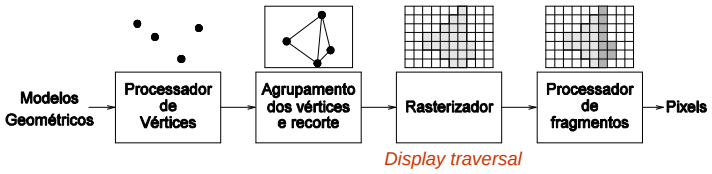
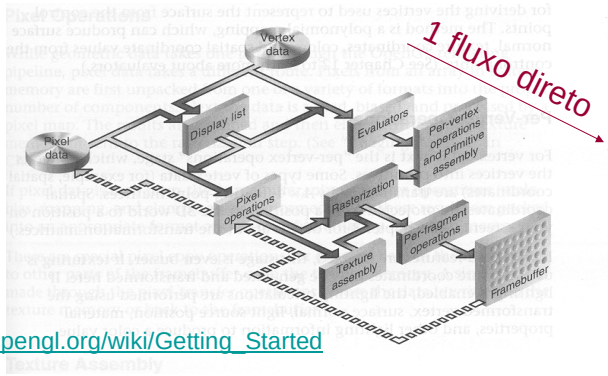
Programação de GPU

Fluxo de Imageamento: OpenGL 1.0



Display traversal



http://www.opengl.org/wiki/Getting_Started

Exemplo

```
int main(int argc, char **argv) {
        glutInit(&argc, argv);
        //glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
        glutInitWindowPosition(100,100);
        glutInitWindowSize(320,320);
        glutCreateWindow("GPU");
        glutDisplayFunc(renderScene);
        glutIdleFunc(renderScene);
        glutReshapeFunc(changeSize);
        glutKeyboardFunc(processNormalKeys);
        glutMainLoop();
        // just for compatibiliy purposes
         return 0;
}
```

Rotina *renderScene*

```
void renderScene(void) {
        //Set the clear color (black)
        glClearColor(0.0,0.0,0.0,1.0);
        //Clear the color buffer
        glClear(GL_COLOR_BUFFER_BIT);
        //stretch to screen
        glViewport(0,0,gw,gh);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0,gw,0,gh);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        //Draw a single line
        glVertex2i(gw/2, gh);
        glEnd();
        //swap buffers (double buffering)
        glutSwapBuffers();
}
```

Rotina *changeSize*

```
//GLUT callback fx
// called when window size changes
void changeSize(int w, int h) {
        //stores the width and height gw = w; gh = h;
        //Set the viewport (fills the entire window)
        glViewport(0,0,gw,gh);
        //Go to projection mode and load the identity
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        //Orthographic projection, stretched to fit the screen dimensions
        gluOrtho2D(0,gw,0,gh);
        //Go back to modelview and load the identity
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}
```

Rotina *changeSize*

```
//GLUT callback fx
// called when window size changes
void changeSize(int w, int h) {
        //stores the width and height gw = w; gh = h;
        //Set the viewport (fills the entire window)
        glViewport(0,0,gw,gh);
        //Go to projection mode and load the identity
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        //Orthographic projection, stretched to fit the screen dimensions
        gluOrtho2D(0,gw,0,gh);
        //Go back to modelview and load the identity
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}
```

Operações com *Framebuffer*

- Teste de Alpha: discardar fragmentos transparentes ($\alpha = 0$).
- Teste de Estêncil: aplicar a máscara contida no *buffer* de estêncil sobre fragmentos.
- Teste de Profundidade: comparar o valor de profundidade do fragmento com o contido no *buffer* de profundidade. *Early z-test!!!*
- Mesclagem por Alpha: composição de cores pelo alpha.

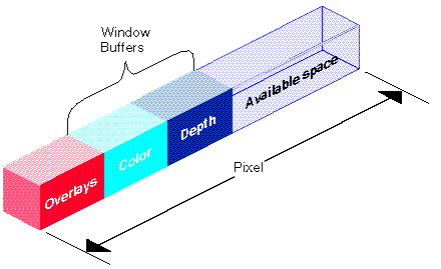
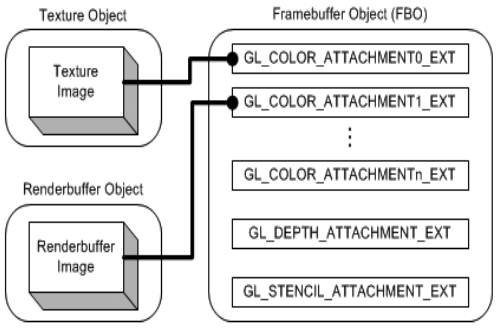


