Pipeline Programável

Thiago Luange Gomes

Introdução

- Nas décadas passadas os hardwares gráficos sofreram grandes transformações.
- A NVIDIA estabeleceu o termo GPU(graphics processing unit) em 1999 para diferenciar a GeForce 256 dos chips anteriores que disponibilizavam somente rasterização.

Introdução

- Uma das mais significantes mudanças foi a introdução de componentes programáveis no pipeline.
- Em 2001, a NVIDIA GeForce 3 foi a primeira a suportar componentes programáveis(shaders) usando DirectX 8.0 e extensões da OpenGL.

Introdução

- Um **shader** pode ser visto como um programa que será executado em algum estágio do pipeline gráfico.
- No DirectX 9.0(2002) foi incluído uma nova linguagem de shader chamada HLSL(Hight Level Shading Language).

Introdução

- HLSL foi desenvolvida pela Microsoft com colaboração da NVIDIA. A NVIDIA também lançou uma variante multi-plataforma chamada Cg.
- Um tempo depois a OpenGL lançou uma linguagem similar chamada GLSL (OpenGL Shading Language).

“Uma nova velha ideia”

- Uma das primeiras linguagens de shader foi a descrita por Rob Cook em 1984. Ele discutia uma linguagem para integrar o processo de sombreamento(shading) e técnicas de texturas através de uma estrutura de árvore(shade trees).

“Uma nova velha ideia”

- Perlin’s Pixel Stream Editor: Desenvolvida por Ken Perlin em 1985.
- RenderMan: Desenvolvida pela Pixar por volta de 1988, baseada no trabalho de Rob Cook.

OpenGL

- Versão 1.0 (1992)
- Versão 1.1 (1997) – Vertex Array
- Versão 1.2 (1998) – Textura 3D
- Versão 1.3 (2001) – Multi-textura, etc.
- Versão 1.4 (2002) – Mipmap automático, etc.
- Versão 1.5 (2003) – Vertex buffer e shaders por meio de extensões.
- Versão 2.0 (2004) – Inclusão da Linguagem de shading

