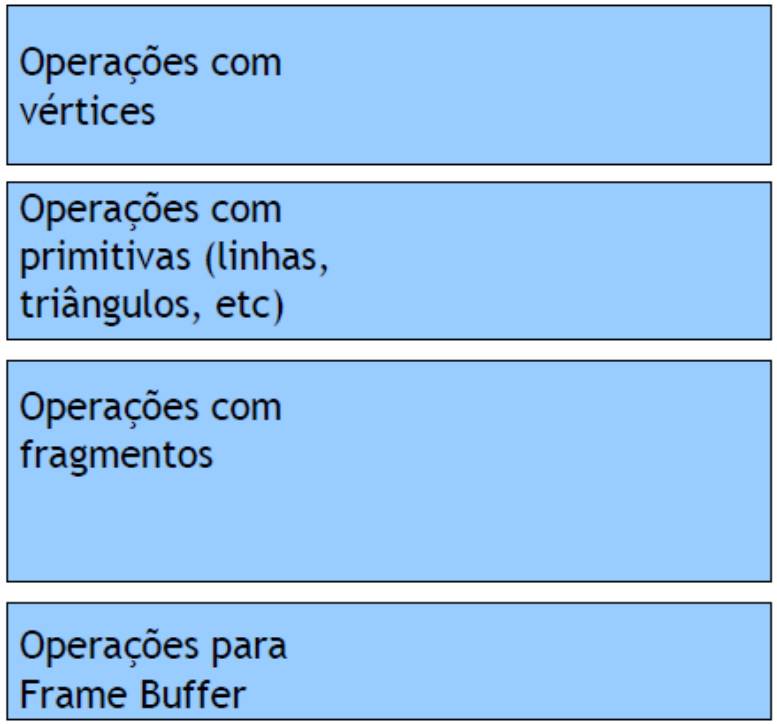
Computação Gráfica

Aula 02 – Pipeline gráfico e primitivas geométricas

Prof. Jean R. Ponciano

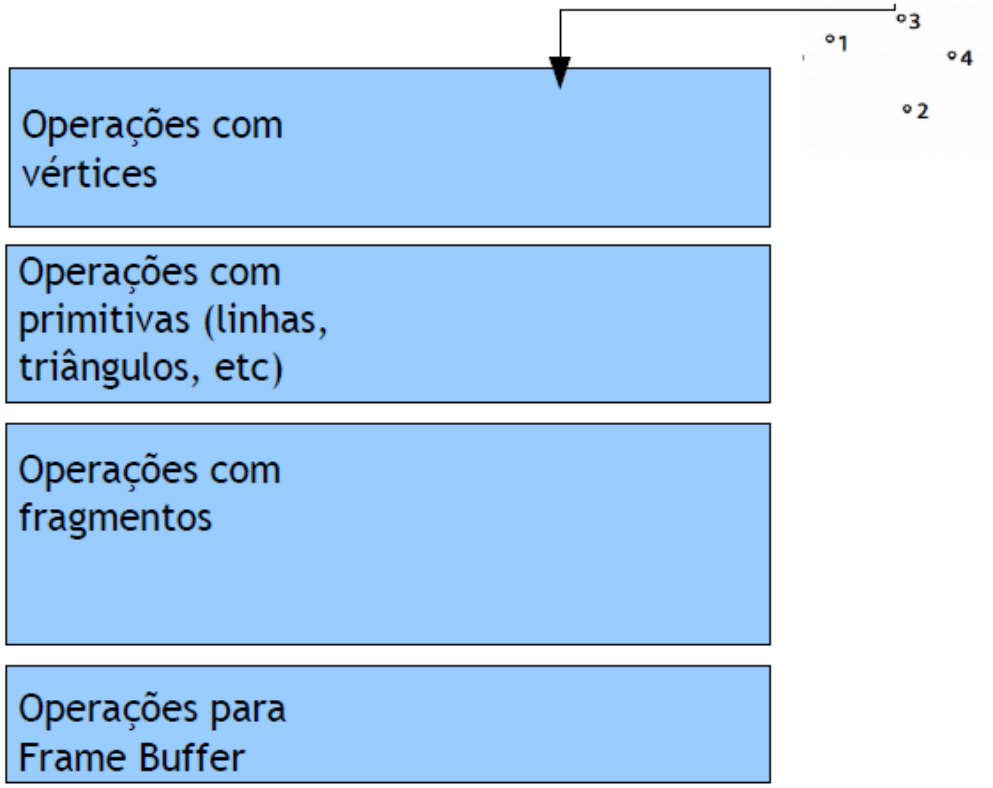
Pipeline gráfico

- Descreve as etapas envolvidas para renderizar um objeto 2D ou 3D.
- Modelo Conceitual simplificado:



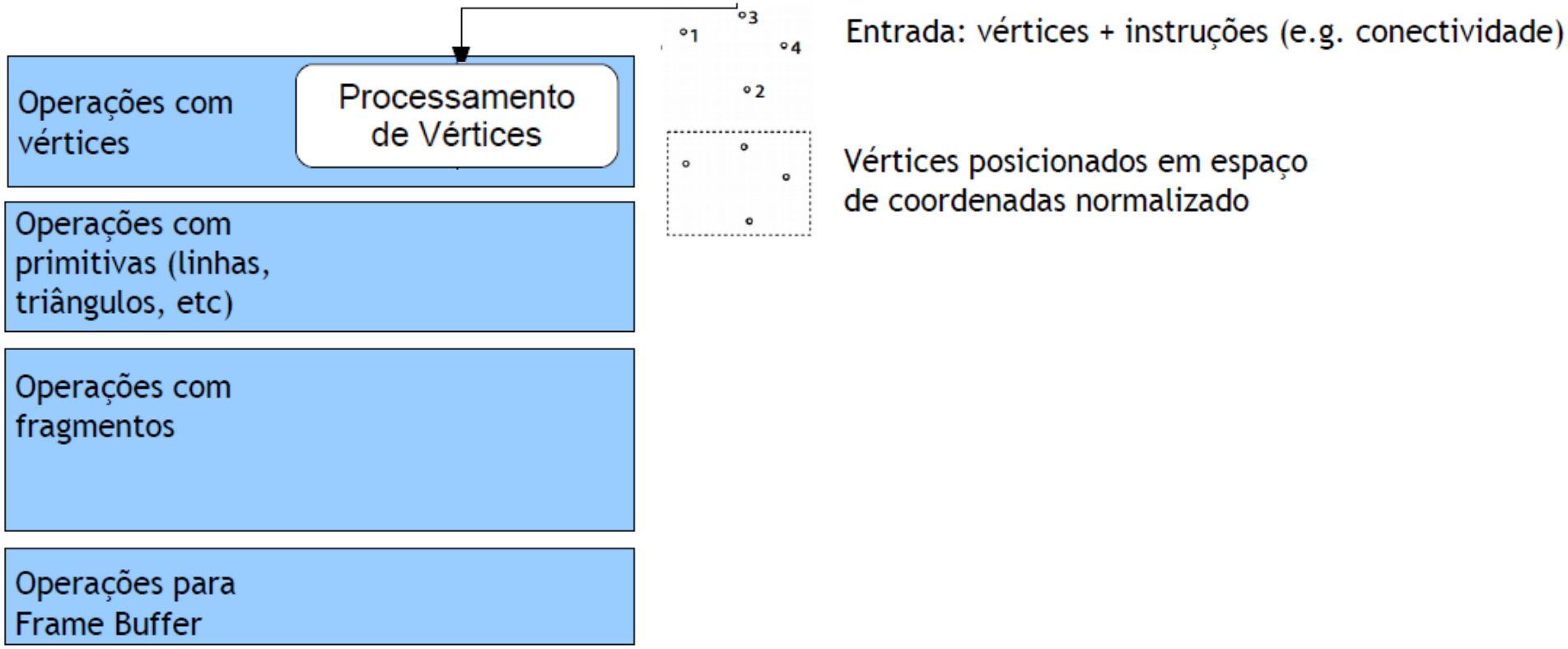
Pipeline gráfico

- Descreve as etapas envolvidas para renderizar um objeto 2D ou 3D.
- Modelo Conceitual simplificado:



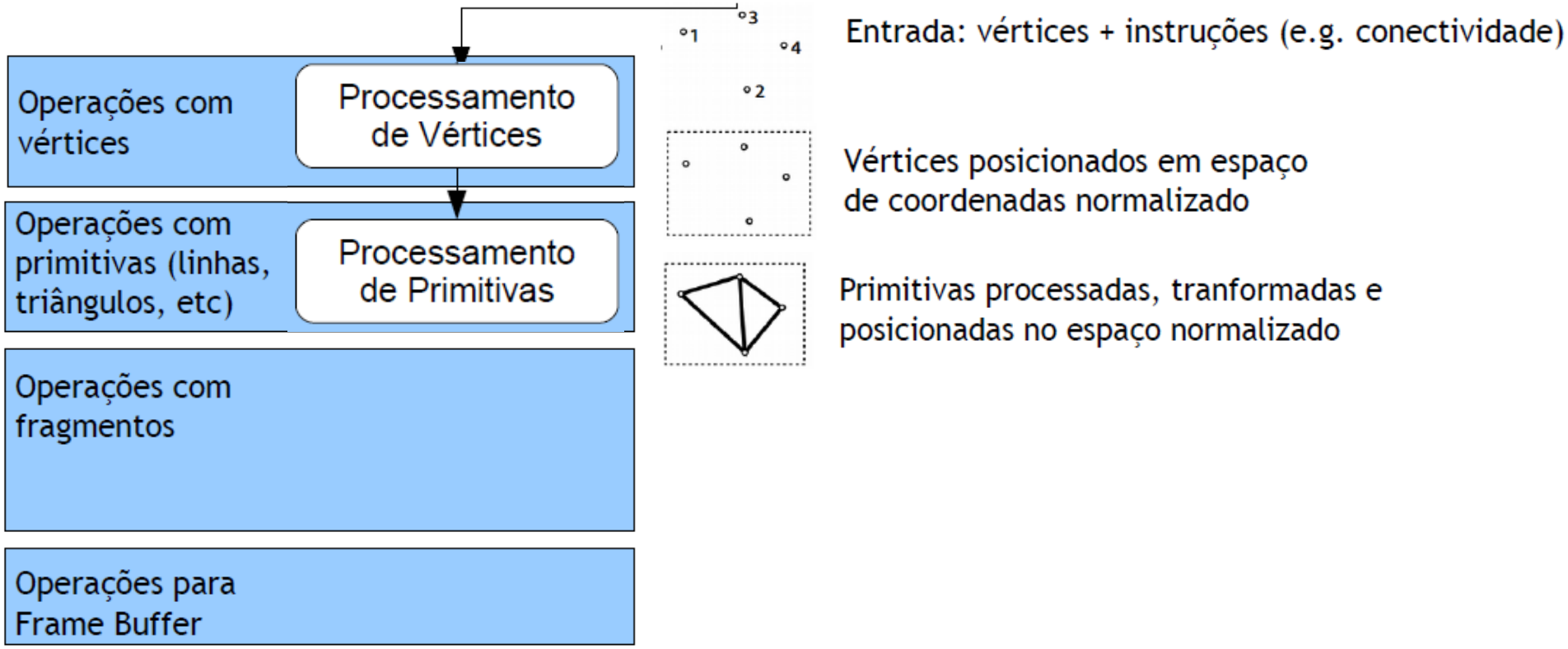
Pipeline gráfico

- Descreve as etapas envolvidas para renderizar um objeto 2D ou 3D.
- Modelo Conceitual simplificado:



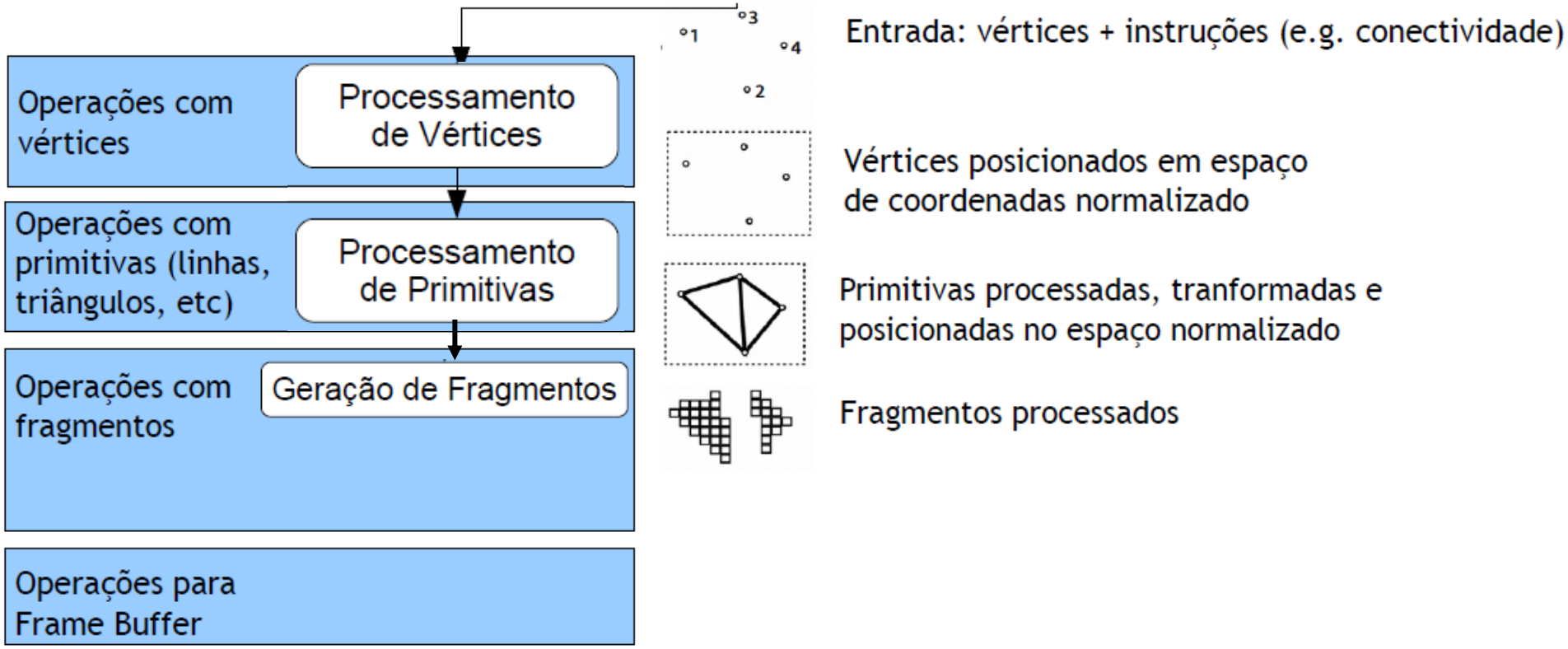
Pipeline gráfico

- Descreve as etapas envolvidas para renderizar um objeto 2D ou 3D.
- Modelo Conceitual simplificado:



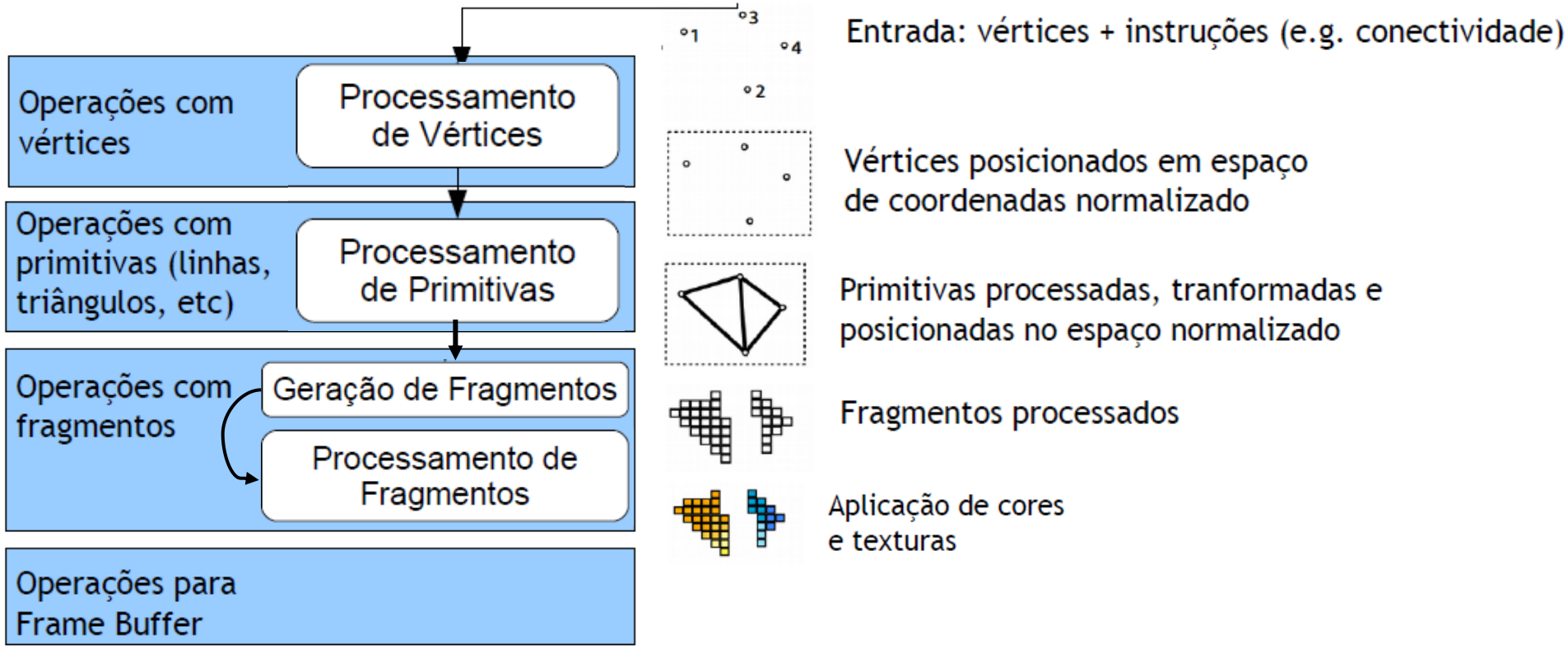
Pipeline gráfico

- Descreve as etapas envolvidas para renderizar um objeto 2D ou 3D.
- Modelo Conceitual simplificado:



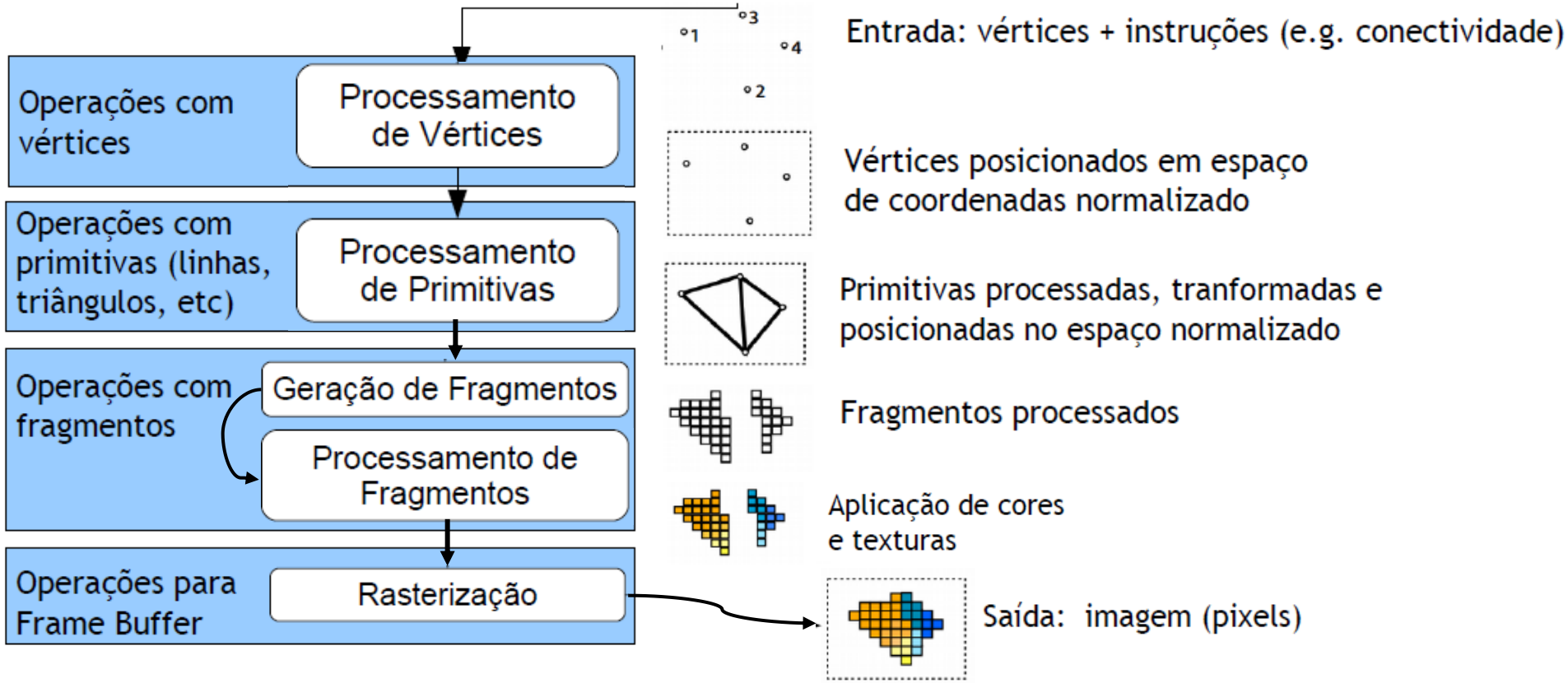
Pipeline gráfico

- Descreve as etapas envolvidas para renderizar um objeto 2D ou 3D.
- Modelo Conceitual simplificado:



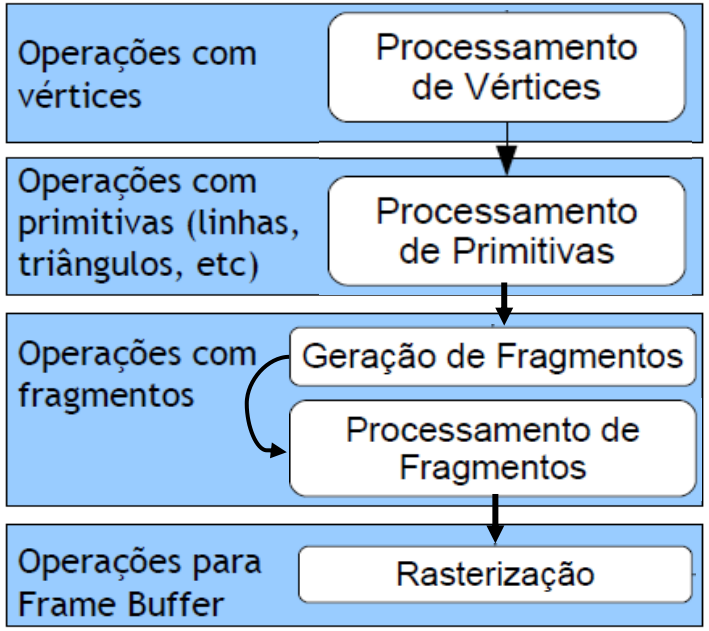
Pipeline gráfico

- Descreve as etapas envolvidas para renderizar um objeto 2D ou 3D.
- Modelo Conceitual simplificado:



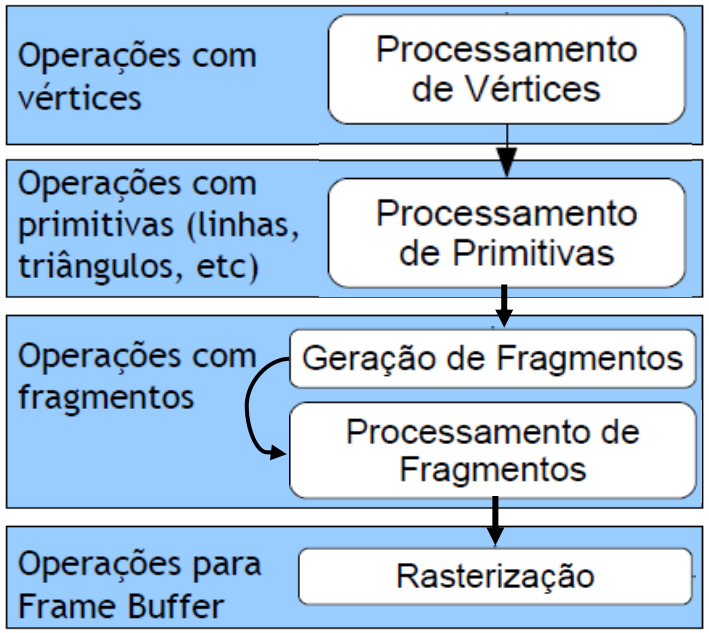
Pipeline gráfico **Fixo**

- Padrão até 2003, com renderização dependente de algoritmos internos da GPU
- Sem liberdade para customizar ou substituir os algoritmos
- Atualmente obsoleto!



Pipeline gráfico **Fixo**

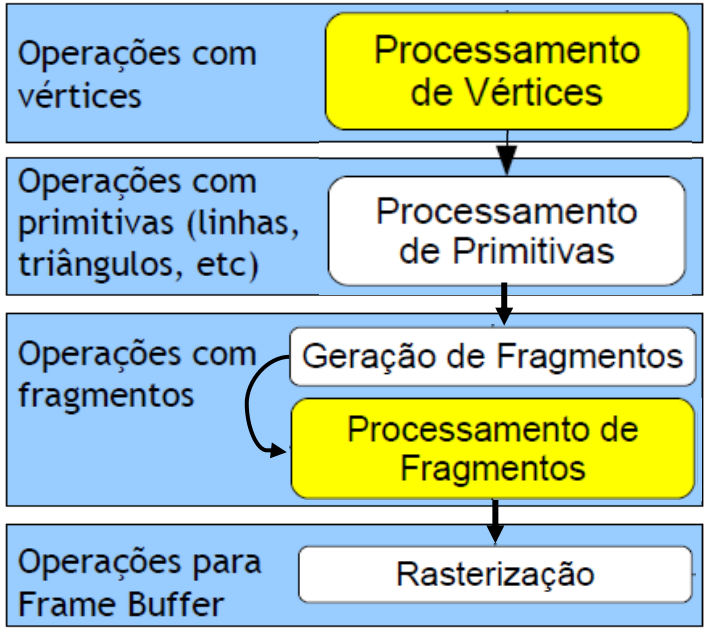
- Padrão até 2003, com renderização dependente de algoritmos internos da GPU
- Sem liberdade para customizar ou substituir os algoritmos
- Atualmente obsoleto!



As operações do pipeline eram fixas e limitadas a estilos específicos de processamento de vértices e de fragmentos

Pipeline gráfico **Programável**

- A partir de 2004, inclusão de **shaders** (i.e., etapas) programáveis (de vértices e de fragmentos --- existe outro também, mas ele é opcional)
- Linguagem de programação própria (por ex., GLSL para a OpenGL)

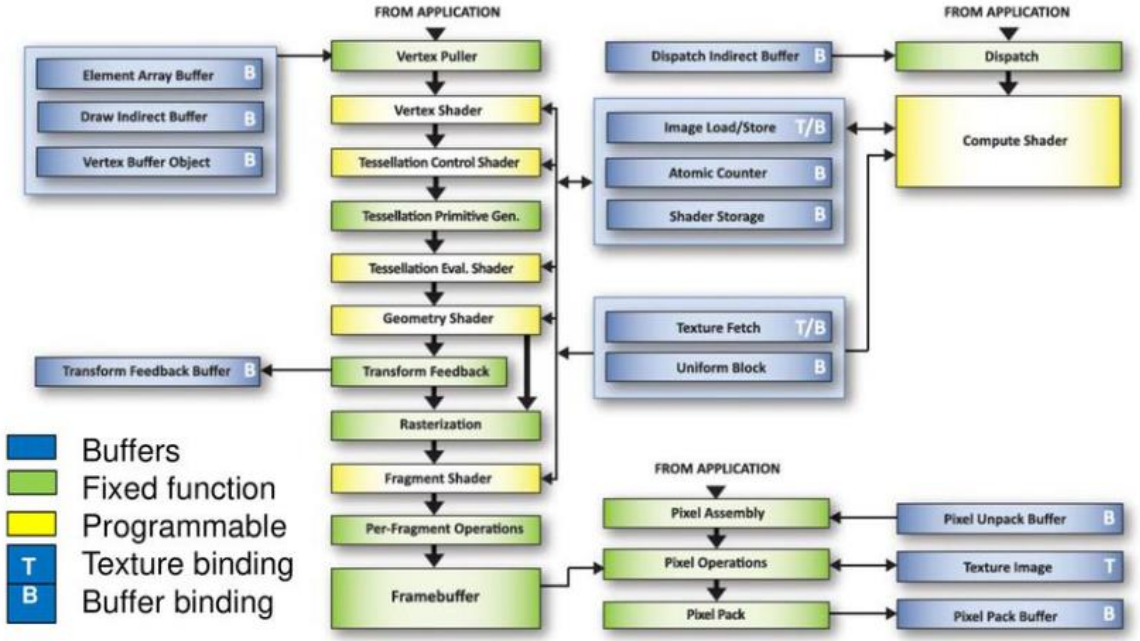


Muitas técnicas que não poderiam ser utilizadas em tempo real por não fazerem parte do pipeline fixo (só pós-processamento) agora podem ser feitas em tempo real.