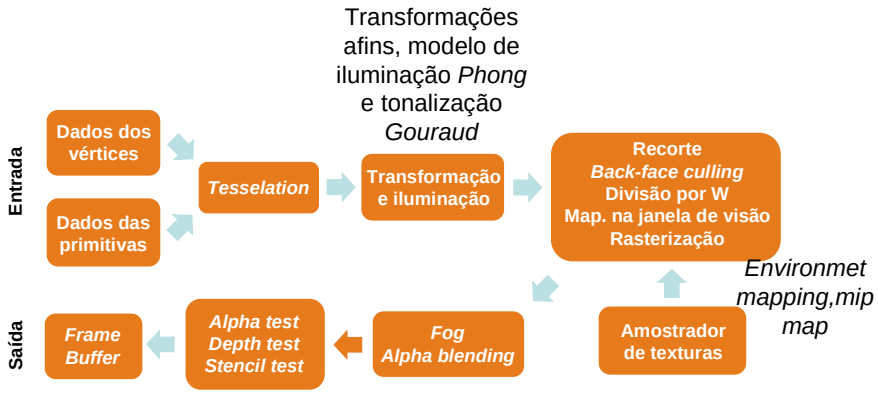
Figure 4. Frame Buffer Pixel Memory

Framebuffer Object (FBO)

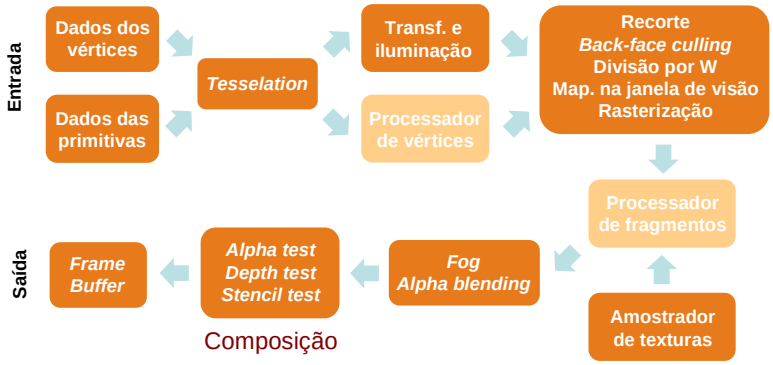
- Uma extensão de OpenGL (2006).
- Um conjunto de ponteiros aos objetos da memória (imagens não exibíveis).
- Imagens → *renderbuffer* object (*offscreen rendering*) e *texture object* (*render to texture*).
- Aplicações:
  - pós-processamento
  - composição



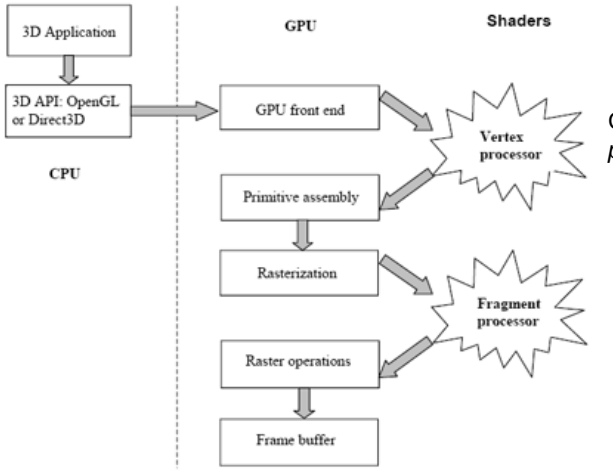
Fluxo de Dados em GPU



Fluxo de Dados em GPU Programável (OpenGL 2.0)



