

# Utilização de Shaders para criação de efeitos realistas para jogos



**Bruno Pereira Evangelista**

Pontifícia Universidade Católica de Minas Gerais



# Sumário

---

- **Apresentação pessoal**
- **Introdução**
- **Pipeline de renderização**
- **Efeitos utilizando GPUs**
- **Shaders, Como utilizar shaders**
- **Ferramentas**
- **Demos de shaders**
- **Mercado**
- **Outros usos da GPU**

# Apresentação

---

- 2002-2006: Bacharelado em Ciência da Computação (PUC Minas)
- 2004-2006: Ministrante de 1 curso e 3 minicursos sobre OpenGL
- 2005: Olympya (Desenvolvimento do MMO FutWeb)
- 2005: Pesquisas na área de não-fotorealismo
  - Renderização de cartoons (Sibgrapi 2005)
- 2006: Pesquisas na área de foto-realismo
  - Renderização de superfícies detalhadas

# Introdução

- A indústria de jogos tem buscado criar ambientes virtuais com alto nível de realismo
- Níveis de realismo tão altos que não possam ser diferenciados do mundo real



# Introdução

---

- Durante muito anos as APIs gráficas (DirectX, OpenGL) trabalharam utilizando um modelo de processamento fixo para renderização das cenas
  - Modelo de iluminação fixo
  - Impossibilidade de aplicação de vários efeitos foto-realistas (e não foto-realistas)
  - Jogos com gráficos semelhantes

# Introdução

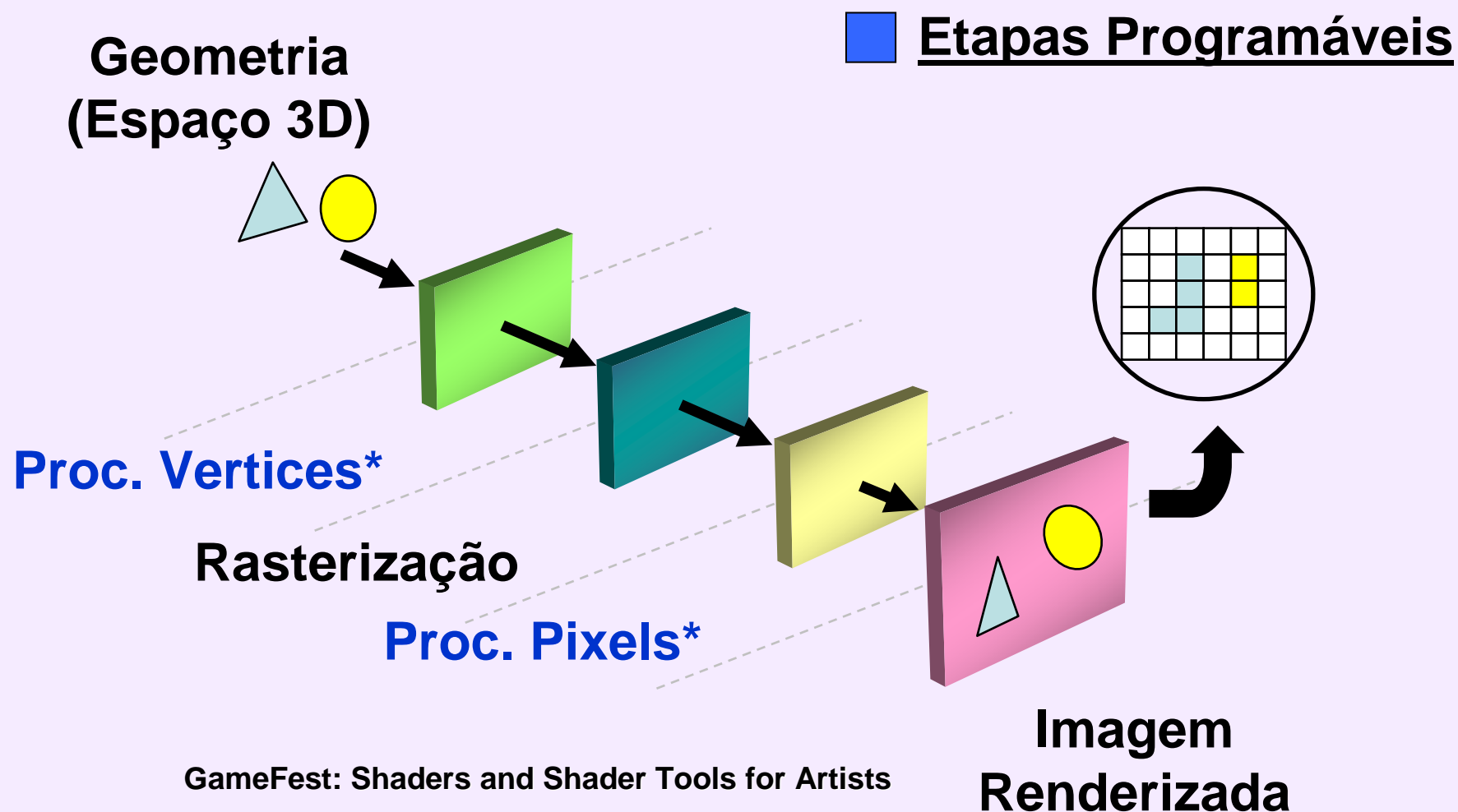
---

- **Devido a esses problemas surgiu a necessidade de alterar a forma como as cenas eram processadas**
- **Isso se tornou possível com a introdução dos hardwares gráficos programáveis (GPUs)**
- **Possibilita criar cenas foto-realistas, com efeitos nunca vistos antes**
- **Necessário utilizar APIs gráficas com suporte as mesmas (DirectX 8+, OpenGL 1.5+)**

# Pipeline de renderização

- Para entender o funcionamento das GPUs vamos dar uma olhada em um diagrama reduzido do pipeline de renderização das cenas
- O pipeline fixo e o programável são semelhantes em seus estágios
- A diferença é algumas etapas fixas passaram a ser programáveis

