Aula 8 - Shaders

Dennis Kerr Coelho

Sequencia

- O que é Shader
- Histórico
- Vertex Shader
- Fragment(Pixel) Shader
- Outros Shaders
- Render Pipeline

O que é Shader

- Shader (Sombreador) é um programa de computador utilizado para sombrear, assim produzir a melhor cor para uma imagem. Numa visão moderna é utilizado para adicionar efeitos de pós processamento a uma imagem.
- Shaders programam efeitos de renderização com uma grande grau de flexibilidade. A maioria dos shader são executados no hardware gráfico.

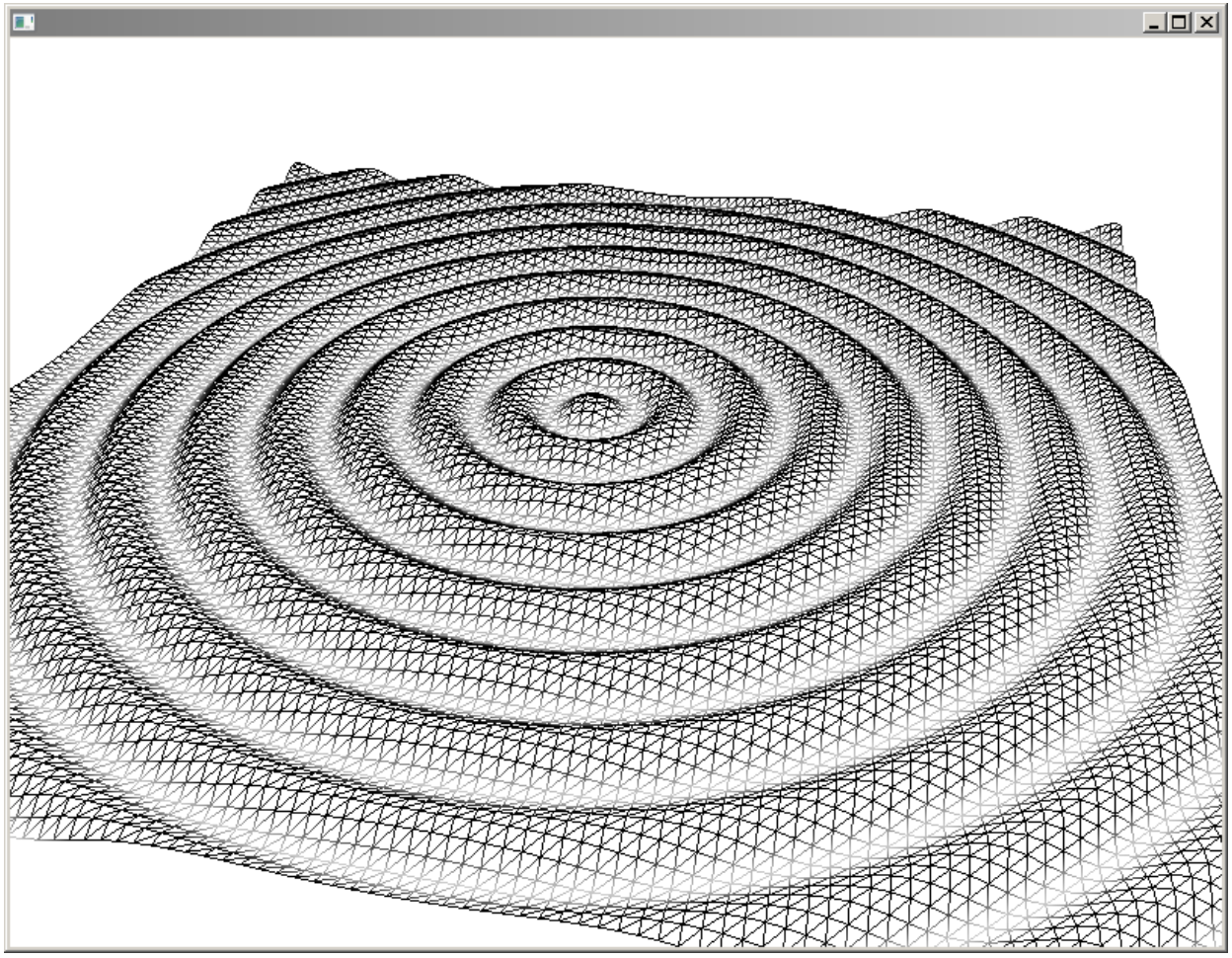
Histórico

- O uso moderno de shader foi introduzido ao publico pela pixar na Specificação da Inteface RenderMan Versão 3.0 publicada em 1988
- A primeira versão de OpenGL a suportar shader foi a versão 1.5 de 2003 que suportava shaders através de extensões.
- A partir da versão 2.0 de Setembro de 2007 o suporte a shader passou a ser nativa do OpenGL.
- No inicio somente era suportado o pixel(fragment) shader, mas logo em seguida veio o suporte a vertex shader.
- A partir da versão 3.2 de agosto de 2009 foi adicionado o suporte a geometry shader.

Vertex Shader

- O Vertex Shader é o estágio Shader programável no pipeline de renderização que manipula o processamento de vértices individuais. Vertex shaders são alimentados pelos dados dos vertices, a partir de um objeto de matriz vértice, por um comando de desenho. Um vértice shader recebe um único vértice do fluxo de vértice e gera um único vértice para o fluxo de saída de vértice. Deve haver um mapeamento 1: 1 de vértices de entrada para saída vértices.

