Desenvolvimento de shaders com CGFX

Computer Graphics and Game Technology

CGGT - 18/05/2007

Alessandro Ribeiro da Silva

Conteúdo

- Apresentação
- Introdução a shaders
- CGFX
- Projeto Peixis
- Processamento de Vertice
- Materiais
- Pos-processamento(PDI)
- Efeitos compostos
- GPGPU
- Considerações finais

Apresentação

- CGGT – Computer Graphics and Game Technology group
  - Voltado para estudos nas áreas:
    - Computação Gráfica
    - Jogos
  - Temas ja abordados neste semestre:
    - Mercado, Jogo completo, Modelagem, XNA

Apresentação

- Participe do grupo através de nossa lista de discussão

https://listas.dcc.ufmg.br/mailman/listinfo/cggt

- Web-site

https://www.dcc.ufmg.br/projetos/cggt

Introdução a shaders

- Shaders são programas que permitem a programação de GPUs
  - Permite implementação de efeitos em tempo-real

Introdução a shaders
História

- Shade Trees – Cook, Robert L. (SIGRAPH84)

http://graphics.pixar.com/ShadeTrees/paper.pdf

- Interface Renderman (1989)
  - Possui vários tipos de shaders diferentes
  - Implementação PRMan – Pixar

https://renderman.pixar.com/products/rispec/

- Stanford RealTime Shading Language (2002)

http://graphics.stanford.edu/projects/shading/

Introdução a shaders
História

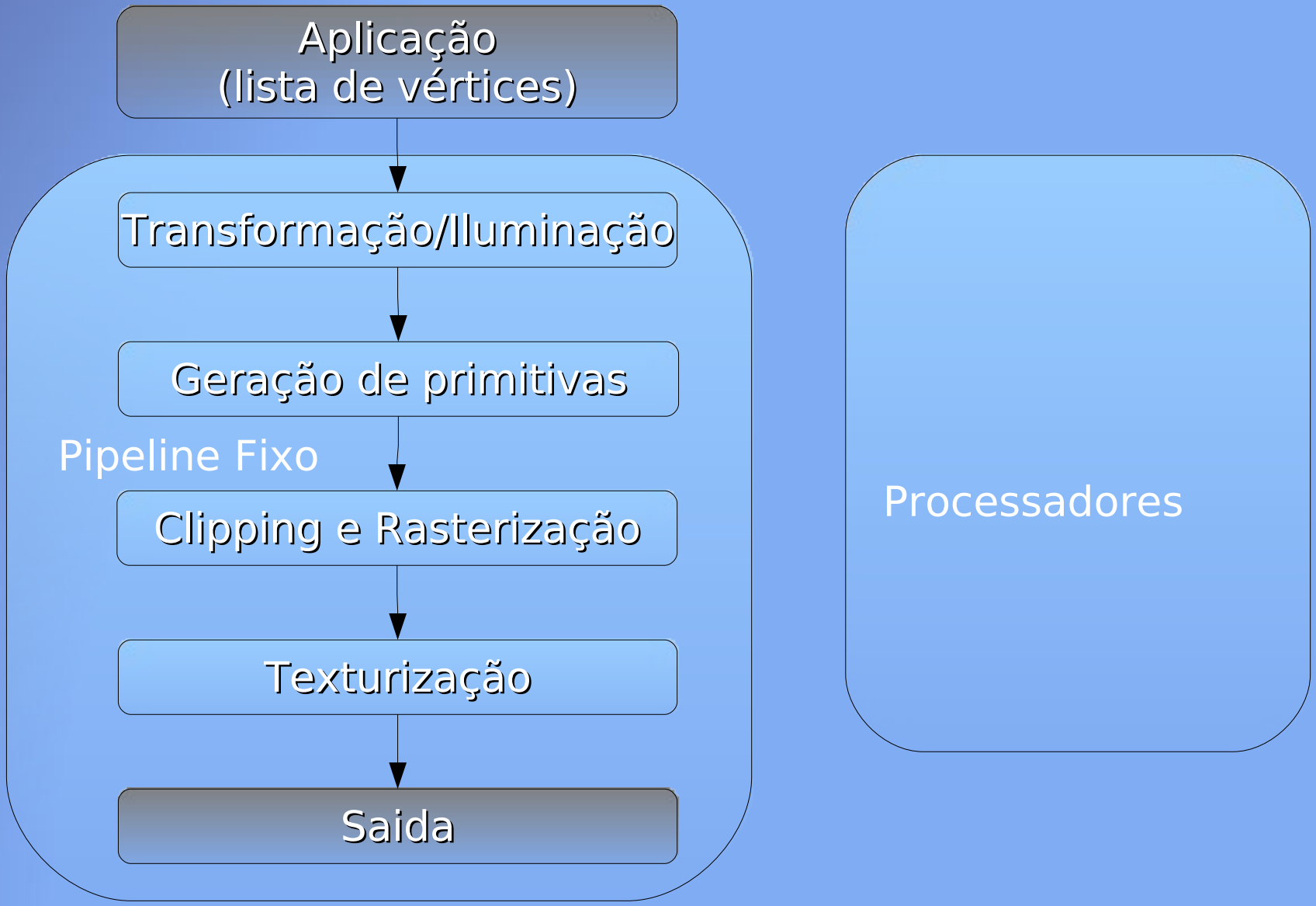
- Hardware doméstico
  - 2001
    - Programação por vértice (texture shader)
  - 2002-2003
    - Desenvolvimento da programação por Pixel
    - Suporte a linguagens de alto nível
    - Multiple Render Target
  - 2004
    - Suporte a branch(controle de fluxo dinâmico)
    - Suporte a leitura de textura no Proc. Vert.