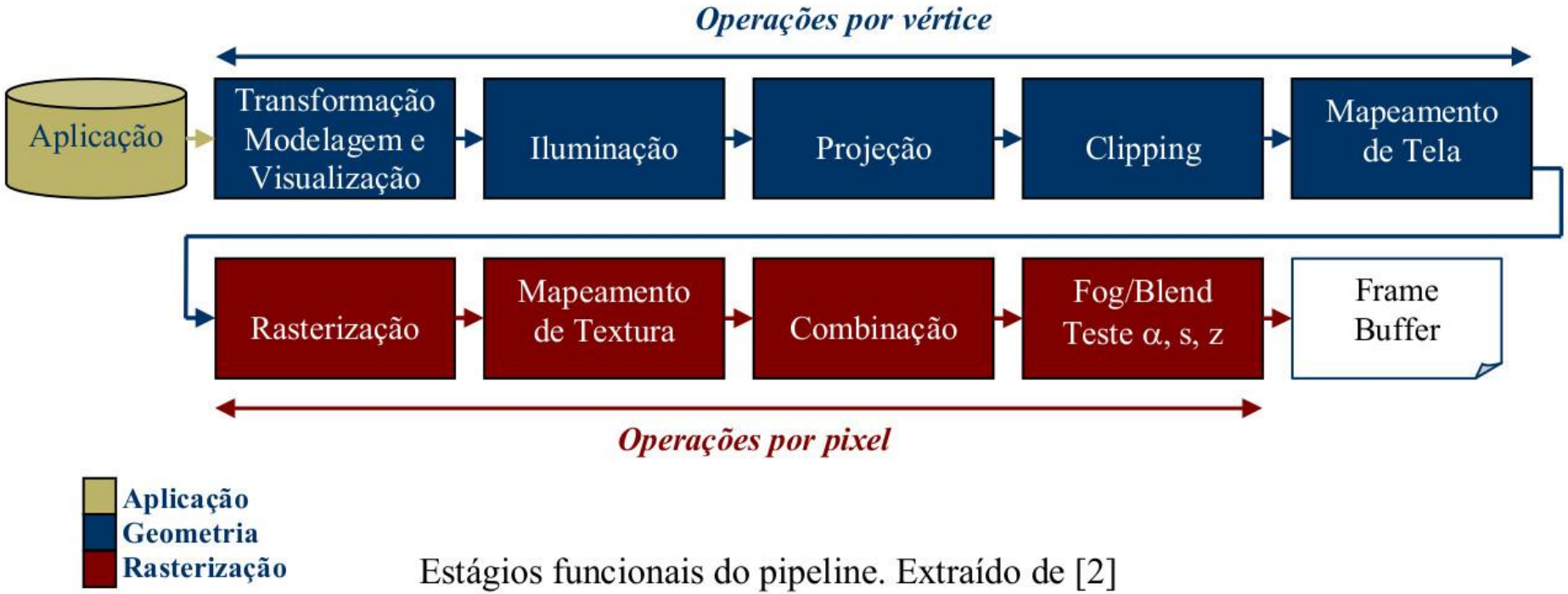
OpenGL

- Versão 3.0 (2008) - 112 funções tornaram-se “deprecated”, restando 126 funções. A programação por shader torna-se obrigatória.
- Versão 3.1 (2009) – As funções “deprecated” foram removidas da API.
- Versão 3.2 (2009) – Introduz os contextos: principal e compatibilidade. Este último para incluir as funções “deprecated”.

OpenGL

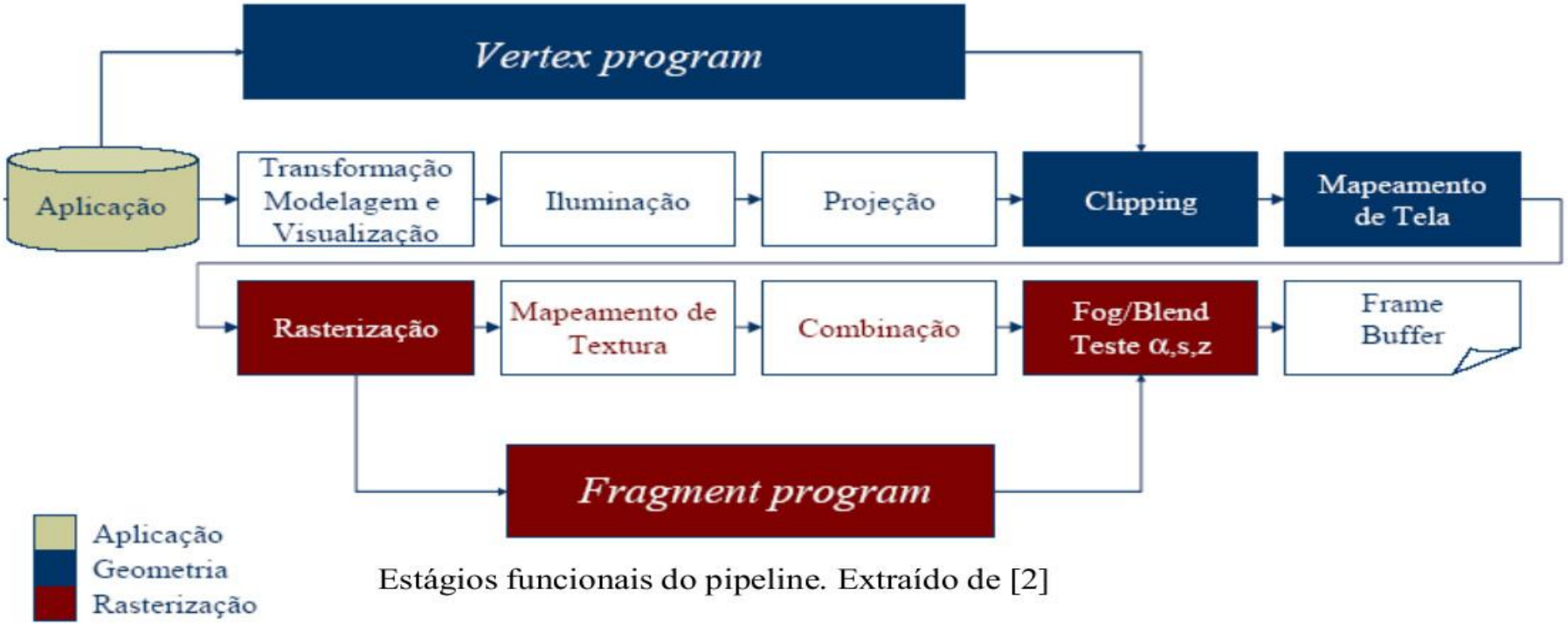
- Versão 3.3 (2010) – Lançada simultaneamente com a versão 4.0. Traz recursos avançados para as GPUs antigas. Mas não inclui os novos estágios do pipeline programável.
- Versão 4.0 (2010) – Adiciona dois novos estágios ao pipeline: Tesselation Control e Tesselation Evaluation.
- Versão 4.3 (2012) – Inclui o compute shader.
- Versão 4.5 (2014) – Última versão lançada até o momento