APIs para Computação Gráfica

- Existem várias APIs para CG
  - Exemplos: OpenGL, DirectX, WebGL, Vulkan, …
  - Cada uma implementa um pipeline gráfico diferente

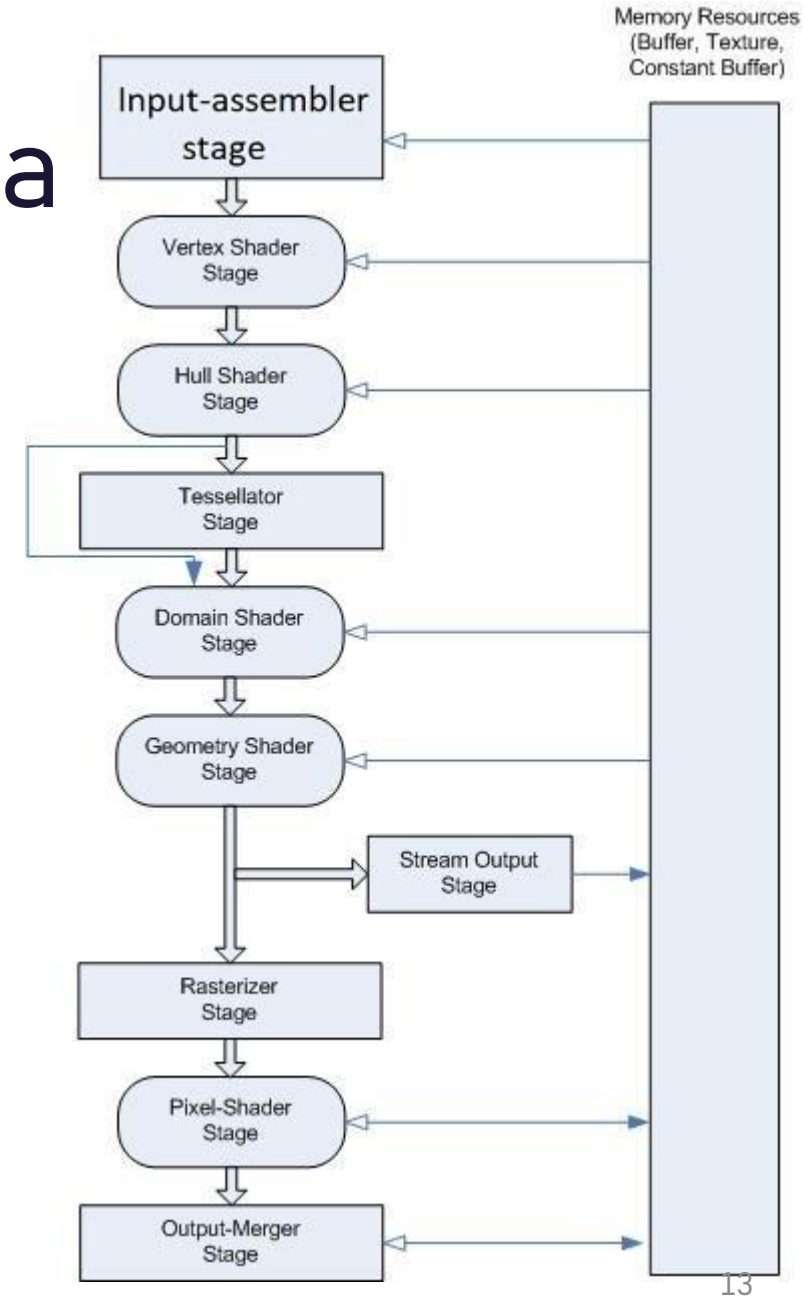
Exemplo: Pipeline
Gráfico da OpenGL 4.6



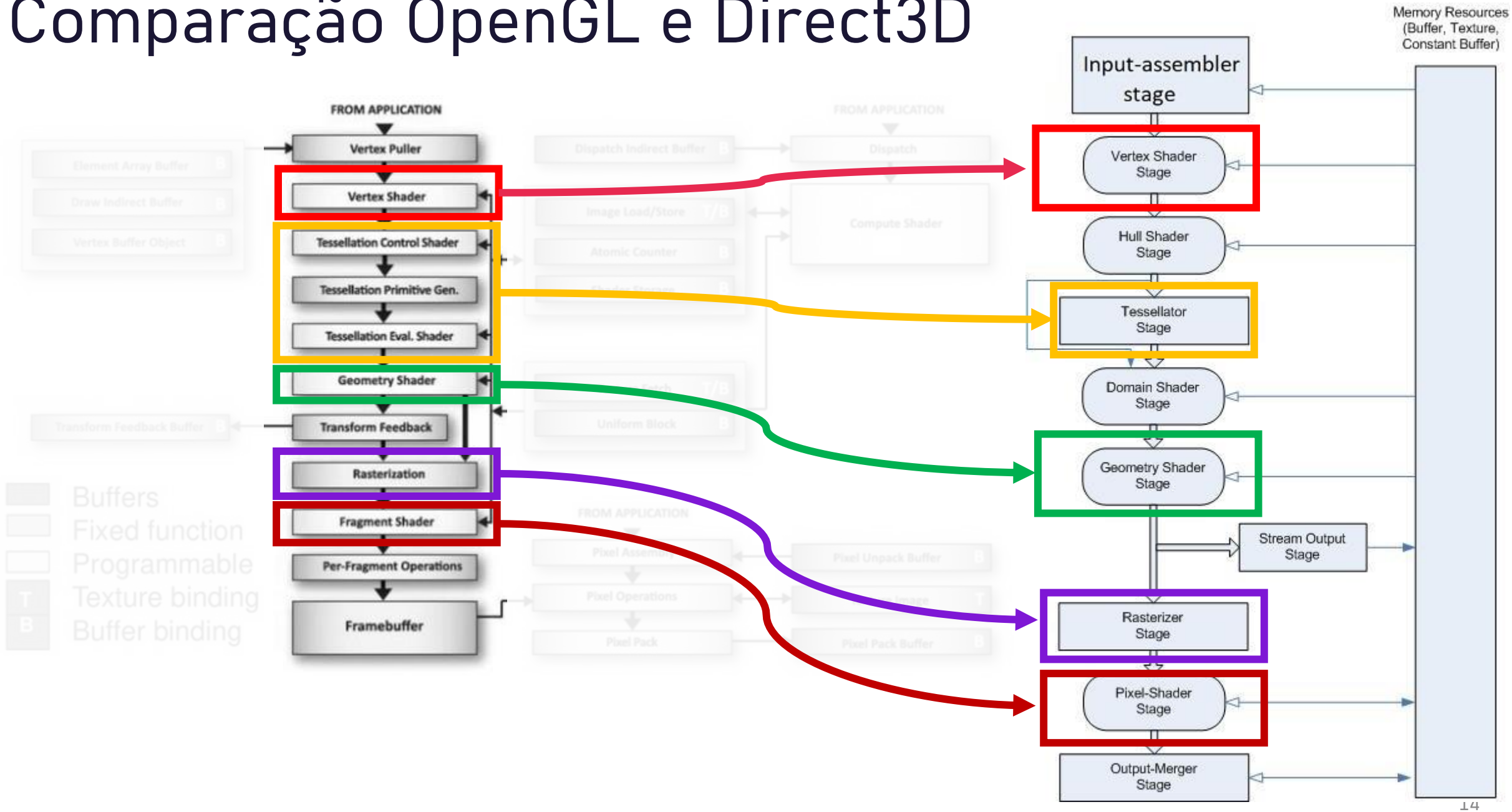
APIs para Computação Gráfica

- Pipeline Gráfico do Direct3D 11

https://learn.microsoft.com/en-us/windows/win32/direct3d11/overviews-direct3d-11-graphics-pipeline



Comparação OpenGL e Direct3D



APIs para Computação Gráfica

- **Usaremos OpenGL na nossa disciplina**
  - API disponível para muitas linguagens de programação (link)
  - Multiplataforma
  - Independente do sistema de janelas
  - Documentação: link

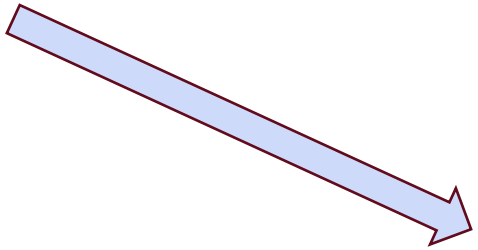
OpenGL – Breve Histórico

- OpenGL 1 lançada em 1994 (pipeline fixo)
- OpenGL 2 lançada em 2004 (shaders (i.e., etapas) programáveis)
- OpenGL 3 adicionou mecanismo de remoção de funcionalidades obsoletas e compatibilidade
- OpenGL 4 lançou novos shaders (tecelagem/*telessation*)
  - Shader de geometria lançado na OpenGL 3.2.