Introdução a shaders
História

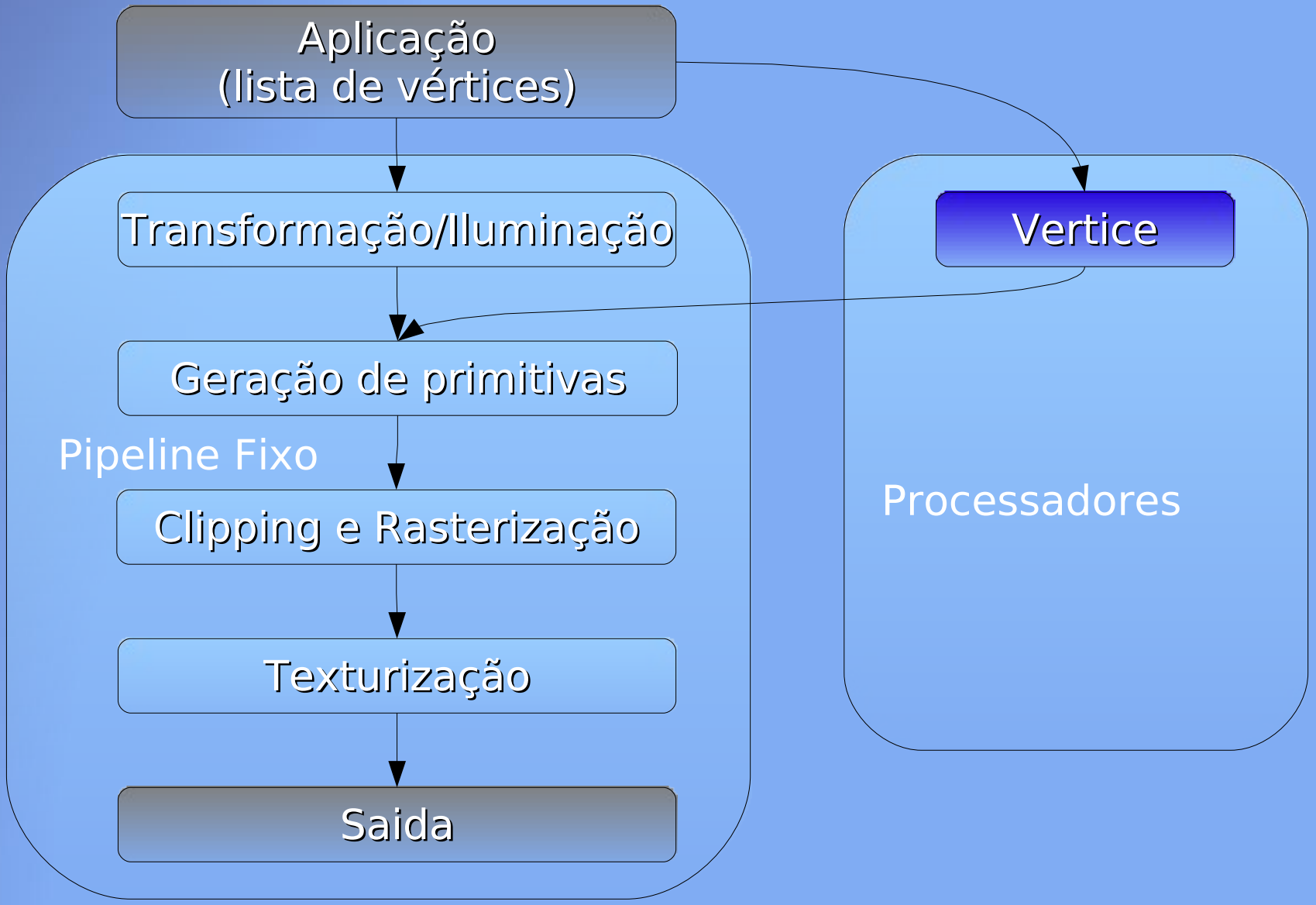
- Hardware
  - 2006-2007
    - Suporte a geração de primitivas
    - Estamos aqui!!!