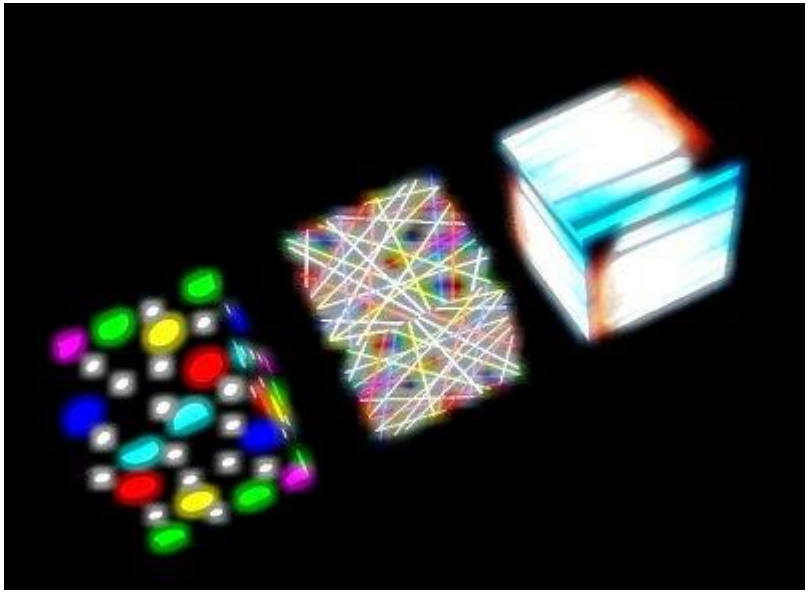
Vertex Shader

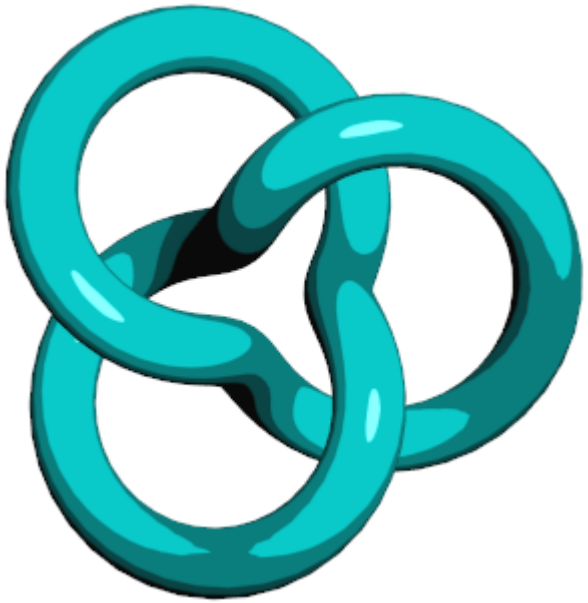


Fragment(Pixel) Shader

- Um Fragment Shader é o estágio Shader que irá processar um fragmento gerado pelo Rasterization em um conjunto de cores e um únic o valor de profundidade.
- O shader de fragmento é o estágio do pipeline OpenGL depois que uma primitiva é rasterizada. Para cada uma das amostras dos pixels cobertos por uma primitiva, um "fragmento" é gerado. Cada fragmento tem uma posição de espaço de janela, alguns outros valores, e contém todos os valores de saída interpolada por vértice da última etapa de processamento Vertex Shader.
- A saída de um shader de fragmento é um valor de profundidade, um valor stencil possível (não modificada pelo shader de fragmento), e zero ou mais valores de cor a ser potencialmente escrito para os buffers nos framebuffers atuais.
- shaders de fragmentos tomam um único fragmento como entrada e produzir um único fragmento como saída.