# Pipeline de renderização



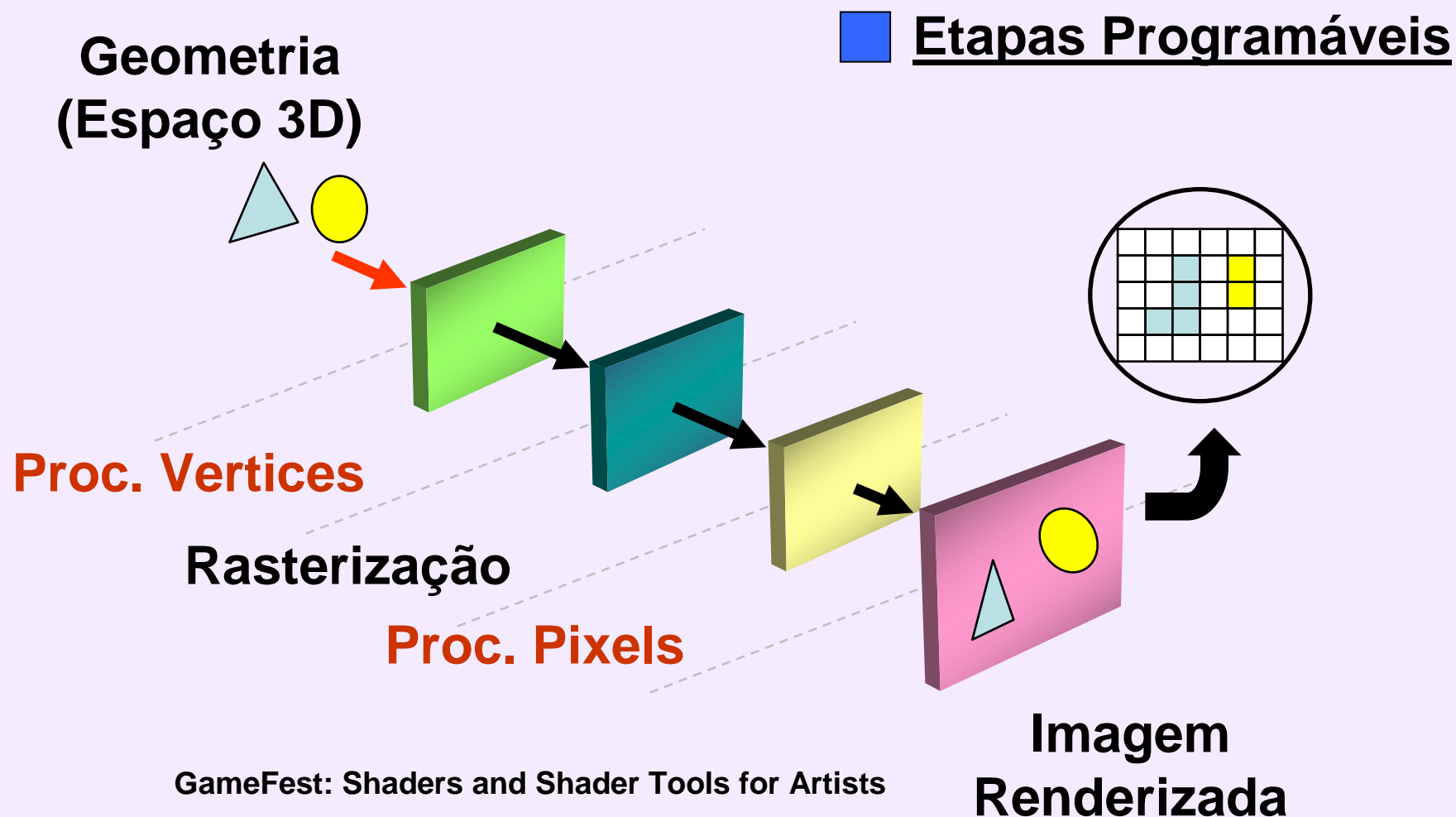
GameFest: Shaders and Shader Tools for Artists

# Geometria (Espaço 3D)

---

- **Informações necessárias para a construção das geometrias**
  - **Posição dos vértices ( $v_1$ ,  $v_2$ ,  $v_2$ )**
  - **Cor dos vértices**
  - **Outros**
- **Informações geralmente exportadas de softwares de modelagem (3D Studio Max, Maya, outros)**