Introdução a shaders
Pipeline de renderização abstrato

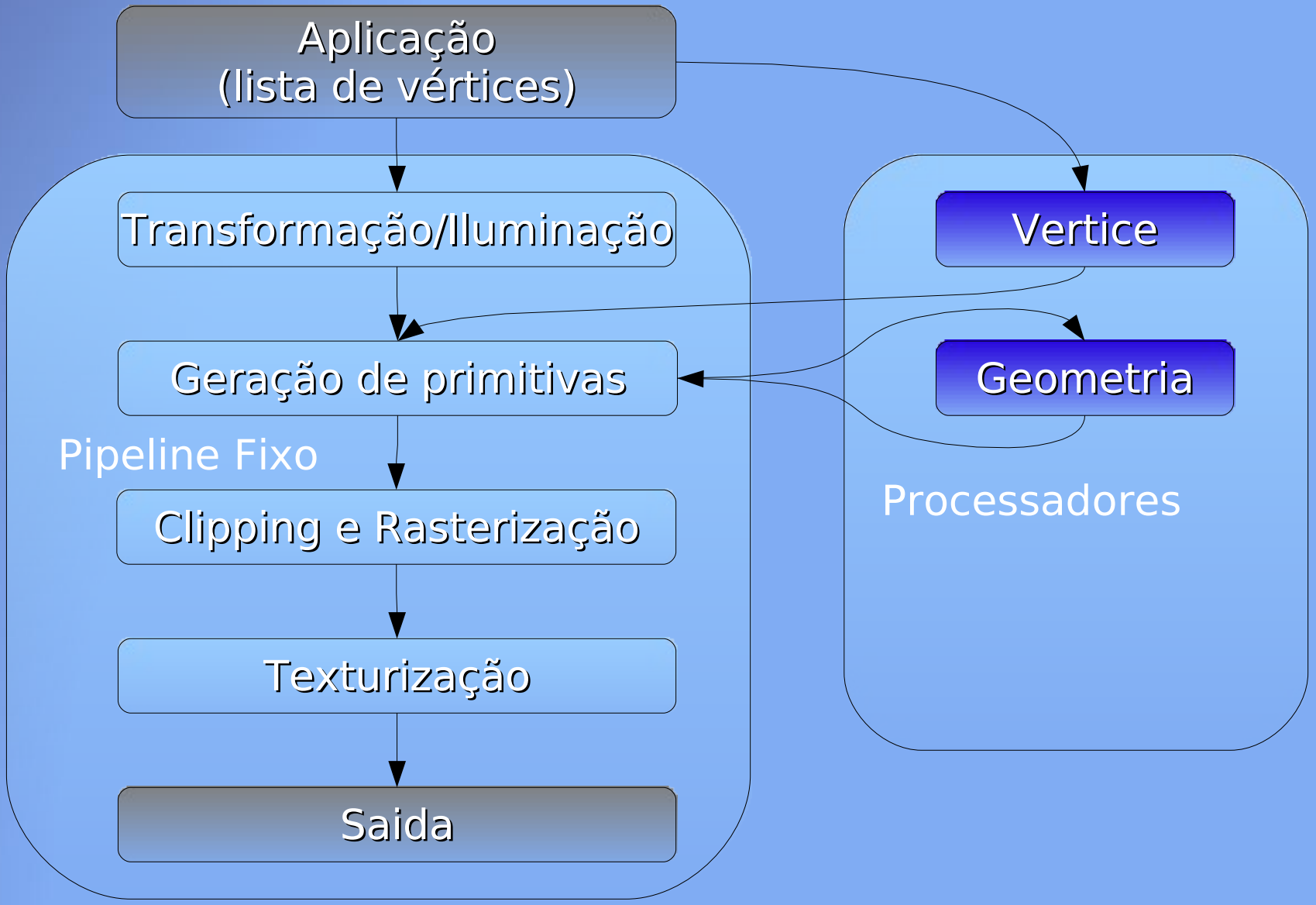
Introdução a shaders
Pipeline de renderização abstrato