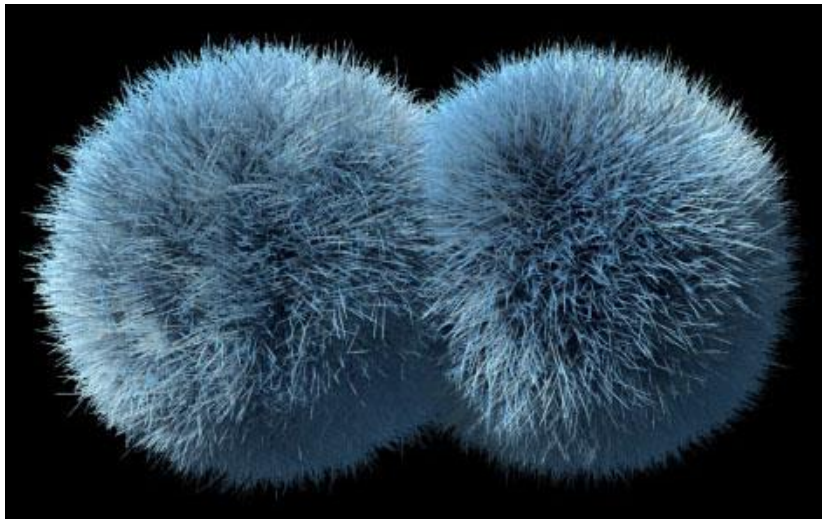
Fragment(Pixel) Shader

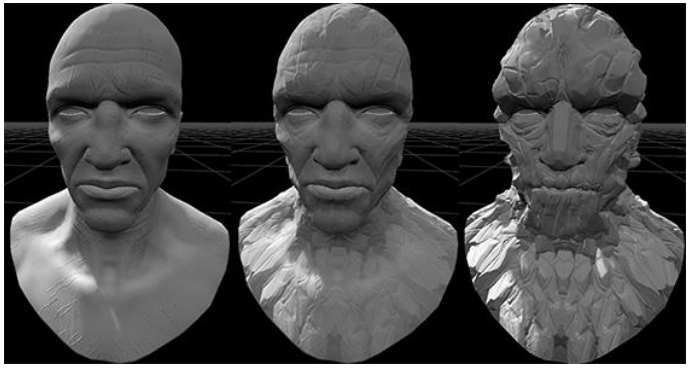




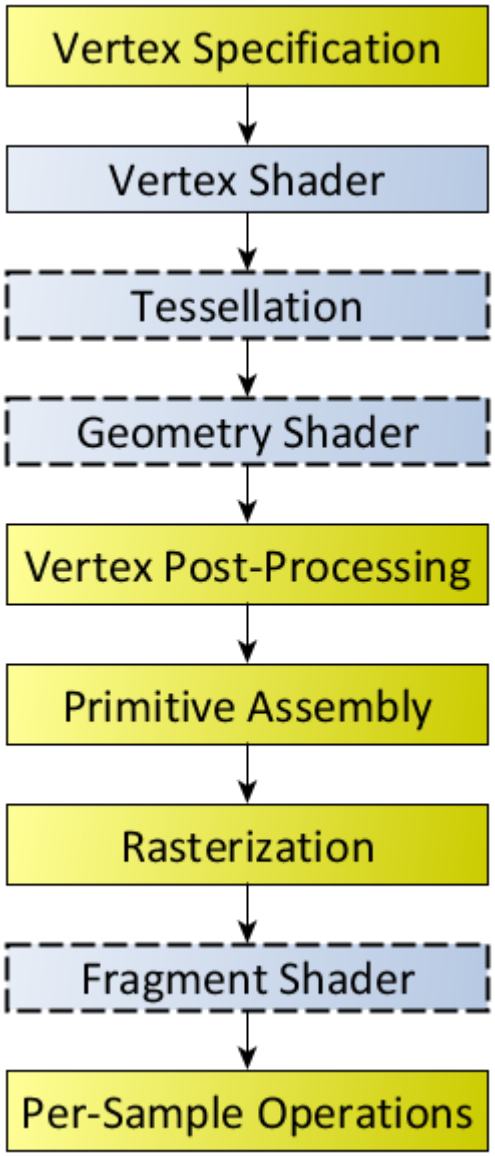
Outros Shaders

- Geometry Shader
- Tesselation





Render Pipeline



OpenGL Shader Language (GLSL)

- Apresentada primeiramente como uma extensão ARB (Architecture Review Board 2003).
- Tornou-se parte integrante da biblioteca OpenGL a partir da versão 2.0.
- Muito semelhante a linguagem C.
- Apesar de possuir sintaxe semelhante, possui diferenças e restrições consideráveis em relação à linguagem C.
- Tais restrições se devem ao tipo de tarefa a que foi destinada e a limitação dos dispositivos gráficos.

*www2.ic.uff.br/~anselmo/cursos/CGI/slidesGrad/CG_aula4(introducaoa**OpenGL**).pdf*

OpenGL Shader Language (GLSL)

- A arquitetura dos dispositivos mais recentes eliminou parte destas restrições juntamente com o surgimento de linguagens apropriadas para tais arquiteturas.
- Exemplos são as arquiteturas/plataformas propostas pela NVidia CUDA – Compute Unified Device Architecture) e ATI-AMD (CAL – Compute Abstraction Layer)
- Nestas arquiteturas é possível desenvolver aplicações de propósito geral sem as idiossincrasias das linguagens de tonalização.
- Entretanto tanto CUDA quanto CAL adotam um modelo de programação completamente distinto do utilizado convencionalmente.

