OpenGL Shading Language

Guilherme S. Moura (gsm@cin.ufpe.br)
Saulo A. Pessoa (sap@cin.ufpe.br)
Vladimir Alves (vap2@cin.ufpe.br)

Felipe Maia

O que é?

- High Level Shading Language para OpenGL
- Descreve shaders
  - Programa que define as propriedades de um vértice (vertex) ou pixel (fragment)
    - Vertex -> Posição
    - Fragment -> Cor

Por que usar

- Versatilidade - funcionalidade programável permite maior expressividade criativa
  - Efeitos de última geração são possíveis em tempo real por GPUs programáveis
- Poder de processamento – mais pode ser feito na GPU
  - Libera a CPU para outras tarefas importantes
  - GPU faz operações em paralelo

Por que usar

- Tudo o que pode ser feito pela funcionalidade fixa de OpenGL pode ser realizado por shaders, e muito mais!
  - Inclusive cálculos não-gráficos
- GLSL é multi-plataforma - através dos vendedores de hardware e através dos sistemas operacionais

Vertex shaders

- Programas que operam sob vértices e seus dados associados;
- Intencionado a realizar:
  - Vertex transformation
  - Normal transformation and normalization
  - Texture coordinate generation
  - Texture coordinate transformation
  - Lighting
  - Color material application

Fragment shaders

- Programas que operam sob fragmentos (pixels) e seus dados associados;
- Intencionado a realizar:
  - Operações em valores interpolados
  - Acesso de textura
  - Aplicação de textura
  - Fog
  - Soma de cores

Gramática GLSL

- Parece com C, mas com algumas importantes diferenças
  - Não possui ponteiros
  - Não possui operadores bit a bit
  - Tipos básicos adicionais, incluindo vetores, matrizes, e samplers. Não há doubles ou strings

Tipos

- Básicos
  - float, int, bool, sampler
- Vetores
  - vec2, vec3, vec4
  - ivec2, ivec3, ivec4
  - bvec2, bvec3, bvec4
- Matrizes
  - mat2, mat3, mat4

Algumas operações

- vec4(1.0, 2.0, 3.0, 4.0)
  - vec4.x => 1.0
  - vec4.xy => vec2(1.0, 2.0)
  - vec4.rgba => vec4(1.0, 2.0, 3.0, 4.0)
  - vec4.b => 3.0
  - vec4.xgb => erro!

Algumas operações

- vec4(1.0, 2.0, 3.0, 4.0)
  - vec4.x => 1.0
  - vec4.xy => vec2(1.0, 2.0)
  - vec4.rgba => vec4(1.0, 2.0, 3.0, 4.0)
  - vec4.b => 3.0
  - vec4.xgb => erro!
- Swizzling e Smearing
  - vec4.wzyx => vec4.abgr => (4.0, 3.0, 2.0, 1.0)
  - vec4.yyy => vec3(3.0, 3.0, 3.0)

Gramática GLSL

- Variáveis built-in (ex. posição e cor)
- Funções built-in (ex. sine, exponentiation, cross product, etc.)
- Fortemente tipada, a única conversão implícita é de int pra float;

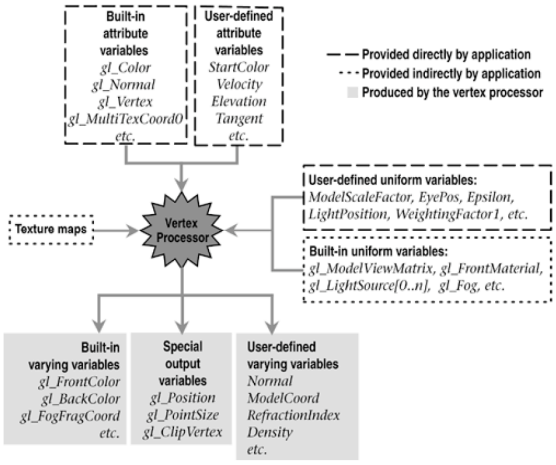
Qualificadores de tipo

- attribute
  - For frequently changing information, ***from the application to the vertex shader***
    - eg. position, normal etc.
    - `attribute vec4  gl_Vertex;`
    - `attribute vec3  gl_Normal;`
- uniform
  - For infrequently changing information, ***from the application to the vertex or fragment shader***
    - eg. light position, texture unit, other constants
    - `uniform mat4 gl_ModelViewMatrix;`
    - `uniform mat3 gl_NormalMatrix`

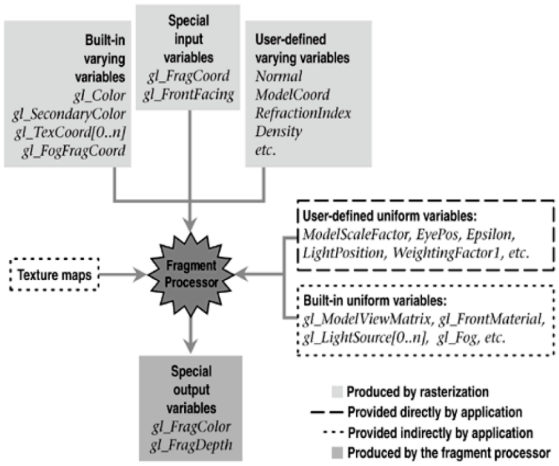
Qualificadores de tipo

- varying
  - For interpolated information passed ***from a vertex shader to a fragment shader***
    - eg. texture coordinates, vertex color
    - `varying vec4 gl_Color;  // fragment`
    - `varying vec4 gl_TexCoord[];`
- const
  - To declare nonwritable, compile time constant variables