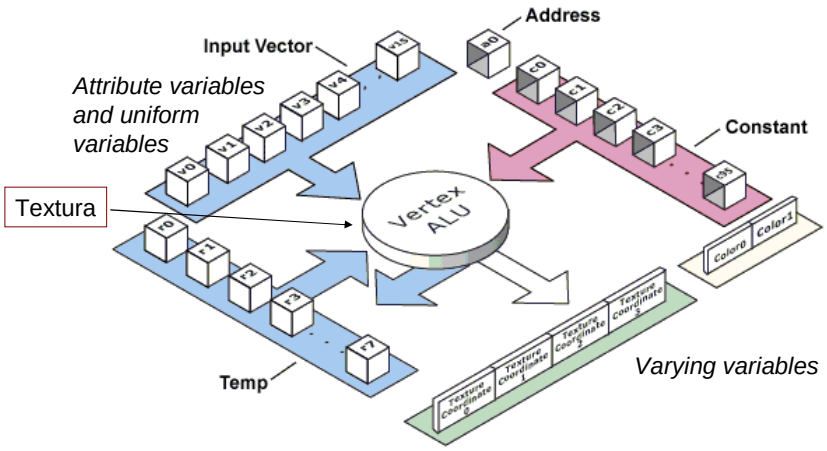
GPU Programável



Operações *per vértice*

Operações *per fragmento*

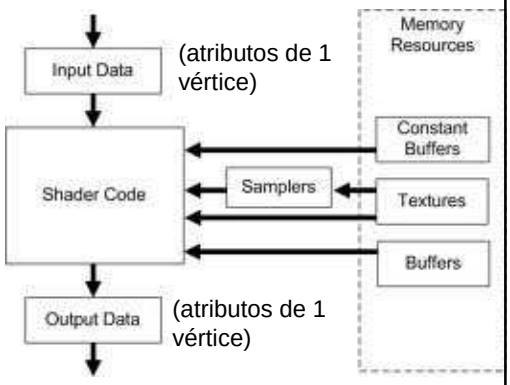
Vertex Shader
Organização



Registrador : 32 *bits*

Vertex Shader
Modelo de Programação

- Programa executado para cada vértice
- Entrada:
  - Um vértice (normalmente no espaço do objeto)
  - Pode conter informações tais como: normal, cor, coordenadas de textura, tamanho do ponto, etc
- Procedimento:
  - Busca de instrução
  - Busca de operandos
- Saída:
  - Vértice em espaço homogêneo de recorte

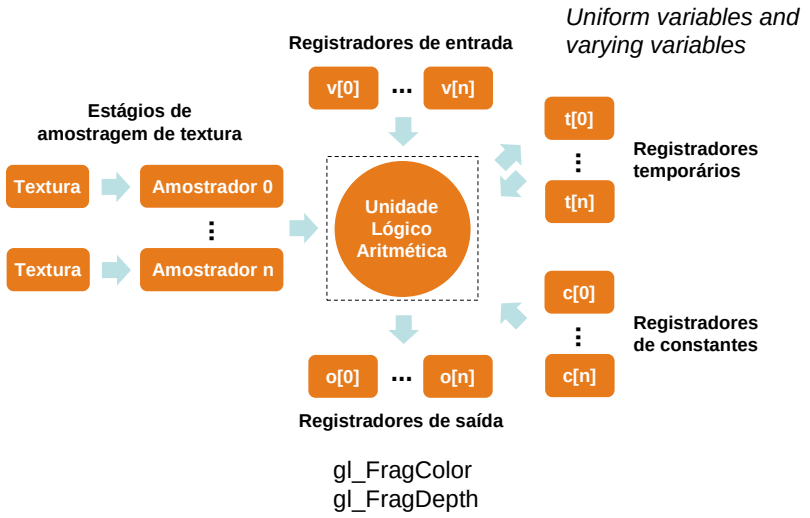


1 instrução/4 valores de 32 *bits (xyzw/RGBA)*

Vertex Shader
Funções

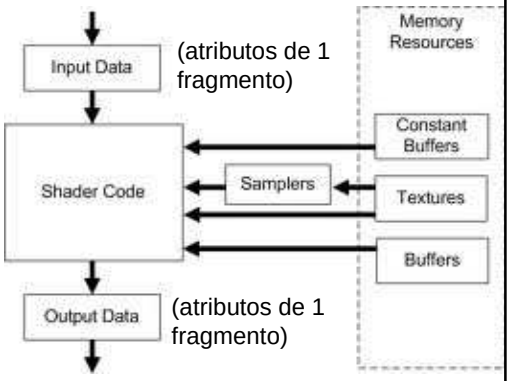
- O que um *vertex shader* pode fazer?
  - Transformação de vértices (mudança de base, rotação, translação, etc)
  - Normalização de vetores
  - Cálculo de iluminação por vértice
  - Geração e modificação de coordenadas de textura
- O que um *vertex shader* não pode fazer?
  - Montagem de primitivas
  - *Frustum culling*, *back-face culling*
  - Divisão por w (perspectiva)
  - Mapeamento na janela de visão