Pipeline Fixo



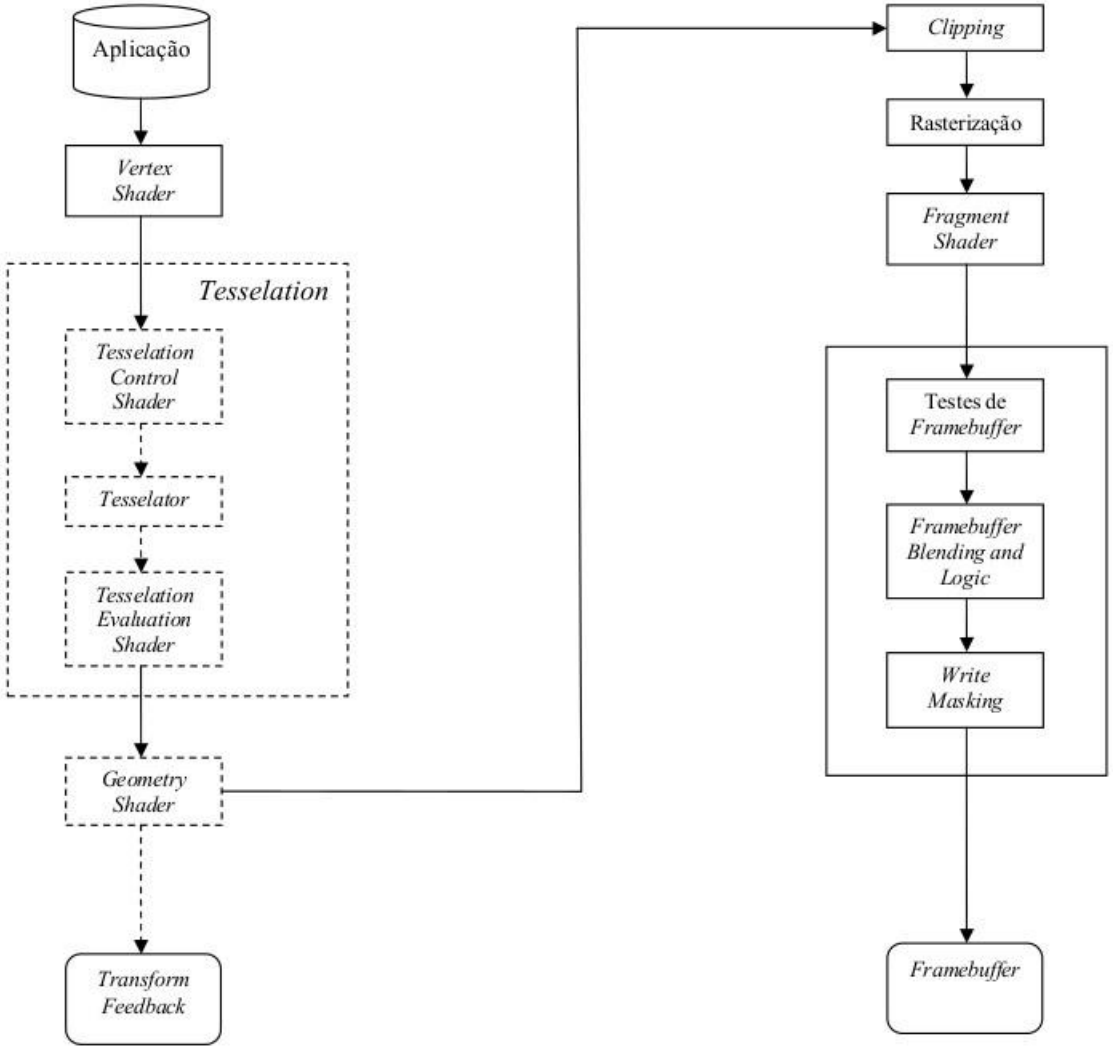
Estágios funcionais do pipeline. Extraído de [2]