*www2.ic.uff.br/~anselmo/cursos/CGI/slidesGrad/CG_aula4(introducaoa**OpenGL**).pdf*

GLSL – Tipos de Dados

- A GLSL suporta tipos de dados vetoriais já que foi criada para geração de efeitos de tonalização e cálculo de illuminação.
- A vantagem em se utilizar tipos vetoriais é a de que o hardware é capaz de realizar operações sobre eles com muito mais eficiência do que sobre tipos escalares.

*www2.ic.uff.br/~anselmo/cursos/CGI/slidesGrad/CG_aula4(introducaoa**OpenGL**).pdf*

Tipos de Dados

Tipos fundamentais:

- void - Tipo nulo empregado em funções que não retornam valores
- vec2, vec3, vec4 - Vetores float de 2, 3 e 4 componentes
- mat2, mat3, mat4 - Matrizes float 2x2, 3x3 e 4x4
- bool, int, float - Tipos escalares comuns
- sampler1D, sampler2D, sampler3D - Utilizados para acessar texturas 1D, 2D, 3D e cubemaps
- sampler1Dshadow, sampler2Dshadow - Utilizados para acessar texturas 1D e 2D contendo valores de profundidade para cálculo de sombras.

*www2.ic.uff.br/~anselmo/cursos/CGI/slidesGrad/CG_aula4(introducaoa**OpenGL**).pdf*

Tipos de Dados

- Os tipos mais comuns são baseados em valores reais.
- Por outro lado, existem variações como inteiros (ivec2, ivec3, ivec4) e booleano (bvec2, bvec3 e bvec4).
- O usuário pode criar vetores de qualquer tamanho como em C com a restrição de que eles sejam unidimensionais.