Vertex Program



Fragment Program



Example: Vertex Shader

```
varying vec4 diffuseColor;
varying vec3 fragNormal;
varying vec3 lightVector;

uniform vec3 eyeSpaceLightPosition;

void main(){

   vec3 eyeSpaceVertex= vec3(gl_ModelViewMatrix * gl_Vertex);
   lightVector= vec3(normalize(eyeSpaceLightPosition - eyeSpaceVertex));
   fragNormal = normalize(gl_NormalMatrix * gl_Normal);

   diffuseColor = gl_Color;
   gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

Example: Fragment Shader

```
varying vec4 diffuseColor;
varying vec3 lightVector;
varying vec3 fragNormal;

void main(){

   float perFragmentLighting=max(dot(lightVector,fragNormal),0.0);

   gl_FragColor = diffuseColor * perFragmentLighting;
}
```


Exercício 1

- Crie um projeto novo no RenderMonkey
- Adicione o efeito padrão “Position” ao workspace (AddDefaultEffect/OpenGL/Position)
- Modifique o modelo 3D para “teapot.3ds”
- Abra o Fragment Shader e modifique a cor de saída para vermelha

Exercício 2

- Abra o projeto Exercicio1.rfx no RenderMonkey
- Abra o Vertex Shader e passe o valor da normal de cada vértice para o Fragment Shader
- Exiba o valor da normal recebido no Fragment Shader como cor
- Altere o Vertex Shader para passar valor da normal em coordenadas de vista

Exercício 3

- Crie um novo projeto no RenderMonkey
- Adicione o efeito padrão “Textured” ao Workspace
- Modifique a textura “base” para o arquivo “DayEarth.jpg”
- Adicione uma textura nova ao efeito e escolha o arquivo “NightEarth.jpg”
- Modifique a origem de ambas as textura para “Bottom Left”
- Adicione um “Texture Object” ao “Pass 0” que referencie a textura nova
- Também adicione uma variável do tipo “Float” ao “Pass 0”
- Modifique o Fragment Shader para que o resultado exibido seja uma mistura entre as texturas “NightEarth.jpg” e “DayEarth.jpg” de acordo com o valor da variável (zero exibe noite e um exibe dia).

Exercício 4

- Implemente o modelo de iluminação de Phong
- Criar variáveis para passar os coeficientes difuso, especular e ambiental do objeto
- Criar variáveis para passar a posição e a cor de uma fonte de luz onidirecional

Posso usar GLSL?

- Atualmente, quase todas as GPUs oferecem suporte
- GLEW
  - OpenGL Extension Wrangler Library
  - *http://prdownloads.sourceforge.net/glew/glew-1.3.2-src.zip?download*
  - *http://glew.sourceforge.net/*
  - *glewinfo.exe*
  - *Irá lhe fornecer as funcionalidades da sua placa*

Instalação e Inicialização

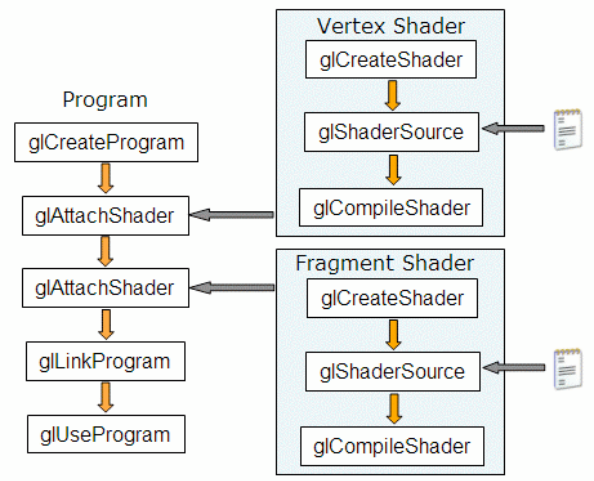
- glew32.dll, glew32.lib, glew.h, wglew.h
- Execute glewInit(); após glutInit();

```
GLenum err = glewInit();
    if (GLEW_OK != err)
    {
    …
    }
```

- Também pode checar extensões

```
if (GLEW_ARB_vertex_shader && GLEW_ARB_fragment_shader)
        printf("Ready for GLSL\n");
else {
        printf("No GLSL support\n");
        exit(1);
}
```


Setup



Duvidas?

- Referencias
  - KESSENICH, John M. The OpenGL Shading Language. Language Version: 1.20.8, http://www.opengl.org/documentation/glsl/
  - ROST, R. J. OpenGL Shading Language. Addison-Wesley Professional, 2004. ISBN 0321197895.
  - Keith O'Conor. An Introduction to the OpenGL Shading Language.