Pipeline Programável



Estágios funcionais do pipeline. Extraído de [2]

OpenGL 4.x:


Aplicação
Vertex Shader
Tesselation
Tesselation Control Shader
Tesselator
Tesselation Evaluation Shader
Geometry Shader
Transform Feedback
Clipping
Rasterização
Fragment Shader
Testes de Framebuffer
Framebuffer Blending and Logic
Write Masking
Framebuffer

Vertex Shader

- Operam sobre os vértices e informações associadas, tais como cores, normais, etc.
- Operações possíveis:
  - Transformações nos Vértices, Transformações de Normais e Normalização, Iluminação, Geração de Coordenadas de Textura, etc.

Vertex Shader

- Cada vértice é trabalhado isoladamente, sem conhecimento dos vértices restantes.
- Deve-se obrigatoriamente calcular uma posição. Cor e outros atributos são opcionais.
- Tem-se acesso ao estado do OpenGL atual, incluindo algumas variáveis pré-calculadas pelo próprio OpenGL.

Fragment Shader

- Operam sobre os fragmentos que são produzidos pelo processo de rasterização.
- Operações possíveis:
  - Cálculo de cores e coordenadas de textura por pixel, Acesso e Aplicação de Texturas, FOG, Cálculo de normais no caso de iluminação por pixel, etc.

Fragment Shader

- Operam isoladamente em um único fragmento, sem qualquer informação sobre os fragmentos vizinhos.
- Fragmentos podem ser descartados
- As coordenadas de um fragmento não podem ser modificadas.
- Tem acesso ao estado do OpenGL atual, incluindo algumas variáveis pré-calculadas pelo próprio OpenGL, mas não tem acesso (de leitura) ao framebuffer.

Com fazer um shader?

- OpenGL Shading Language(GLSL):
  - É uma linguagem de alto nível e por ser parecida com C/C++ o desenvolvimento é “fácil”.

Vertex Shader

```
#version 130

in vec3 Position;

void main()

{

gl_Position = vec4(Position.x,Position.y, Position.z, 1.0);

}
```

Fragment Shader

```
#version 130

out vec4 FragColor;

void main()

{

FragColor = vec4(1.0,0.0, 0.0, 1.0);

}
```

Como usar o shader?

- Mostrar o código 0

Sintaxe básica de GLSL

- Tipos:
  - int,uint,bool,float, bool
- Vectors
  - vec2,vec3,vec4(float) ,ivec2,uvec2,bvec2...

Sintaxe básica de GLSL

- vec4 a = vec4(1.0,2.0,3.0,4.0);
- vec4 b = a.xyzw;
- vec4 b = a.rgba;
- vec4 b = a.stpq;
- vec4 b = a.rrbb; //(1.0,1.0,3.0,3.0)
- ver3 b = a.xyz;

Sintaxe básica de GLSL

- Matriz:
  - mat2,mat3,mat4
  - m[1][3] contém o segundo elemento da 4 linhas
  - Os operadores estão sobrecarregados para permitir operações de matrizes.