- Também existem OpenGL ES para dispositivos móveis e WebGL para Javascript (navegadores) (ver www.webglsamples.org)

Sistema de janelas

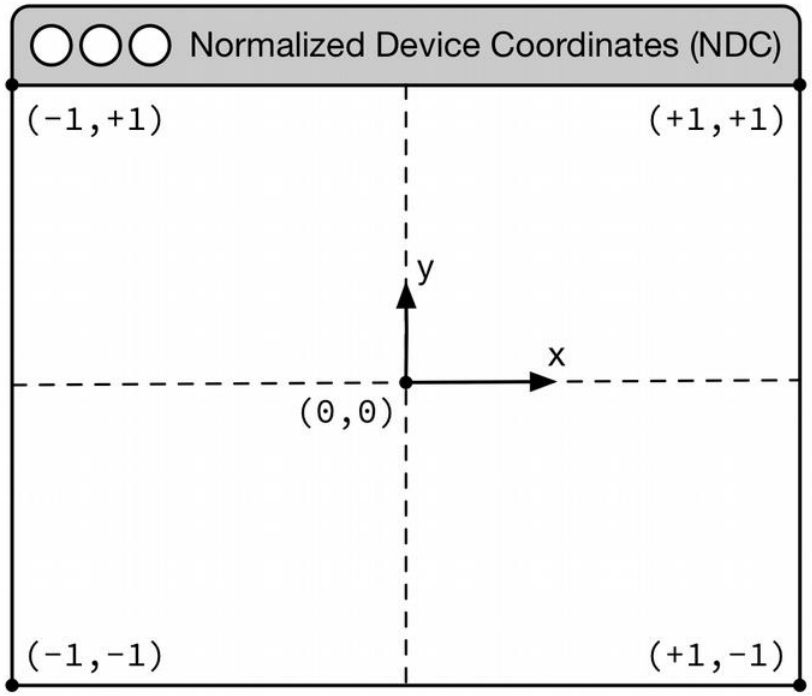
- Uma aplicação com OpenGL precisa de uma janela para renderizar a imagem
  - GLUT (obsoleto) e FreeGLUT
  - GLFW
  - QT
  - SDL



Usaremos essa
[www.glfw.org]

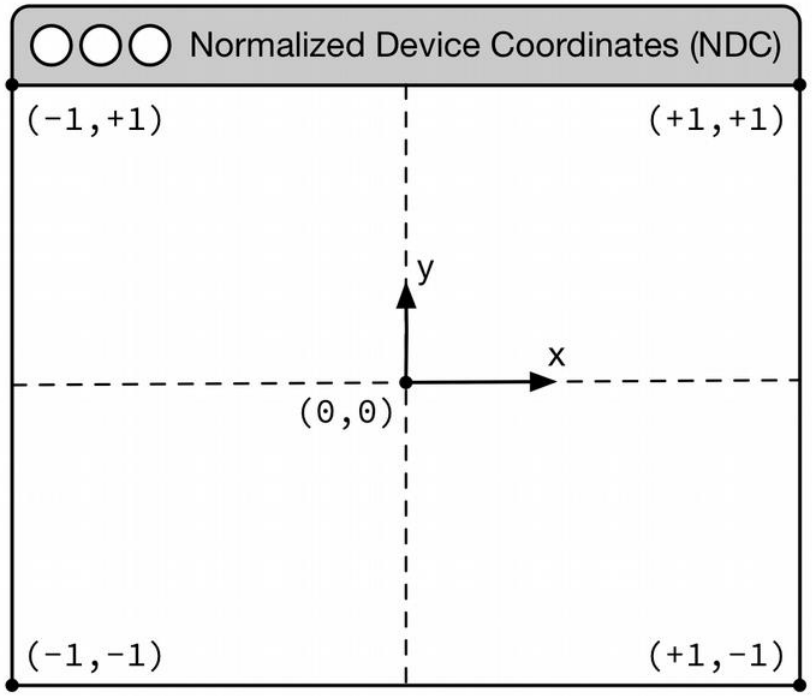
Janelas com OpenGL

- Para o OpenGL, as janelas utilizam **coordenadas x, y, z entre 0 e 1**.
  - Sistema de coordenadas cartesiano
  - O centro da janela corresponde à coordenada (0,0,0)



Janelas com OpenGL

- Para o OpenGL, as janelas utilizam **coordenadas x, y, z entre 0 e 1**.
  - Sistema de coordenadas cartesiano
  - O centro da janela corresponde à coordenada (0,0,0)



Nossa primeira janela:
Aula02.Ex01 (Jupyter Notebook)

Primitivas Geométricas

- Para renderizar alguma imagem, precisamos utilizar primitivas geométricas
  - Elementos gráficos simples que levam a objetos complexos quando combinados
  - 2D: ponto, linha, polilinha, polígono, elipse, arco
  - 3D: cubo, esfera, cilindro, cone, tubo.

- Tudo começa com o vértice:
  - Ponto representado por 4 coordenadas (x,y,z,w).
  - Motivo: coordenadas homogêneas (veremos em aula futura!)

Primitivas Geométricas

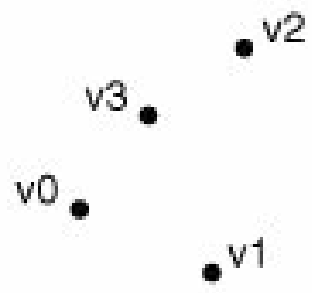
- Para renderizar alguma imagem, precisamos utilizar primitivas geométricas
  - Elementos gráficos simples que levam a objetos complexos quando combinados
  - 2D: ponto, linha, polilinha, polígono, elipse, arco
  - 3D: cubo, esfera, cilindro, cone, tubo.

Aula02.Ex02 (Jupyter Notebook)

- Tudo começa com o vértice:
  - Ponto representado por 4 coordenadas (x,y,z,w).
  - Motivo: coordenadas homogêneas (veremos em aula futura!)

Linhas

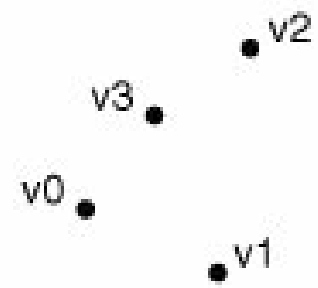
- Três formas de renderizar linhas.