# Pipeline de renderização

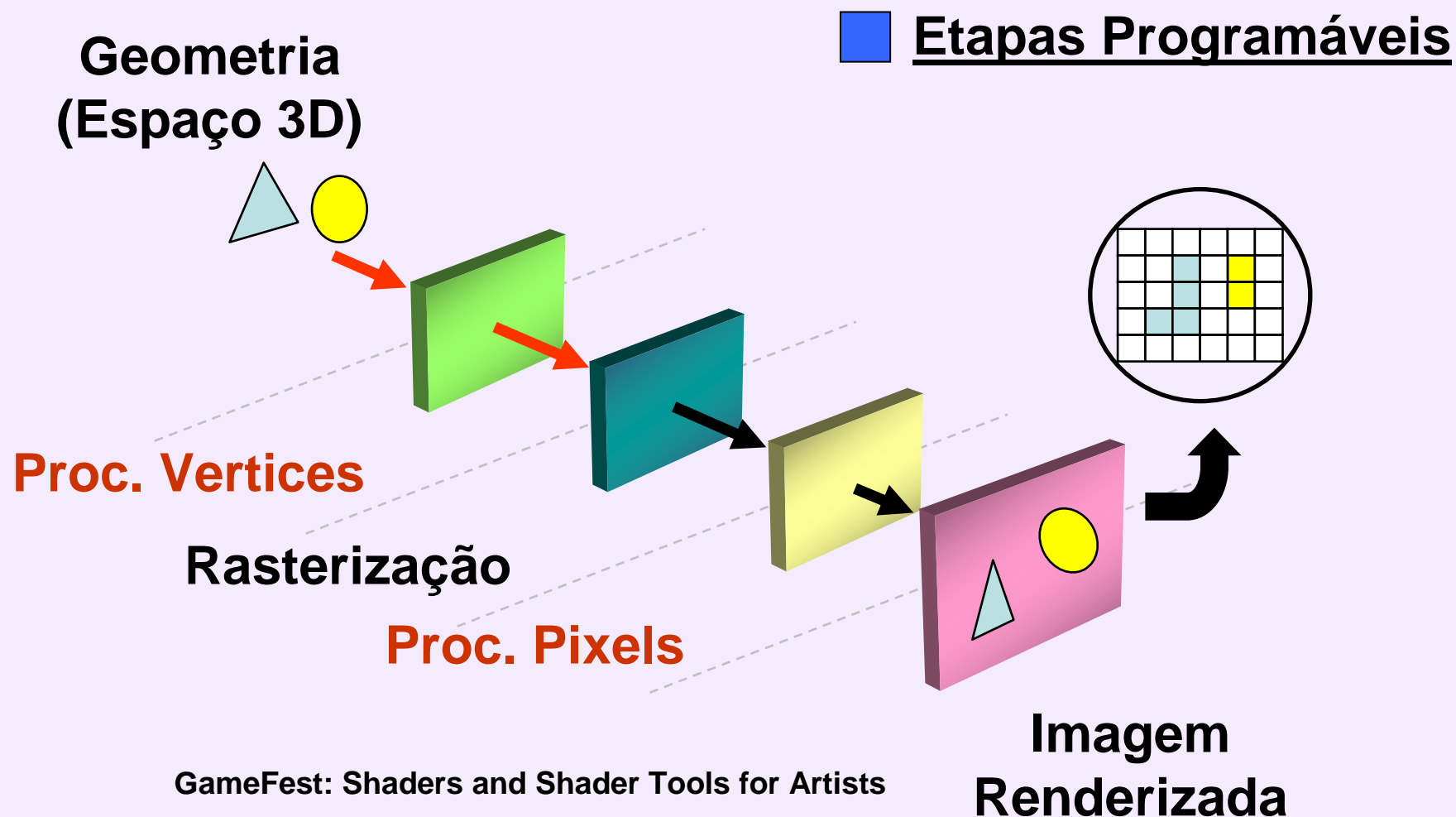


GameFest: Shaders and Shader Tools for Artists

# Processamento de vértices

- **Processa os vértices, e informações relativas aos vértices, para poderem ser exibidos na tela**
  - **Câmeras (Visão, Projeção)**
  - **Iluminação**
  - **Deformações nas malhas**
  - **Animações**
  - **Outros**

# Pipeline de renderização



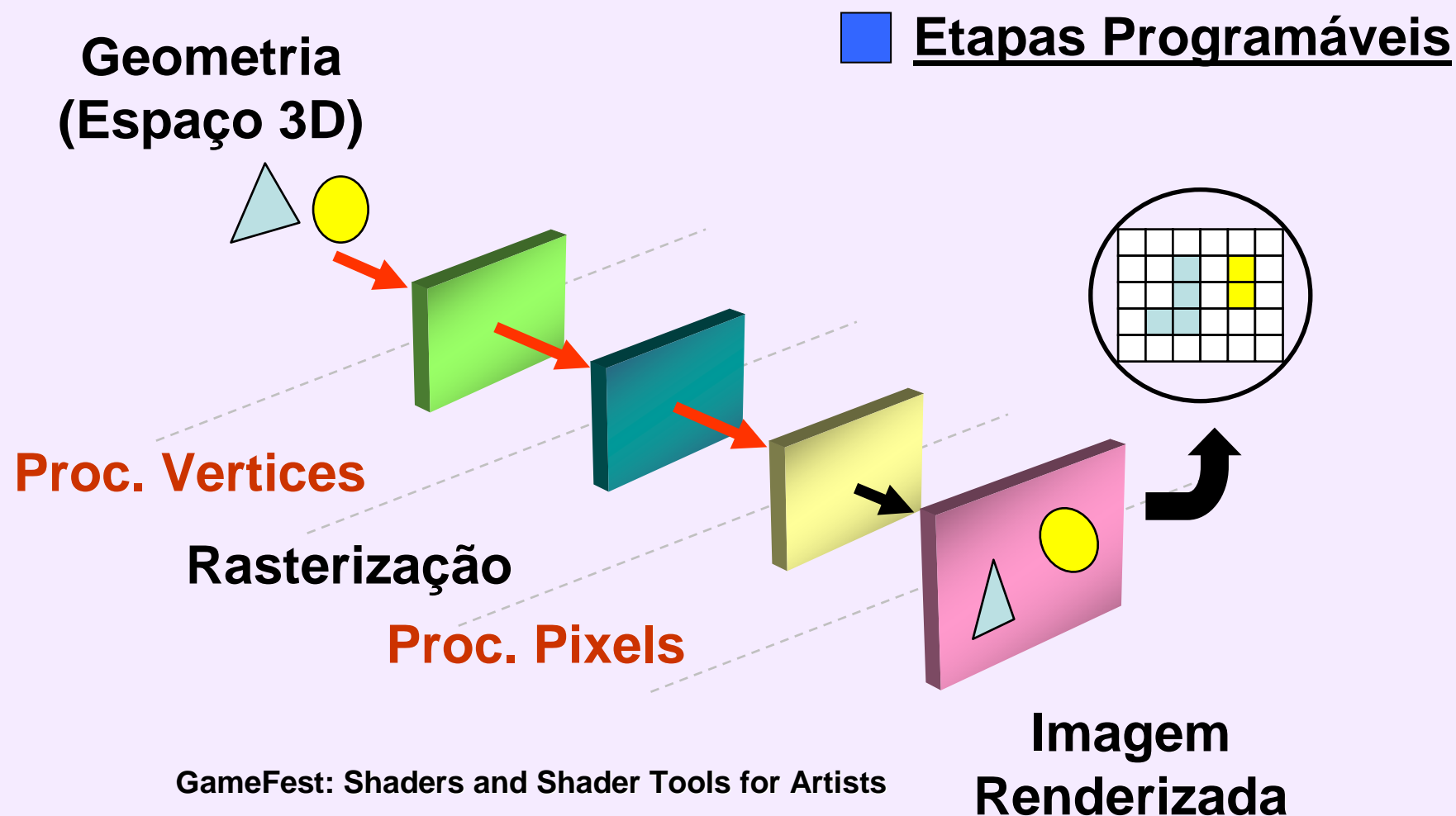
GameFest: Shaders and Shader Tools for Artists