*www2.ic.uff.br/~anselmo/cursos/CGI/slidesGrad/CG_aula4(introducaoa**OpenGL**).pdf*

Tipos de Dados

- Não existe tipo char nem strings.
- Não existem ponteiros.
- Não é possível alocar vetores dinâmicamente.

*www2.ic.uff.br/~anselmo/cursos/CGI/slidesGrad/CG_aula4(introducaoa**OpenGL**).pdf*

Tipos de Dados

- A linguagem suporta estruturas pré-definidas ou definidas pelo usuário
- Exemplo:

```
struct gl_LightSourceParameters {
vec4 ambient; // ambiente
vec4 diffuse; // difuso
vec4 specular; // especular
vec4 position; // posicao
vec4   halfVector; // vetor médio
vec3 spotDirection; // direção se for spot
float spotExponent; // expoente se for spot
float spotCutoff; // ângulo de cutoff se for spot
float spotCosCutoff; // cosseno do ângulo de cutoff
float constantAttenuation; // fator de atenuação constante
float linearAttenuation; // fator de atenuação linear
float quadraticAttenuation;// fator de atenuação quadrático
};
uniform gl_LightSourceParameters gl_LightSource[gl_MaxLights];
```

*www2.ic.uff.br/~anselmo/cursos/CGI/slidesGrad/CG_aula4(introducaoa**OpenGL**).pdf*

Tipos de Dados

Em GLSL cada variável possui um qualificador associado que indica como a variável deve ser interpretada:

- nenhum - Variável local ou parâmetro de entrada para função
- const - Constante ou parâmetro de função somente de leitura
- attribute - Define uma conexão entre o programa de vértice e a OpenGL para valores que variam por vértice
- uniform - Define um valor que não é alterado durante o desenho de uma primitiva
- Varying - Define uma ligação entre a saída de um program de vértices e o resultado interpolado que serve de entrada para um programa de fragmentos
- in - Parâmetro de entrada para uma função
- out - Parâmetro de saída não inicializado
- Inout - Parâmetro de entrada e saída

*www2.ic.uff.br/~anselmo/cursos/CGI/slidesGrad/CG_aula4(introducaoa**OpenGL**).pdf*

Tipos de Dados

- Variáveis globais (definidas fora de uma função) podem ter qualificadores const, attribute, uniform ou varying.
- Variáveis locais só podem utilizar o quantificador const
- Parâmetros para funções podem usar in, out ou inout.

*www2.ic.uff.br/~anselmo/cursos/CGI/slidesGrad/CG_aula4(introducaoa**OpenGL**).pdf*

Qualificador attribute

- É empregado somente em programas de vértices.
- Serve para que este possa receber valores que variam a cada vértice.
- Tais valores são enviados diretamente do programa do usuário ou providos pela biblioteca.

Exemplos:

```
attribute vec4 gl_Color;
attribute vec3 gl_Normal;
attribute vec4 gl_Vertex;
```

*www2.ic.uff.br/~anselmo/cursos/CGI/slidesGrad/CG_aula4(introducaoa**OpenGL**).pdf*

Qualificador uniform

- Empregado tanto nos programas de vértice quanto nos programas de fragmento.
- Representam variáveis recebidas do programa do usuário ou internas à biblioteca cujo conteúdo não varia durante o desenho de uma primitiva.
- Por exemplo: não é possível realizar transformações geométricas entre um par glBegin ... glEnd e por esse motivo, a variável gl_ModelviewMatrix (que contém a matriz de visualização) é definida como uniform.

```
uniform mat3 gl_ModelviewMatrix;
```

*www2.ic.uff.br/~anselmo/cursos/CGI/slidesGrad/CG_aula4(introducaoa**OpenGL**).pdf*

Qualificador varying

- Tem como função criar uma ligação entre uma variável no programa de vértice e a mesma no programa de fragmento.
- Variáveis declaradas como varying têm seu conteúdo interpolado pela etapa de conversão matricial.
- Desta forma, o programa de fragmento recebe o valor específico para o fragmento em questão e não a saída do programa de vértice.

*www2.ic.uff.br/~anselmo/cursos/CGI/slidesGrad/CG_aula4(introducaoa**OpenGL**).pdf*

Programar