Fragment Shader
Organização



Fragment Shader
Modelo de Programação

- Programa executado para cada fragmento
- Entrada:
  - Um fragmento interpolado segundo os modelos *flat* ou *smooth*
  - Inclui as informações (também interpoladas) da saída do *vertex shader*
- Procedimento:
  - Busca de instrução
  - Busca e operandos
- Saída:
  - Cor (RGBA)
  - Profundidade



Fragment Shader
Funções

- O que um *fragment shader* pode fazer?
  - Amostragem e combinação de texturas
  - *Bump mapping*, *environment mapping*, cálculo de *fog* e uma infinidade de outras operações envolvendo cores
- O que um *fragment shader* não pode fazer?
  - Mudar o modelo de interpolação
  - Realizar teste alfa, teste de profundidade, teste do buffer estêncil
  - Habilitar ou desabilitar o modo de *dithering*

Shading Language

- A linguagem de programação utilizada para escrever um *shader* é chamada *shading language* – termo proveniente do *RenderMan*, da *Pixar (RSL ou SL)*
- 2002: *Architecture Review Board* (ARB) GPU *assembly language* → repertório de instruções em *assembly*
- 2004: GLSL (*OpenGL Shading Language*, slang), incorporada na especificação de OpenGL 2.0 → estrutura similar a da linguagem C
  - http://nehe.gamedev.net/data/articles/article.asp?article=21
- 2006: HLSL (*Microsoft DirectX High Level Shader Language*)/Cg (nVidia C for Graphics) → estrutura similar a da linguagem C
  - http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=47

