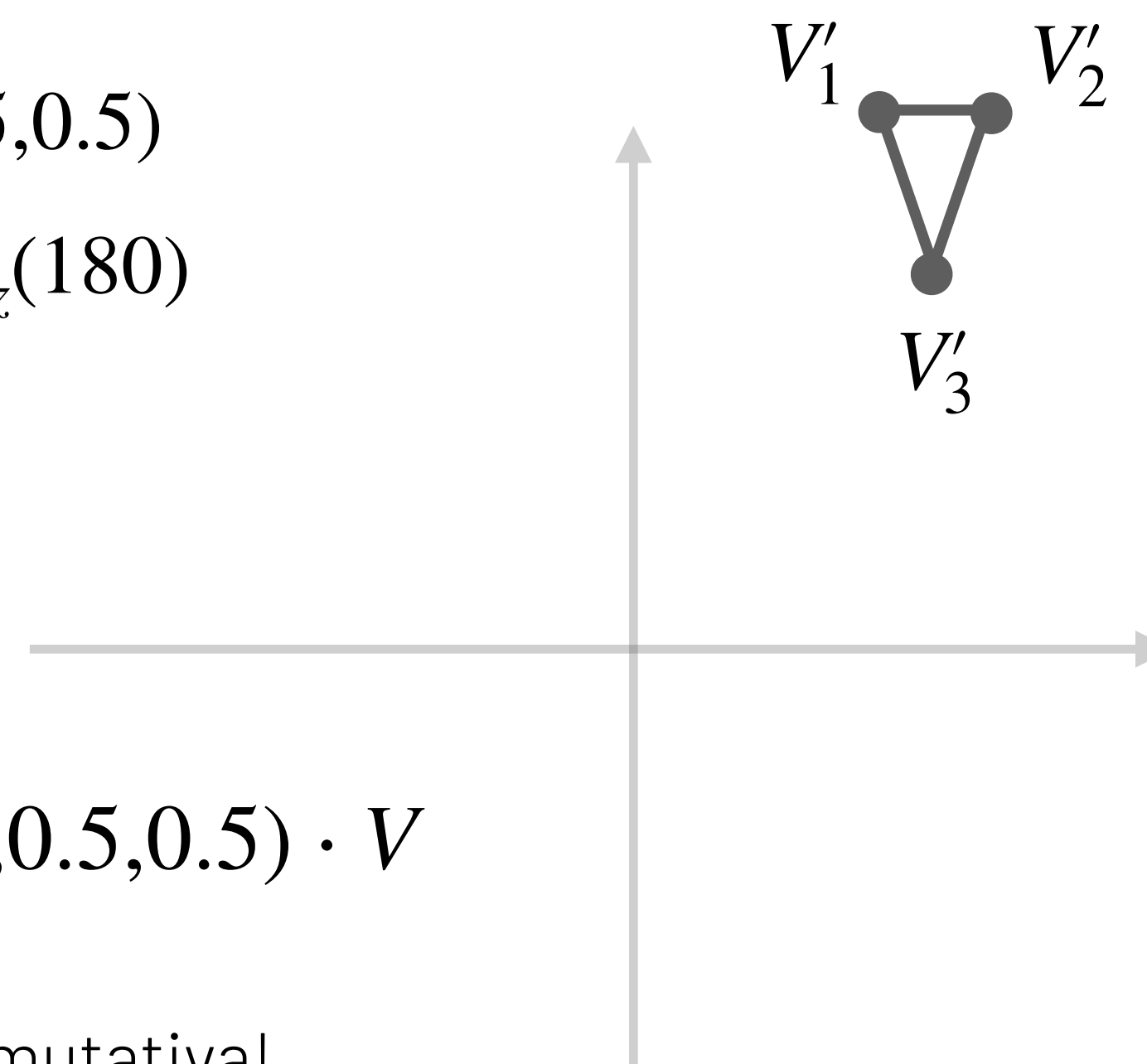
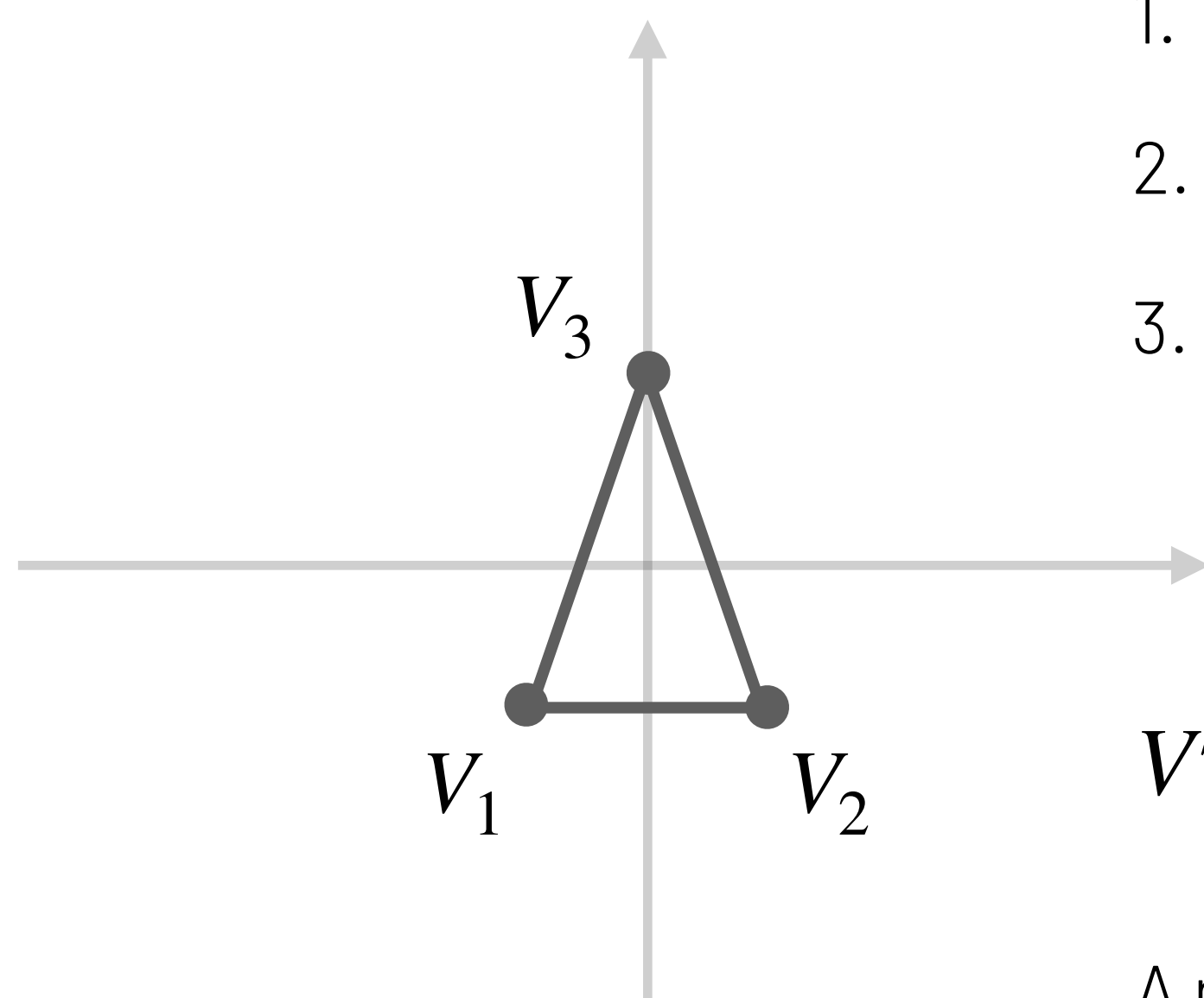
$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Coordenadas Homogêneas



