UFABC

Computação Gráfica

André Brandão

Aula 13

Mais sobre OpenGL

Aula 13

Slides de autoria do
Professor André Balan

UFABC

Conteúdo

- Transformação "**Window to Viewport**"
- Biblioteca de transformações 2D
- Transformações 2D em OpenGL

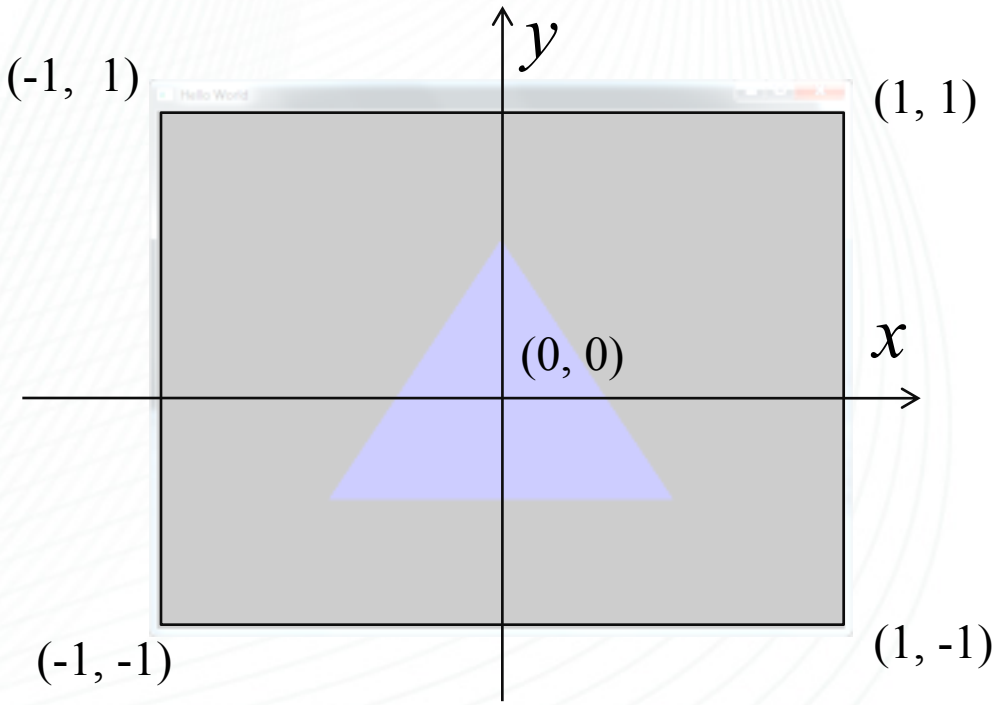
Window to Viewport

- Depois de executar o vertexShader para um vértice, o OpenGL precisa mapear a posição final desse vértice para as coordenadas da tela.

- Essa é uma transformação conhecida como "*window to viewport*"

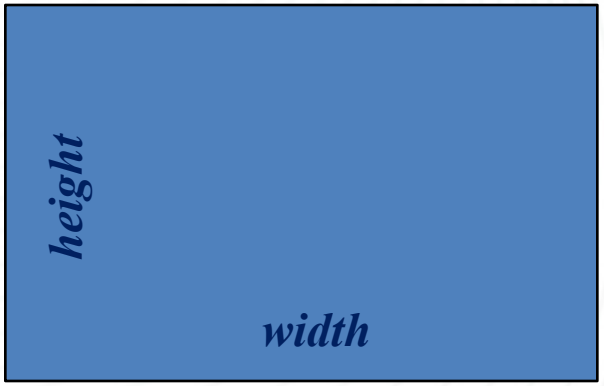
Window to Viewport

- **Objetivo**: transformar as coordenadas dos pixels do sistema normalizado [-1, 1] para o sistema de coordenadas da janela OpenGL (***viewport***)



Window to Viewport

- Usamos a função **glViewport** para definirmos a janela de visualização OpenGL, geralmente, **ocupando toda a janela**, de largura *width* e altura *height*:



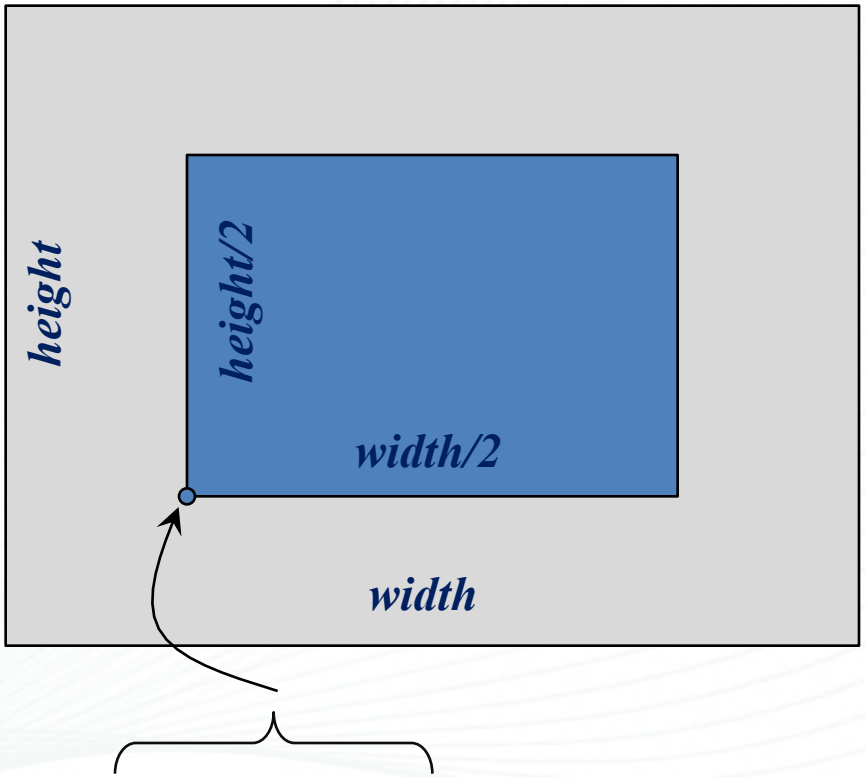

```
glViewport(0, 0, width, height);
```

Canto **inferior esquerdo** da viewport

Largura e **altura** da viewport

Window to Viewport

- Se quisermos, por exemplo desenhar somente no centro da janela..




```
glViewport(width/4, width/4, width/2, height/2);
```

Window to Viewport

- Quando definir a *viewport* no OpenGL?
  - Uma única vez quando se cria a janela OpenGL (a glfw define automaticamente a viewport na criação da janela)
  - Quando a janela é redimensionada (opcional)

Window to Viewport

- Função **call-back** da GLFW para redimensionamento da janela OpenGL

```
void on_resize(GLFWwindow* window, int width, int height) {
    glViewport(0, 0, width, height);
}
```

- Definindo a função call-back para a GLFW

```
.
.
.
glfwSetFramebufferSizeCallback(window, on_resize);
.
.
.
```

Window to Viewport

- A transformação "*window to viewport*" é uma composição de translação e escalas!

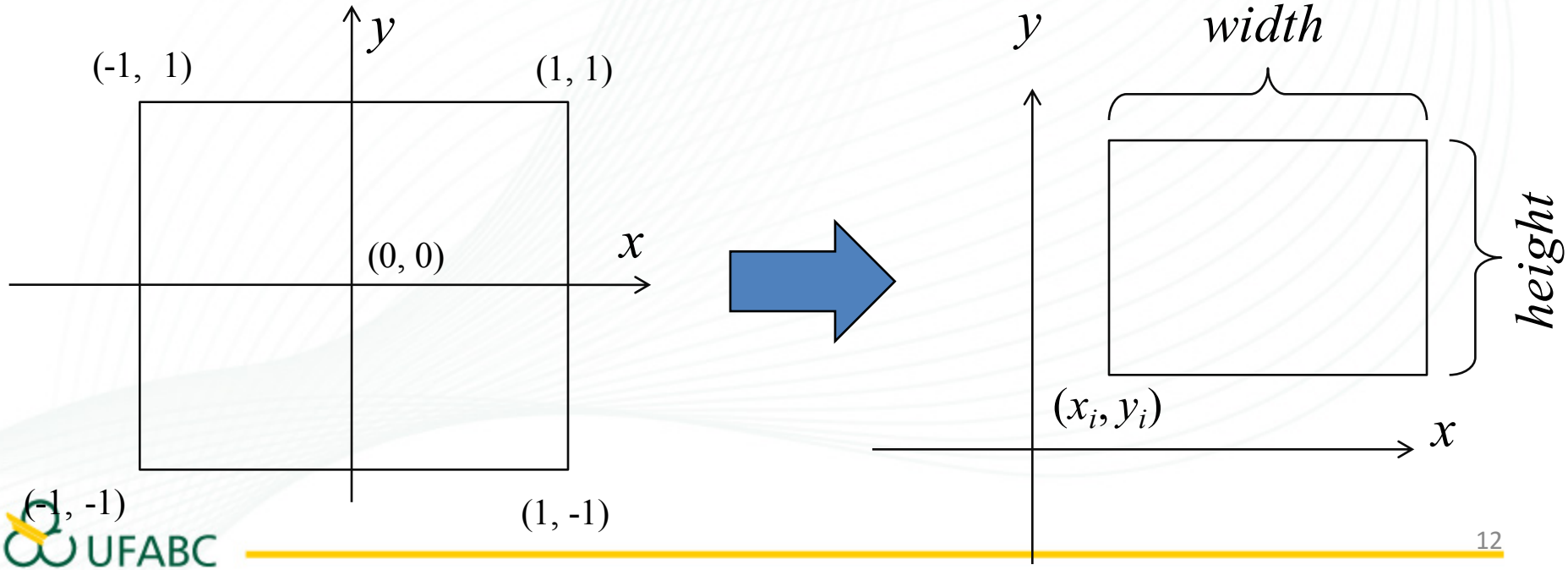
- O OpenGL faz isso automaticamente, depois que um vértice é processado pelo vertexShader