Sintaxe básica de GLSL

- Estruturas de controle:
  - for,while, do-while, if-else
  - discard: descarta um fragmento.

Comunicação com a OpenGL

- Comunicação feita através das variáveis in/out e uniform.
- A diferença entre variável in e uniform é que variáveis in contêm dados que é específico do vértice e são recarregados com um novo valor para cada vértice. Enquanto o valor das variáveis uniform permanece constante.

Exemplo

- Mostrar o código 1 e 2

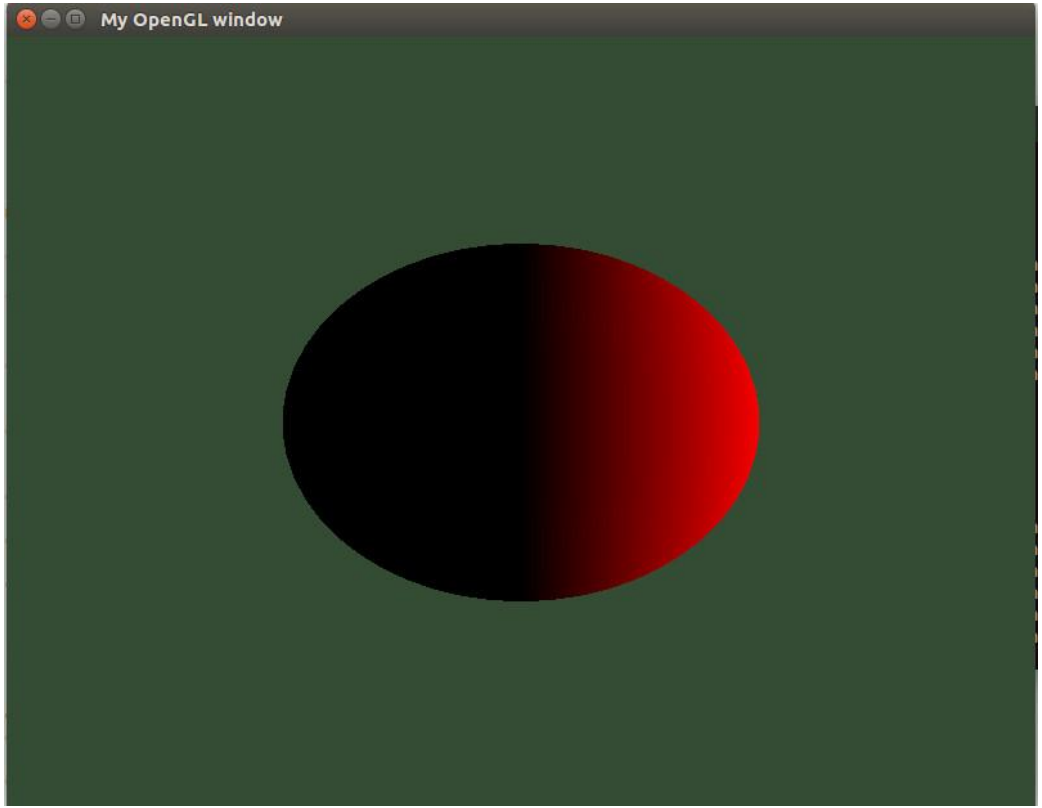
Como construir as transformações?

- Pyrr: Provides 3D mathematical functions using the power of NumPy.
- PyGLM: OpenGL Mathematics (GLM) library for Python
- etc.

Como fazer uma visualização perspectiva?

- Nada mais que uma nova matriz …
- Mostrar o código 4

Como fazer iluminação?



Referências Bibliográficas

- [1] OpenGL.org, https://www.opengl.org/
- [2] Celes, W. Notas de Aula. PUC-Rio, 2006.
- [3] Toth,Attila. Modern OpenGL programming in python: https://github.com/totex/PyOpenGL_tutorials