# Rasterização

---

- Transforma as geometrias do espaço vetorial para pixels (Imagem 2D)
- Geração de uma imagem bidimensional contendo a cena
- Necessário para ser exibido para o usuário

# Pipeline de renderização



GameFest: Shaders and Shader Tools for Artists

# Processamento de pixels

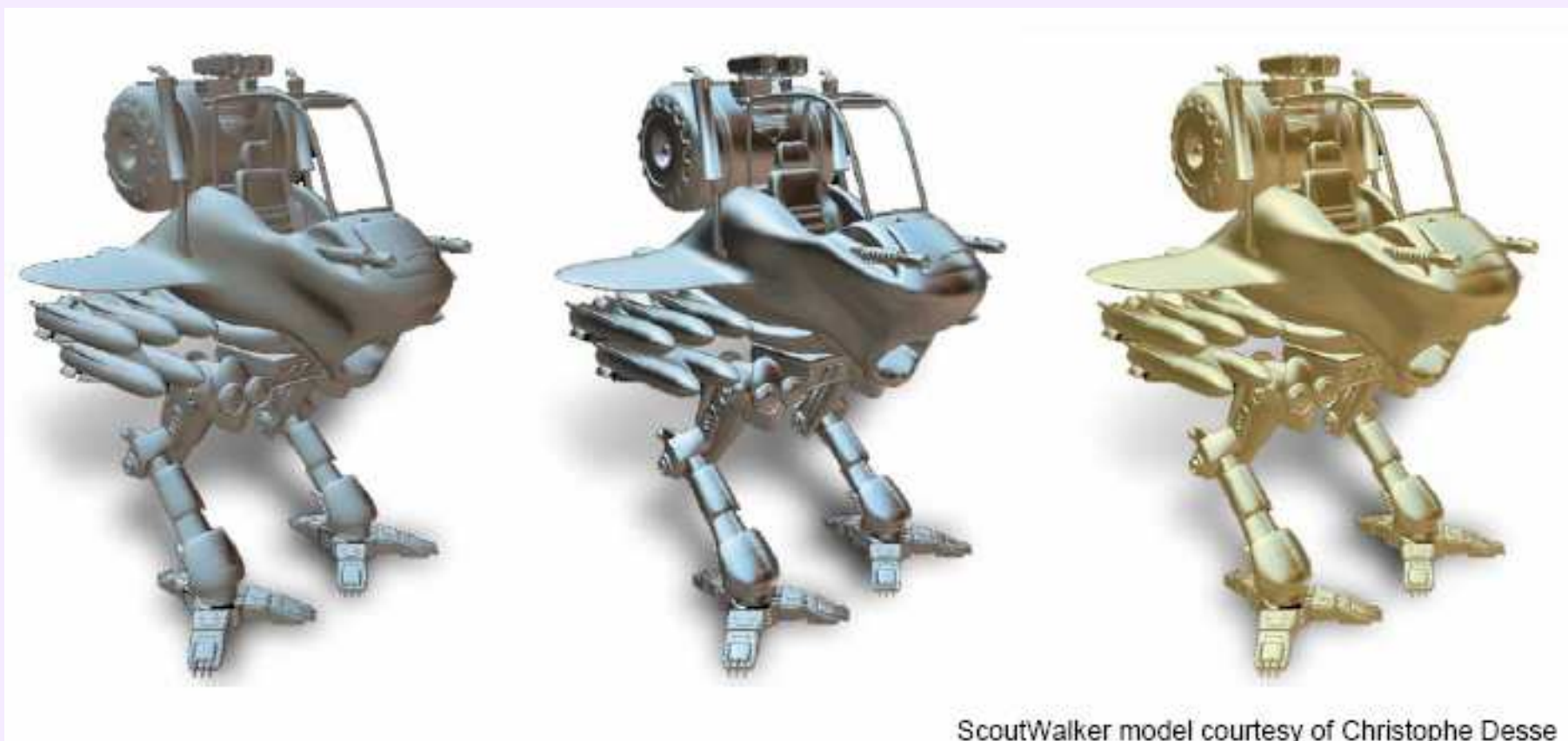
---

- **Processa a imagem 2D gerada**
- **Filtros para remoção de serrilhamentos**
- **Adição de efeitos (Blur, Glow, ...)**
- **Outros**

**Video**

**Quest 3D – 2006  
Azureus Temple**

# Novos efeitos utilizando GPU



ScoutWalker model courtesy of Christophe Desse

## Modelos de luzes reais

# Novos efeitos utilizando GPU



**Cube mapping**  
**Reflexão e refração por pixel**

# Novos efeitos utilizando GPU



Unreal Engine

## High Dynamic Range

# Novos efeitos utilizando GPU



Relief Mapping (Policarpo 2006)