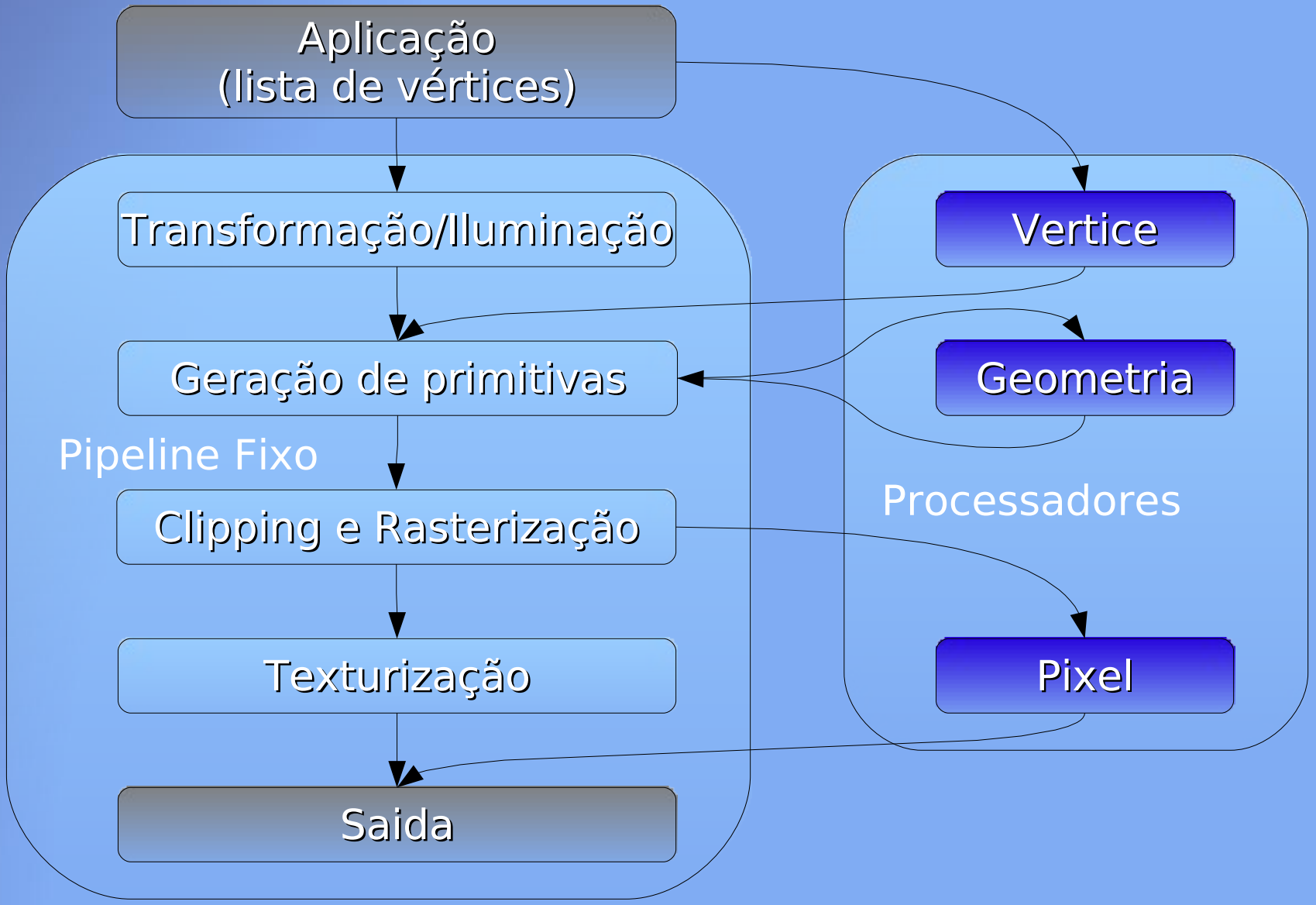
Introdução a shaders
Pipeline de renderização abstrato