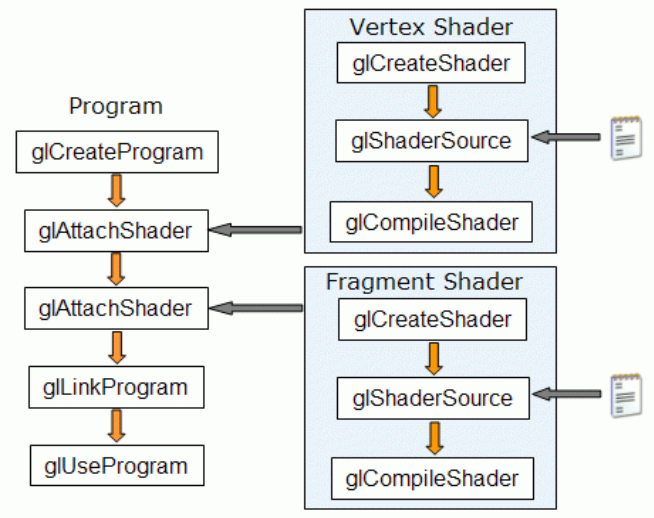
Exemplos

http://nehe.gamedev.net/article/glsl_an_introduction/25007/

http://www.informit.com/articles/article.aspx?p=171029&seqNum=2

http://www.clockworkcoders.com/oglsl/tutorial5.htm

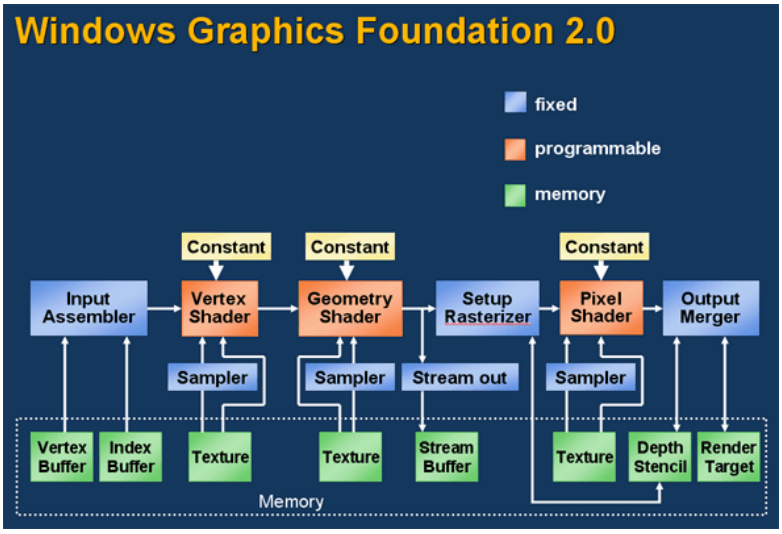
Programação em GLSL

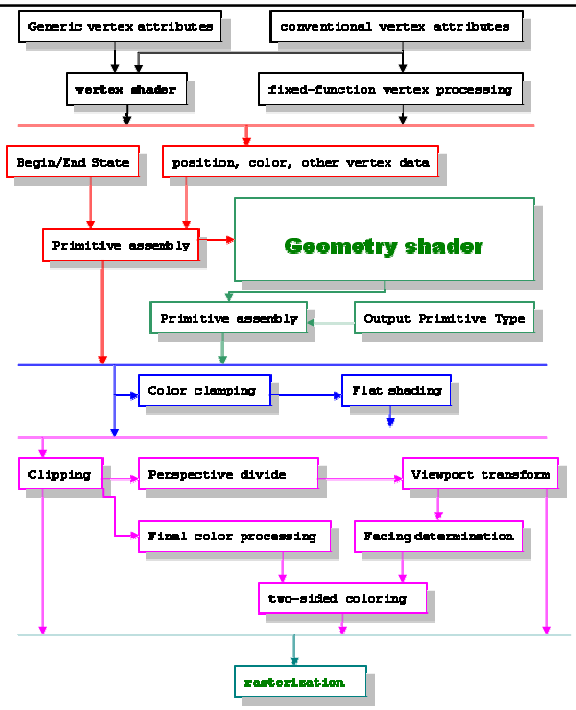


Exemplo

http://www.lighthouse3d.com/opengl/glsl/index.php?intro

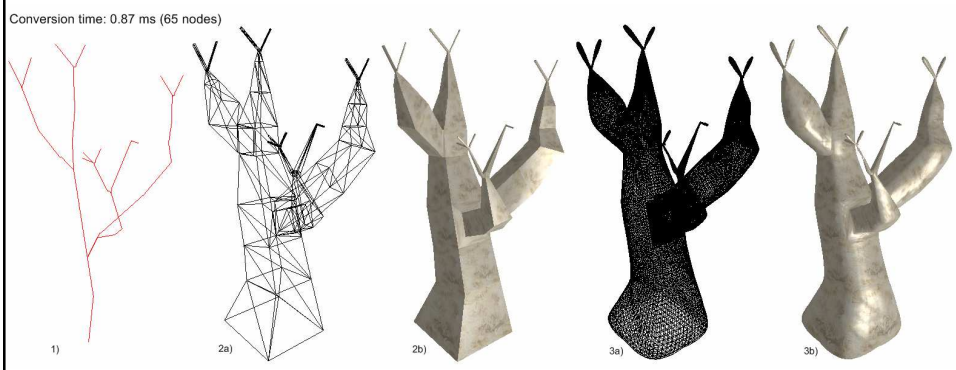
Geometry Shader (OpenGL 3.2)





Geometry Shader
Funções

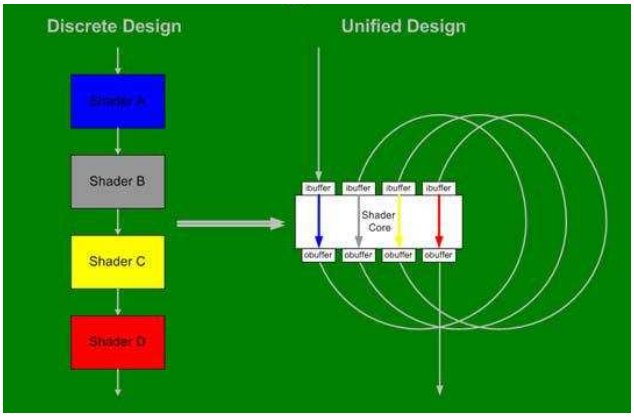
- O que um *geometry shader* pode fazer?
  - Pode derivar novas primitivas gráficas a partir dos dados processados pelo *vertex shader*.
  - Pode remover primitivas geradas pelo *vertex shader*.
  - Pode determinar comprimento de arestas.
  - Pode gerar equação dos planos
- O que um *geometry shader* não pode fazer?
  - Não pode alterar a conectividade das primitivas gráficas.
  - Não pode gerar primitivas “totalmente novas”.



Exemplo

http://www.opengl.org/wiki/Geometry_Shader_Examples

Unidades de *shaders* unificados (> 2006)



Um Aplicativo

http://cirl.missouri.edu/gpu/glsl_lessons/glsl_geometry_shader/index.html