## Simulação de detalhes em superfícies

# Novos efeitos utilizando GPU



Hugo Beyer



ATI – The Matrix

## Renderização de peles e cabelos humanos

# Novos efeitos utilizando GPU



## Renderização de Cartoons

# Shaders

---

- O que são shaders?
- Shaders são pequenos programas que são executados nas GPUs
- Utilizados para definir o processamento dos vértices e pixels da cena
- Na nova versão do HLSL (Shader Model 4) também será possível definir o processamento das geometrias

# Shaders

---

- **Linguagens para programação de shaders**
  - **Cg (nVidia), GLSL (OpenGL), HLSL (Microsoft), Sh, outras**
  - **Todas as linguagens possuem sintaxe similar ao C**
- **Assim como todo programa os shaders também precisam ser compilados e linkados**
  - **Geralmente ocorre em tempo de execução**

# Shaders

---

- São independentes do programa executável
- Permite a troca dos shader sem necessidade de recompilação do código
- Pode-se criar shaders com diferentes recursos, para diferentes hardwares gráficos
- Pode-se trocar dinamicamente os shaders que são utilizados para renderização das cenas

# Vertex Shaders

---

- **Processamento aplicado a cada vértice da geometria**
- **Têm como entrada todos os atributos dos vértices**
  - **Posição, Normal, Cor, Coordenada de textura, outros**
  - **Além dessas informações o programa pode fornecer outros parâmetros ao shader**
- **Pode fornecer dados de entrada para o Pixel Shader**

# Pixel Shaders

---

- **Processamento aplica a cada *pixel* (ou fragmento) da tela**
- **Têm como entrada todos os atributos dos *pixels***
  - **Cor, Profundidade, Transparência, outros**
- **GPUs geralmente possuem um poder de processamento de *pixels* muito superior ao de vértices**

# Exemplo de código

... Informações de matrizes ...

```
float3 lightPos = {10.0f, 10.0f, -10.0f}; // Posição da luz
```

```
float3 objectColor = {1.0f, 0.0f, 0.0f}; // Cor do objeto
```

// Informações passadas para o vértice e pixel shader

```
struct a2v { float4 pos : POSITION; float3 normal : NORMAL; };
```

```
struct v2p { float4 hpos : POSITION; float3 color : COLOR0; };
```

// Vertex Shader

```
v2p VertexShader(a2v IN) {
```

```
    v2p OUT;
```

```
    OUT.hpos = mul(IN.pos, matWVP); // Posição do vértice na tela
```

```
    float3 n = normalize(mul(IN.normal, matW)); // Vetor normal
```

```
    float3 l = normalize(lightPos - mul(IN.pos, matW)); // Vetor luz
```

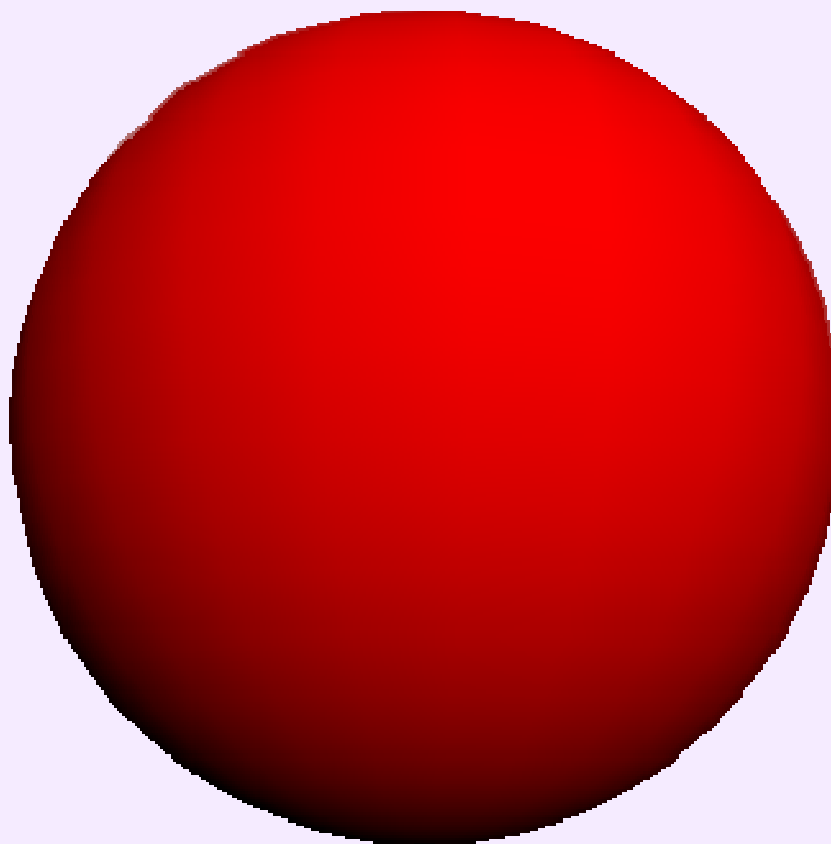
```
    OUT.color = objectColor * saturate(dot(n, l)); // Cor de saída
```

```
    return OUT;
```

```
}
```

# Resultado

---



**Iluminação por *pixel***

# Como utilizar um shader

---

- Alguns passos são necessários para utilização de um shader
- Criação de um programa para renderização utilizando uma API gráfica com suporte a shaders
- Carregar e compilar o código do shader
- Fornecer os parâmetros necessários para execução do shader
- Ativar o shader
- Desenhar a cena