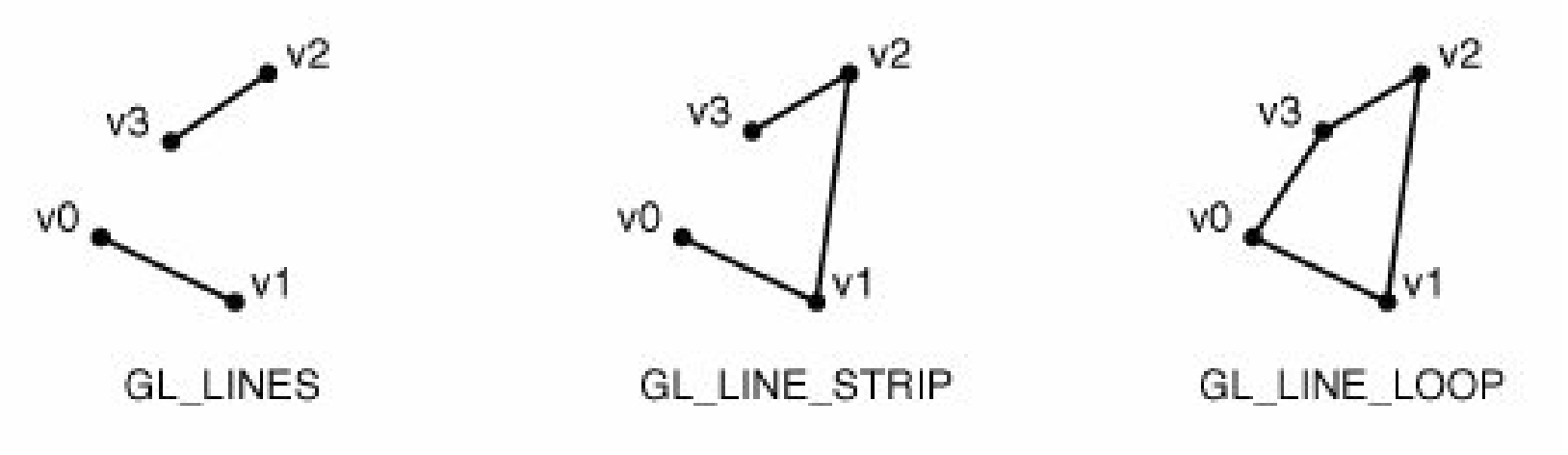


- GL_LINES: Renderiza a cada dois vértices
  - (v0,v1),(v2,v3)
- GL_LINE_STRIP: Renderiza de forma encadeada
  - (v0,v1),(v1,v2),(v2,v3)
- GL_LINE_LOOP: Renderiza de forma encadeada até atingir o primeiro vértice
  - (v0,v1),(v1,v2),(v2,v3),(v3,v0)

Linhas

- Três formas de renderizar linhas.

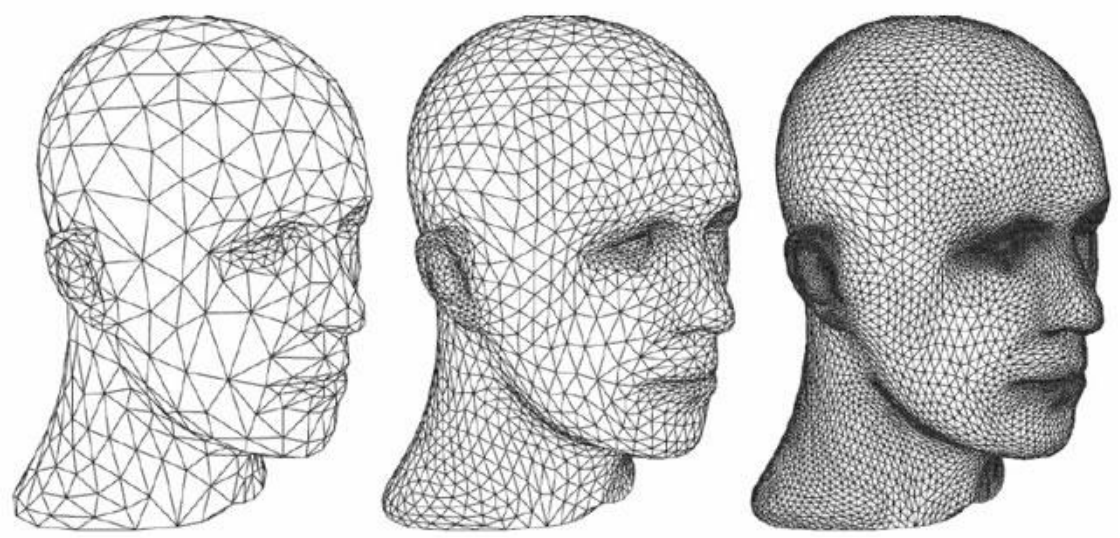




Aula02.Ex03 (Jupyter Notebook)

Triângulos

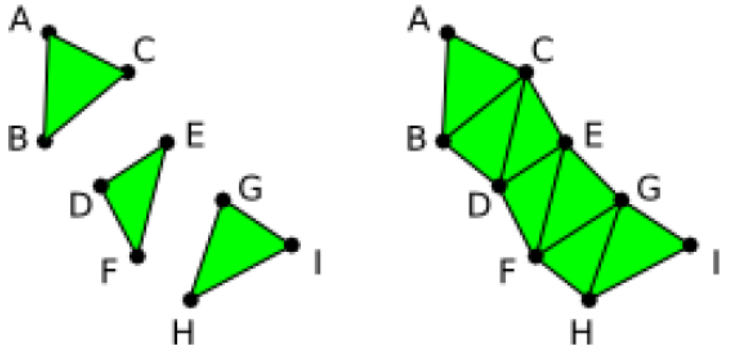
- Triângulos são primitivas importantes para a computação gráfica moderna.
- A GPU divide malhas em pequenos triângulos antes de desenhá-las: tecelagem (ou tesselação).



Triângulos

- Triângulos são primitivas importantes para a computação gráfica moderna.
- A GPU divide malhas em pequenos triângulos antes de desenhá-las: tecelagem (ou tesselação).

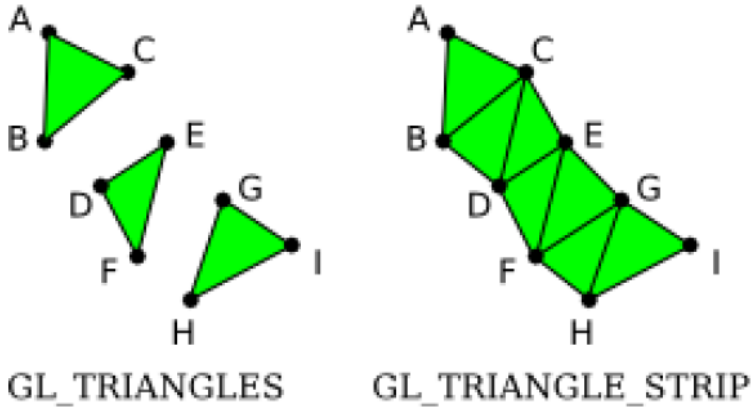
- Algumas primitivas para triângulos (existem outras):



Triângulos

- Triângulos são primitivas importantes para a computação gráfica moderna.
- A GPU divide malhas em pequenos triângulos antes de desenhá-las: tecelagem (ou tesselação).