Window to Viewport

- Quais as transformações que levam do sistema de coordenadas 1 para o sistema de coordenadas 2?

Sistema 1

Sistema 2



Window to Viewport

- Em ordem cronológica:

  1. T(1, 1)
  2. S(*0.5*, *0.5*)
  3. S(*width*, *height*)
  4. T(x_i, y_i)

Window to Viewport

- Ou seja....

$$x' = (x + 1)\frac{width}{2} + x_i$$

$$y' = (y + 1)\frac{height}{2} + y_i$$

Window to Viewport

- Note que o sistema de coordenadas inicial tem uma razão de aspecto 1:1, ou seja, é um quadrado.

- Se definirmos uma viewport como sendo um retângulo, veremos que os objetos serão achatados ou esticados ao serem desenhados

- Sendo assim, por enquanto, é melhor definirmos uma viewport também quadrada

Window to Viewport

- Máximo quadrado dentro de uma janela retangular!
  - Função call-back da GLFW

```
void on_resize(GLFWwindow* window, int width, int height) {
    if (height<width) glViewport(0, 0, height, height);
    else glViewport(0, 0, width, width);
}
```

Biblioteca de funções geométricas 2D

Transformações 2D

- Vamos criar uma classe em C++ para transformações 2D

```
class mat33 {
public:
    mat33();  // Construtor
    ~mat33();  // Destrutor
    // Ponteiro público para os elementos da matrix
    const float *mptr;

    void identity();  // Carrega a matriz identidade
    void rotate(int degrees); // Multiplica à esquerda uma matriz de rotação
    void translate(float dx, float dy); // Multiplica à esquerda uma matriz de translação
    void scale(float sx, float sy); // Multiplica à esquerda uma matriz de escala
    void print();  // Imprime no console a matriz atual

private:
    array <float, 9> m;  // Matriz (vetor) principal
    array <float, 9> maux; // // Matriz (vetor) auxiliar
};
```

Transformações 2D

- Exemplo da rotação `void` **`rotate`**`(int theta)`

R(θ) **m**

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ m_6 & m_7 & m_8 \end{bmatrix}$$

$$\begin{bmatrix} \cos(\theta)m_0 - \sin(\theta)m_3 & \cos(\theta)m_1 - \sin(\theta)m_4 & \cos(\theta)m_2 - \sin(\theta)m_5 \\ \sin(\theta)m_0 + \cos(\theta)m_3 & \sin(\theta)m_1 + \cos(\theta)m_4 & \sin(\theta)m_2 + \cos(\theta)m_5 \\ m_6 & m_7 & m_8 \end{bmatrix}$$