# Ferramentas

---

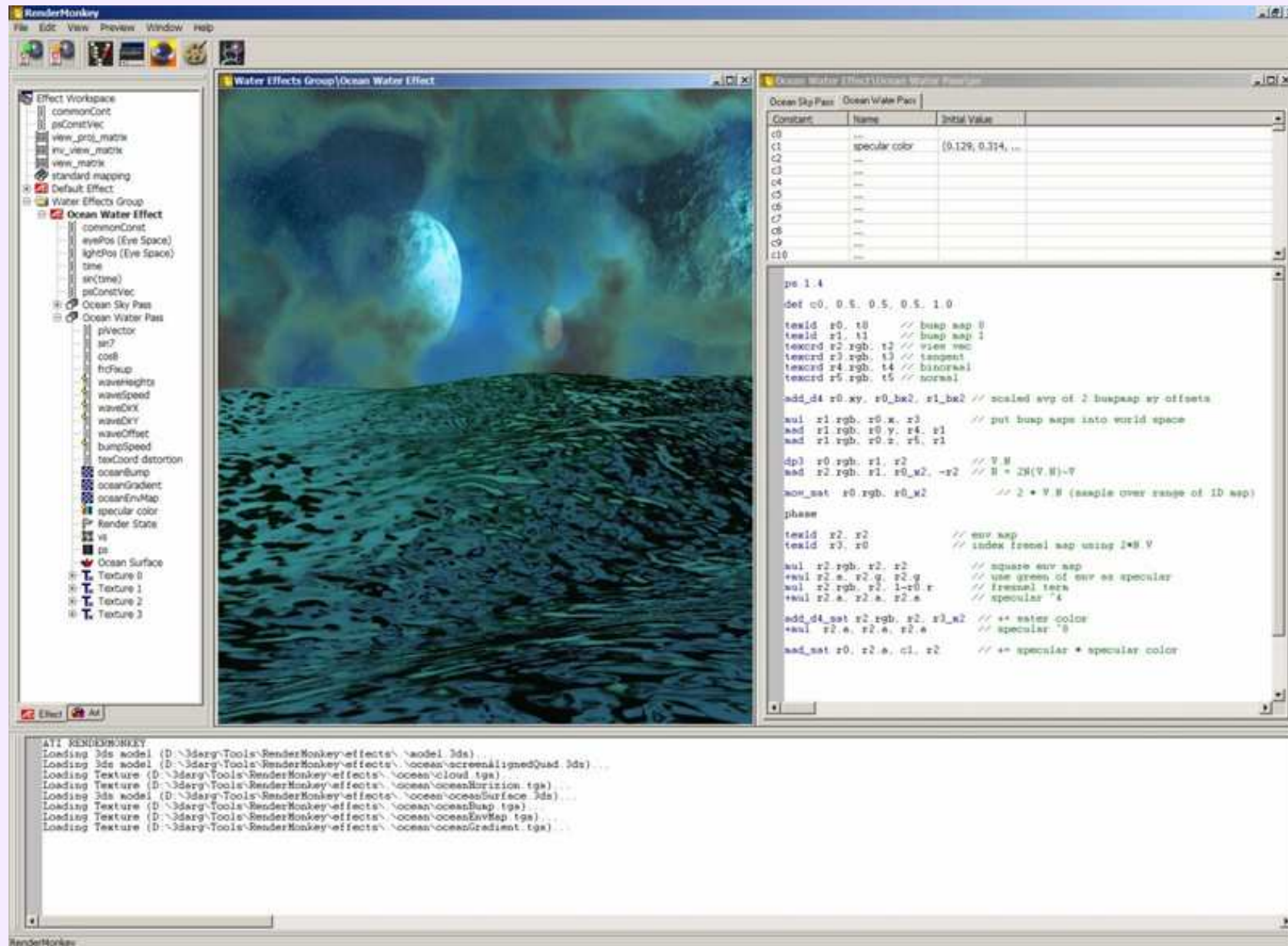
- **Otimizam o processo de desenvolvimento de shaders**
- **Permite que modificações no shader, e seus parâmetros, sejam observadas em tempo real**
- **Permite uma maior integração dos artistas no desenvolvimento dos shaders**

# Ferramentas

---

- **Existem diversas ferramentas para desenvolvimento de shaders, algumas populares são:**
  - **FX Composer (nVidia)**
  - **Render Monkey (ATI)**
  - **Shader Studio (Garage Games)**
  - **Outros**

# ATI Render Monkey



# FX Composer

The screenshot displays the NVIDIA FX Composer interface for a material named "Glow.fx". The central editor shows the following HLSL code:

```
OutCol += tex2D(GlowSamp2, IN.TexCoord2) * (WTS_0/WTS_NORMALIZE);
OutCol += tex2D(GlowSamp2, IN.TexCoord3) * (WTS_1/WTS_NORMALIZE);
OutCol += tex2D(GlowSamp2, IN.TexCoord4) * (WTS_2/WTS_NORMALIZE);
return Glowness*OutCol;
}

/////////

// just drawn model itself

float4 PS_Model(VS_OUTPUT IN) : COLOR
{
    float4 Col = IN.Diffuse * tex2D(ModelTexSamp, float2(IN.TexCoord));
    return Col;
}

// add glow on top of model

float4 PS_GlowPass(VS_OUTPUT IN) : COLOR
{
    float4 tex = tex2D(GlowSamp1, float2(IN.TexCoord));
    return tex;
}

/////////

technique Glow_9Tap
{
    pass GlowBuffer <
        string rendertarget = "GlowMap1";
        float cleardepth = 1.0f;
        dword clearcolor = 0x0;
    > {
```

The interface includes several panels:

- Materials:** A grid of material preview spheres.
- Textures:** A grid of texture preview images.
- Scene Graph:** A hierarchical tree view of the scene objects, including "Scene", "Root Transform", "Frame\_World", "Frame\_polySurface530", "Frame\_Disc", "GeoPipe : Disc", "Frame\_Face", "Frame\_Hair", "Frame\_L\_Eye", and "Frame\_L\_ForeArm".
- Properties:** A panel for the "Glow" material, showing parameters like "ModelTex", "GlowMap1", "GlowMap2", "Colors", "Parameters", "lightdir", "Source Texture Brightness", "Glow Strength" (set to 3), "WvpM", "WorldM", and "Texel Stride for Blur".
- Shader Perf:** A panel showing performance metrics for the "Glow\_9Tap" technique, including "Pixel Shader: 255.255 #Instructions: 8", "Scheduling", and "This shader has a general efficiency rating of 50%".
- Scene:** A 3D preview window showing a dragon-like creature with glowing blue eyes and wings, labeled "Frame: 0 Perspective".