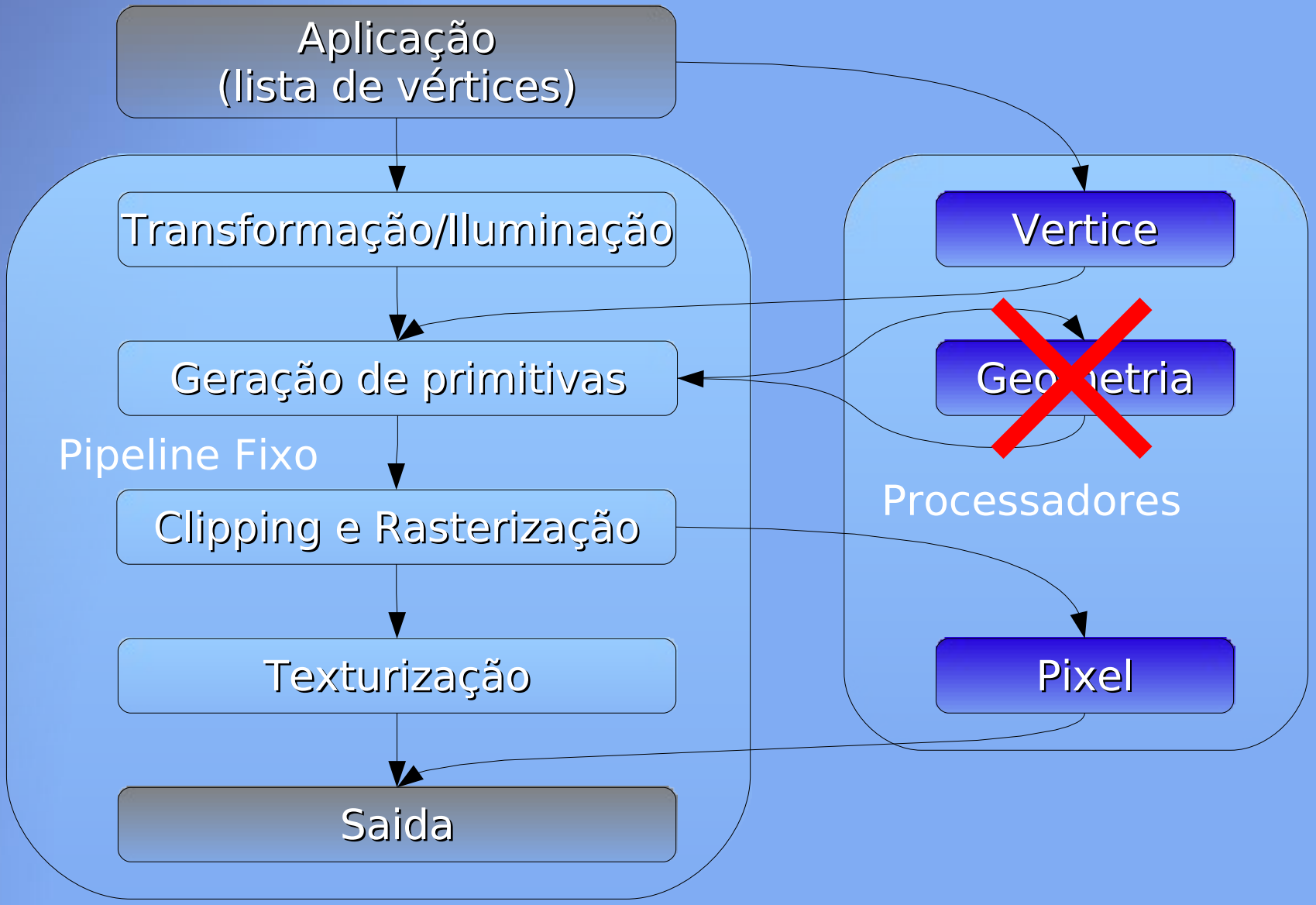
Introdução a shaders
Pipeline de renderização abstrato



Introdução a shaders
Pipeline de renderização abstrato



Introdução a shaders
Pipeline de renderização abstrato



Introdução a shaders
Linguagens para programar shader

- Assembly (DirectX, OpenGL)
  - A eficiência do código depende do programador
  - O desenvolvimento ou manutenção podem consumir muito tempo

Introdução a shaders
Linguagens para programar shader

```
!!ARBfp1.0
TEMP t1; TEMP t2; TEMP t3; TEMP t4; TEMP t5;
TEMP a;  TEMP b;  TEMP c;  TEMP d;  TEMP e;
ADD a,{0,-2},fragment.texcoord[0];
ADD b,{0,-1},fragment.texcoord[0];
MOV c,fragment.texcoord[0];
ADD d,{0, 1},fragment.texcoord[0];
ADD e,{0, 2},fragment.texcoord[0];
TEX t1, a, texture[0], RECT;
TEX t2, b, texture[0], RECT;
TEX t3, c, texture[0], RECT;
TEX t4, d, texture[0], RECT;
TEX t5, e, texture[0], RECT;
ADD t1,t1,t2;
ADD t3,t3,t4;
ADD t1,t1,t3;
ADD t1,t1,t5;
MUL result.color.rgb,t1,{0.2,0.2,0.2};
END
```

Introdução a shaders
Linguagens para programar shader

- Linguagens de shader (HLSL – DirectX,
  GLSL – OpenGL,
  Cg – Nvidia)
  - Alto nível
  - A eficiência do código depende do compilador e programador
    - OBS.:
      - GLSL – o compilador está no driver de video