- Algumas primitivas para triângulos (existem outras):

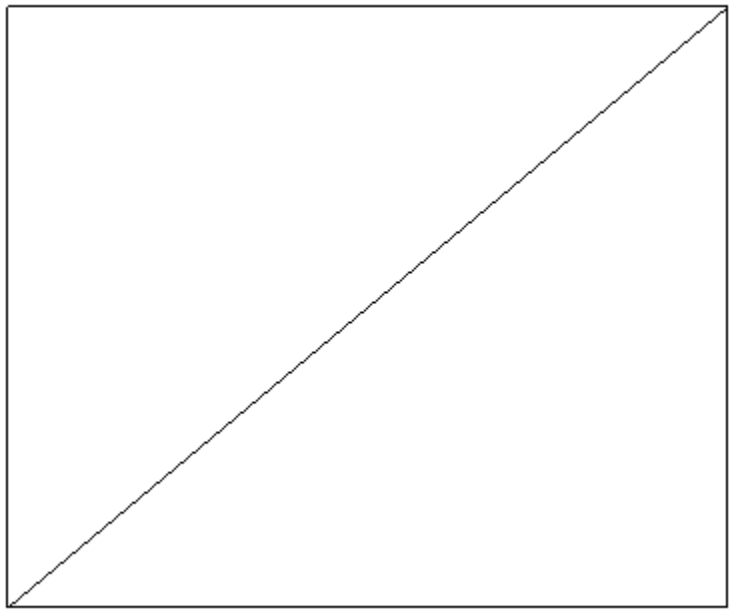


Aula02.Ex04
(Jupyter Notebook)

Quadrados e Círculos

- Podem ser desenhados usando triângulos!

```
while not glfw.window_should_close(window):

    glfw.poll_events()



    glPolygonMode(GL_FRONT_AND_BACK,GL_LINE
    glClear(GL_COLOR_BUFFER_BIT)
    glClearColor(1.0, 1.0, 1.0, 1.0)

    glDrawArrays(GL_TRIANGLE_STRIP, 0, 4)

    glfw.swap_buffers(window)

glfw.terminate()
```

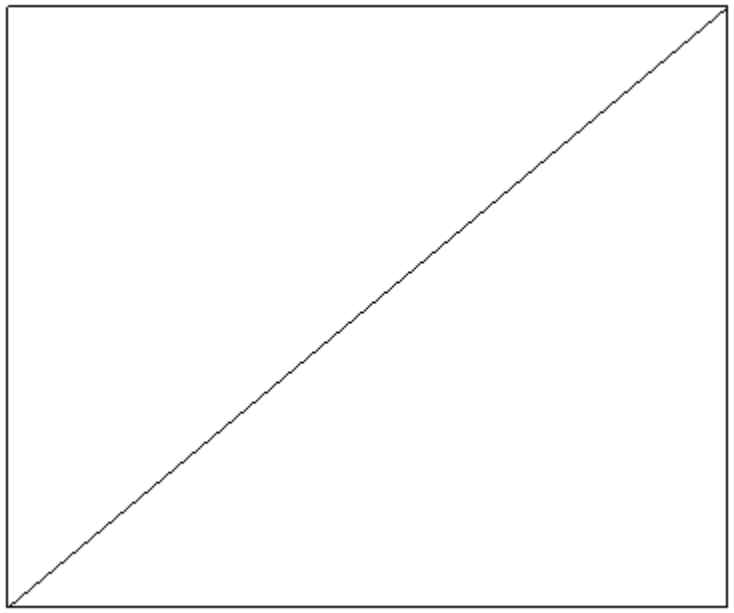


Quadrados e Círculos

- Podem ser desenhados usando triângulos!

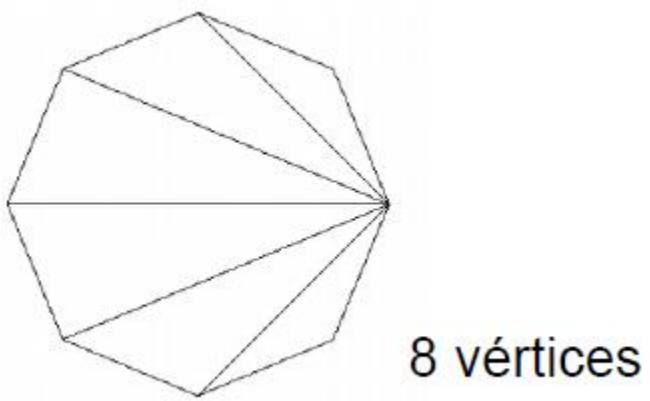
Vejam depois:
Aula02.Ex05
(Jupyter Notebook)

```
while not glfw.window_should_close(window):

    glfw.poll_events()



    glPolygonMode(GL_FRONT_AND_BACK,GL_LINE
    glClear(GL_COLOR_BUFFER_BIT)
    glClearColor(1.0, 1.0, 1.0, 1.0)

    glDrawArrays(GL_TRIANGLE_STRIP, 0, 4)

    glfw.swap_buffers(window)

glfw.terminate()
```

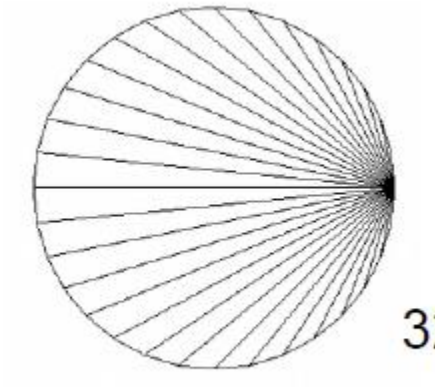


Quadrados e Círculos

- Podem ser desenhados usando triângulos!

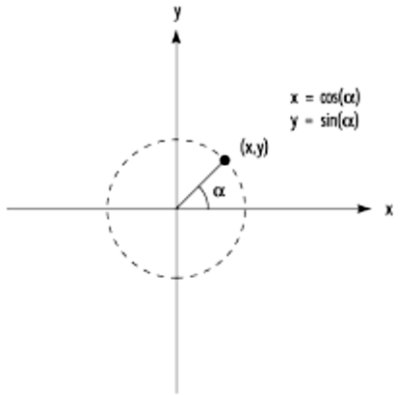


8 vértices



32 vértices

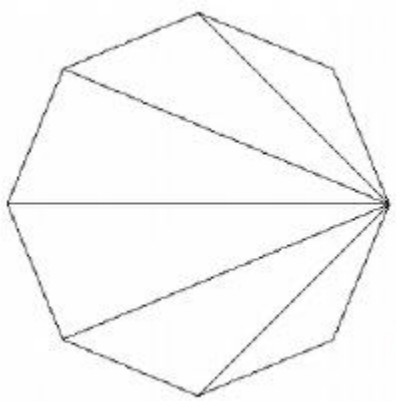
- Primitiva: GL_TRIANGLE_FAN
- Para determinar as coordenadas dos vértices...



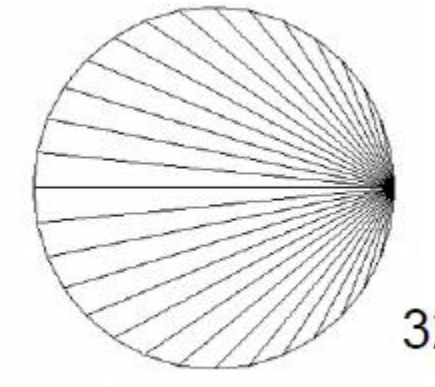
Quadrados e Círculos

Aula02.Ex06
(Jupyter Notebook)

- Podem ser desenhados usando triângulos!

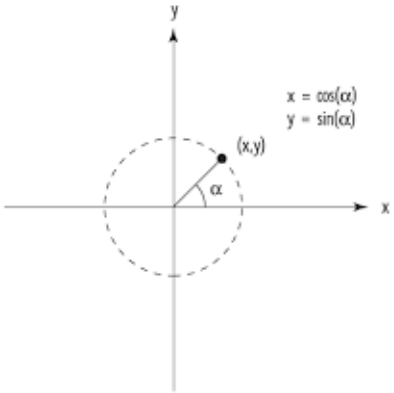


8 vértices



32 vértices

- Primitiva: GL_TRIANGLE_FAN
- Para determinar as coordenadas dos vértices...



Cores

Aula02.Ex07
(Jupyter Notebook)

Ver depois:
Aula02.Ex07 - Extra
(Jupyter Notebook)

Aula02.Ex08
(Jupyter Notebook)

Bibliografia

- Essa aula foi baseada no seguinte material:

  - Computação Gráfica ICMC/USP. Profs. Ricardo Marcacini e Alaor Cervati Neto.

  - SHREINER, Dave et al. OpenGL programming guide: The Official guide to learning OpenGL, version 4.3. Addison-Wesley, 2013.