The status bar at the bottom indicates "Ready" and "Ln 361 Col 19 CAP".

**Exemplo 1:**

**Shaders para iluminação:**

**Iluminação por vértice**

**Iluminação por pixel**

**FX Composer**

**Exemplo 2:**

**Cube Mapping:  
Reflexão e refração por pixel**

**FX Composer**

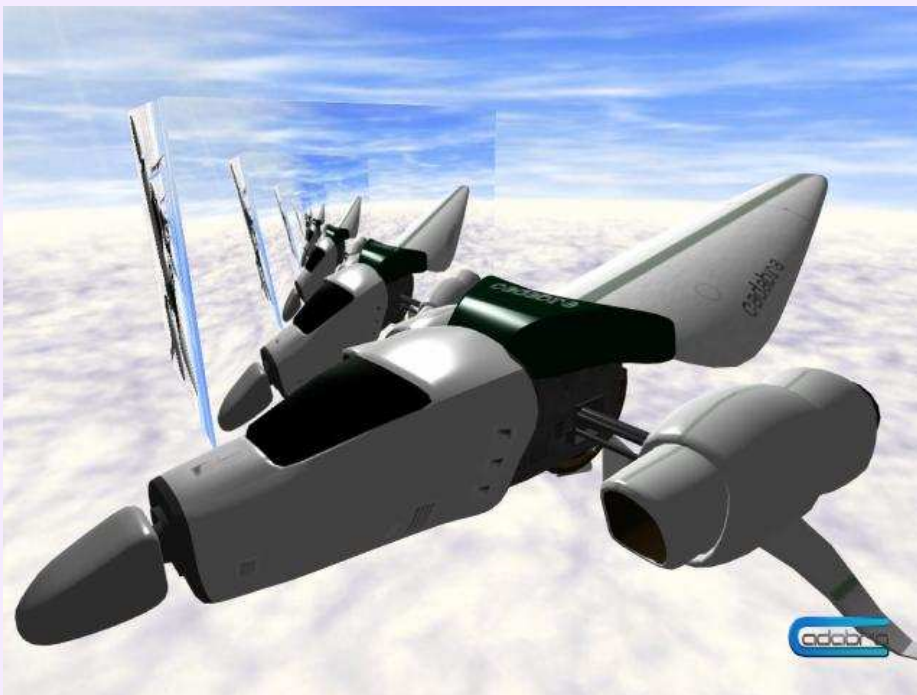
# Motores 3D

---

- **Vários motores gráficos possuem suporte a shaders**
- **Motores mais completos suportam várias linguagens de shaders (HLSL, Cg, GLSL)**
- **Motores podem vir com parses para fazer conversões entre linguagens de shaders (WildMagic 3 - Eberly)**
- **Alguns motores são distribuídos com vários shaders inclusos o que facilita a adição de efeitos visuais aos jogos**

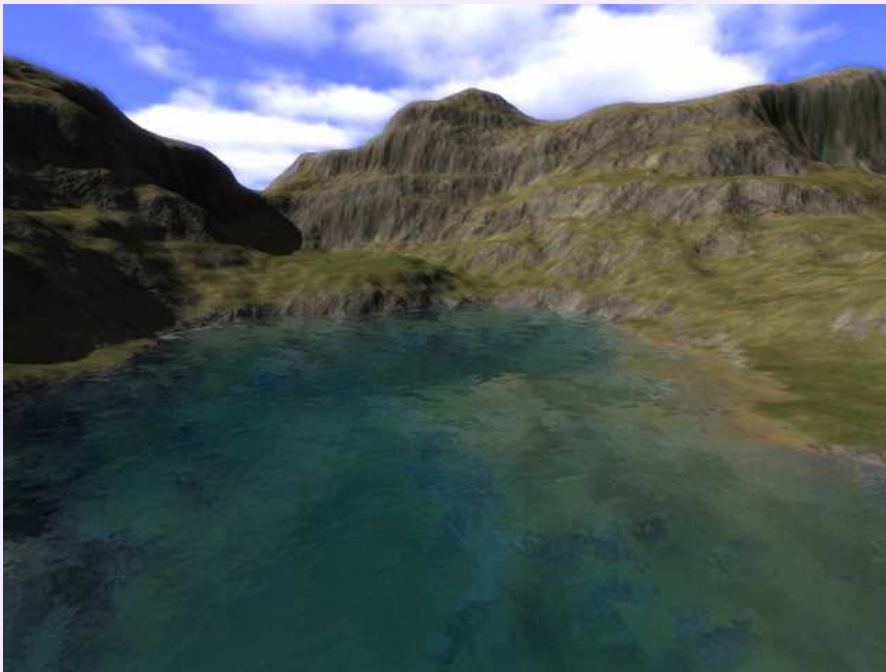
# Motores 3D

- Motores gratuitos com suporte a shaders
  - Cadabra
  - Ogre 3D



# Motores 3D

- Motores de baixo custo com suporte a shaders
- Torque Shader Engine
- C4 Engine



# Motores 3D

---

- **Motores de alto custo com suporte a shaders**
  - **Crytek**
  - **Doom 3**
  - **Unreal 3**
- **Esses motores fazem grande parte do processamento das cenas, utilizando shaders**
  - **Podendo inclusive realizar os cálculos de física na GPU**

# Mercado

---

- Grande parte do esforço no desenvolvimento dos jogos atuais está ligado a produção de shaders
- Jogos podem possuir várias versões do mesmo shader (funcionar em várias GPUs)
- No DirectX 10 não haverá mais pipeline fixo, tudo será feito por shaders
- Estão diretamente ligados ao realismo do jogo, alto investimento