Utilizando coordenadas homogêneas, podemos realizar composições de transformações multiplicando as matrizes de transformação:

1. Escala uniforme de 0.5: $S(0.5,0.5,0.5)$
2. Rotação de 180 graus no eixo z: $R_z(180)$
3. Translação $T(1,2,0)$



$$V' = T(1,2,0) \cdot R_z(180) \cdot S(0.5,0.5,0.5) \cdot V$$

A multiplicação de matrizes não é comutativa!

- ▶ Mesmas transformações em ordens diferentes podem gerar resultados diferentes!
- ▶ Fazemos a escala e a rotação primeiro, pois elas são afetadas pela translação

Criando Matrizes de Transformação em OpenGL



Como os modelos podem possuir muitos vértices, é mais eficiente aplicar as transformações na GPU. Para isso, criamos e combinamos as matrizes na CPU e enviá-las ao Vertex Shader:

CPU

1. Buscar o endereço da matriz(uniform) no shader
2. Criar uma matriz 4×4 de transformação na CPU
3. Enviar essa matriz para a GPU

```
GLint modelLoc = glGetUniformLocation(shaderProgram, "model");

// Terceiro triângulo no centro e acima, com rotação e escala
Matrix4 model3 = Matrix4::CreateTranslation(Vector3(0.0f, 0.25f, 0.0f));
model3 = Matrix4::CreateScale(Vector3(0.5f, 0.5f, 1.0f)) * model3;
model3 = Matrix4::CreateRotationZ(Math::ToRadians(180.0f)) * model3;

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, model3.GetAsFloatPtr());
glDrawArrays(GL_TRIANGLES, 0, 3);
```

GPU

1. Criar uma matriz 4×4 (uniform) no vertex shader
2. Multiplicar essa matriz pelos vértices de entrada

```
constexpr std::string_view vertexShaderSource = R"(
#version 330 core
layout (location = 0) in vec2 aPos;
uniform mat4 model;

void main() {
    gl_Position = model * vec4(aPos, 0.0, 1.0);
}
)";
```

Próxima aula



A28: Gráficos 3D III

- ▶ Câmeras
 - ▶ Perspectiva e Ortográfica
- ▶ Projeção para a Tela
- ▶ Texturas