m <= maux

Transformações 2D

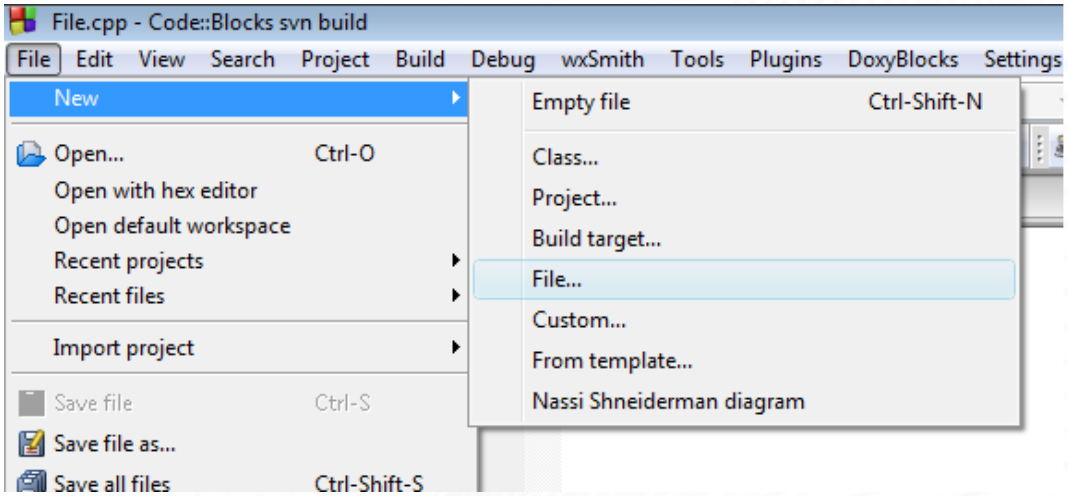
- Função completa

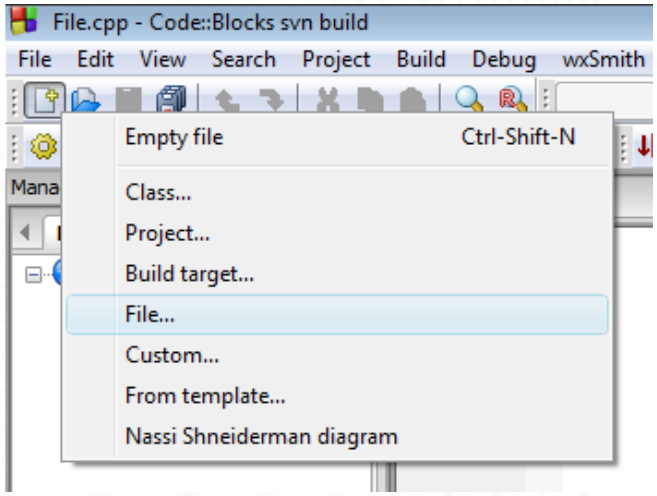
```
void mat33::rotate(int degrees) {
        float rads = degrees*PI/180;
        float c = cos(rads);
        float s = sin(rads);

        maux = {c*m[0]-s*m[3],  c*m[1]-s*m[4], c*m[2]-s*m[5],
                s*m[0]+c*m[3],  s*m[1]+c*m[4], s*m[2]+c*m[5],
                m[6],           m[7],          m[8]
         };

        m = maux;
}
```

Transformações 2D

- Tarefa: Baixar os arquivos mat33.hpp e mat33.cpp no Repositório do Tidia;
- Incluir os arquivos baixados no projeto OpenGL desenvolvido na "Atividade 01”
- Implementar as demais funções.

Incluir os arquivos no projeto





Incluir arquivos no projeto

- No topo do arquivo main.cpp, inclua

#include “mat33.hpp"

Implementar funções

- No arquivo mat33.cpp, as funções identity e rotate já estão implementadas.

Implementar funções

```
void mat33::identity() {
    m = {1, 0, 0, 0, 1, 0, 0, 0, 1};
    mptr = m.data();
}

void mat33::rotate(int degrees) {
    float rads = degrees*PI/180;
    float c = cos(rads);
    float s = sin(rads);
    maux = {c*m[0]-s*m[3],  c*m[1]-s*m[4], c*m[2]-s*m[5],
            s*m[0]+c*m[3],  s*m[1]+c*m[4], s*m[2]+c*m[5],
            m[6],           m[7],          m[8]
           };
    m=maux;
}
```

Implementar funções

- Implementar as funções:

  void mat33::translate(float dx, float dy);

  void mat33::scale(float sx, float sy);

Links

- Incluir arquivos no projeto:

http://wiki.codeblocks.org/index.php/Creating_a_new_project

- Definições mútiplas de função:

http://stackoverflow.com/questions/17904643/error-with-multiple-definitions-of-function

- Exemplos de implementações de funções de transformação:

http://www.programming-techniques.com/2012/03/3d-transformation-translation-rotation.html

Funções implementadas

- As funções implementadas no arquivo “mat33.cpp” serão utilizadas em alguma atividade que será entregue.

Fim da Aula 13

André Luiz Brandão