# Mercado

---

- **No exterior já existem cargos específicos para criação de shaders**
  - **Área de pesquisa contínua**
  - **Trabalho com tecnologia de ponta**
- **Jogos brasileiros já estão utilizando shaders em grande parte dos efeitos**
  - **Taikodom**

# Outros usos para a GPU

---

- **Devido a seu alto poder de processamento, também é possível utilizar as GPUs para realizar cálculos de propósito geral (General Purpose GPU)**
  - **Cálculo de estrutura de proteínas**
  - **Resolução de equações algébricas e sistemas lineares**
  - **Algoritmos de ordenação**
  - **Outros**

# Outros usos para a GPU

---

- A utilização da GPU para realização desses cálculos de propósito geral ocorre devido ao seu alto poder de processamento
- As GPUs atuais possuem um poder de processamento, superior aos processadores

# Conclusão

---

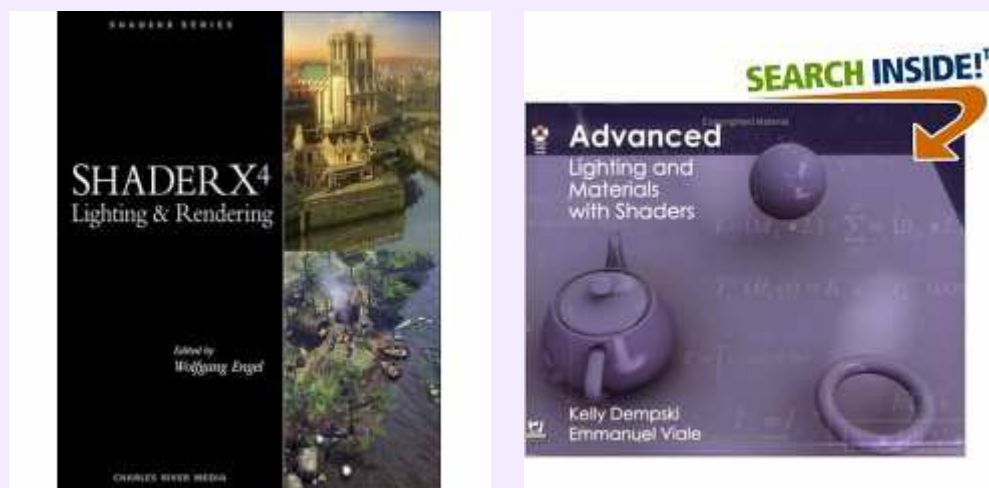
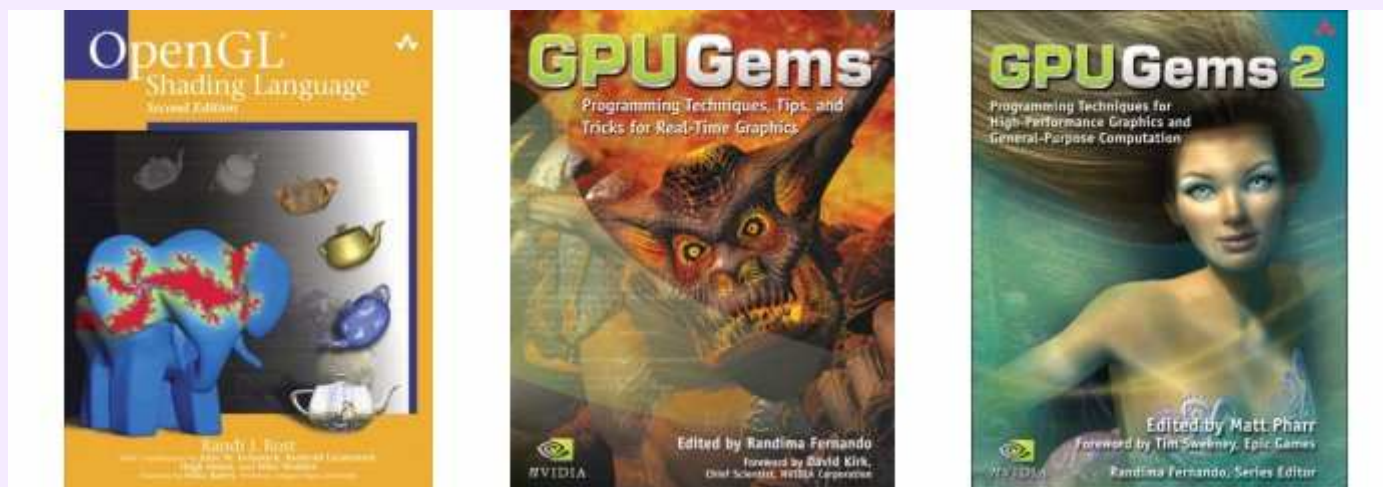
- **A programação com shaders ainda impõem algumas restrições**
  - **Tamanho do código do programa**
  - **Desvios dinâmicos (if, switch, while, ...)**
  - **Precisão nos cálculos (float8, float16)**
  - **Outros**
- **Essas restrições tem sido superadas com a evolução dos shaders**

# Conclusão

---

- **Versões mais recentes das APIs gráficas (DirectX e OpenGL) devem abandonar o pipeline fixo de renderização**
- **Possibilita otimizar melhor o pipeline programável**
- **A versão 10 do DirectX já não possui mais o pipeline padrão de renderização**
- **É necessário o uso de shaders para a criação de qualquer aplicativo**

# Livros



# Perguntas?

---

**Bruno Pereira Evangelista**  
[bpevangelista@gmail.com](mailto:bpevangelista@gmail.com)

**Página pessoal:**  
[www.brunoevangelista.com](http://www.brunoevangelista.com)

**"De fato, que aproveitará ao homem ganhar  
o mundo inteiro mas perder sua alma?"**

**Mateus 16, 26**