Introdução a shaders
Linguagens para programar shader

```
void main(out float4 outputColor: COLOR0,
          in float2 texcoord: TEXCOORD0,
          in float4 inputColor:COLOR,
          uniform samplerRECT texture){
  float3 aux=0;
  for(int i=-2;i<=2;i++)
    aux += texRECT(texture, texcoord + float2(0,i) ).rgb;
  outputColor = float4(aux*0.2,1);
}
```

Introdução a shaders
Linguagens para programar shader

- Cg - “C” for Graphics
  - Portável (Windows, Linux, Mac e Solaris)
  - A linguagem é praticamente a mesma linguagem do HLSL
    - Pode-se utilizar shaders de uma em outra com pouco esforço de conversão

Introdução a shaders
Linguagens para programar shader

- Cg - “C” for Graphics
  - Não é implementada em driver de video ou mesmo em hardware
    - É uma linguagem virtual independente de API
  - Funciona com OpenGL ou DirectX
    - Compila os shaders para uma das linguagens nativas de cada API (GLSL, ARB_FP, HLSL, ...)
    - Como possui um parser e compilador interno, o mesmo ainda pode realizar otimizações em seu código

Introdução a shaders
Linguagens para programar shader

- Cg - “C” for Graphics
  - Baseado em perfis (Profiles) de compilação
    - Para saber qual conjunto de recursos serão necessários
      - Utilização ou não de branch dinâmico
      - Faz a query automática de extensões OGL
- `Perfis:`
  - `arbfp1, fp40, ps_3_0, ...`
- `Ver página 52 do UserGuide.pdf`

CGFX

- Formato de arquivo para utilização do Cg que permite controle de estados das APIs e quais shaders podem ser utilizados separados por técnicas

CGFX

Exemplo

```
float4x4 ModelViewProjMatrix : ModelViewProjectionMatrix;
...
struct VertIn{
  float4 pos : POSITION;
  float4 Normal : NORMAL;};
struct Vert2Frag{
  float4 pos : POSITION;
  float4 Color: COLOR;};

void vert(uniform float4x4 mvp,...,
          in VertIn IN, out Vert2Frag OUT){ ... }

float4 frag(in Vert2Frag IN):COLOR{ return IN.Color; }

technique teste <int it = 10; string name = "hoho";> {
 pass P0{
  VertexProgram = compile arbvp1 vert(ModelViewProjMatrix,...);
  FragmentProgram = compile arbfp1 frag();
 }
}
```

CGFX

Exemplo

```
float4x4 ModelViewProjMatrix : ModelViewProjectionMatrix;
...
struct VertIn{
  float4 pos : POSITION;
  float4 Norm
struct Vert2F
  float4 pos
  float4 Colo

void vert(uniform [illegible] mvp,...,
          in VertIn IN, out Vert2Frag OUT){ ... }

float4 frag(in Vert2Frag IN):COLOR{ return IN.Color; }

technique teste <int it = 10; string name = "hoho";> {
 pass P0{
  VertexProgram = compile arbvp1 vert(ModelViewProjMatrix,...);
  FragmentProgram = compile arbfp1 frag();
 }
}
```

Matriz
- float2x2
- float3x3
- float4x4

CGFX

Exemplo

```
float4x4 ModelViewProjMatrix : ModelViewProjectionMatrix;
...
struct VertIn{
  float4 pos : POSITION;
  float4 Normal : NORMAL;};
struct Vert2Frag{
  float4 pos : POSITION;
  float4 Color
Vetor
    float
    float2
    float3
    float4
void vert(unif                    ,...,
          in Ve                   ert2Frag OUT){ ... }

float4 frag(in Vert2Frag IN):COLOR{ return IN.Color; }

technique teste <int it = 10; string name = "hoho";> {
 pass P0{
  VertexProgram = compile arbvp1 vert(ModelViewProjMatrix,...);
  FragmentProgram = compile arbfp1 frag();
 }
}
```

CGFX

Exemplo

```
float4x4 ModelViewProjMatrix : ModelViewProjectionMatrix;
...
struct VertIn{
  float4 pos : POSITION;
  float4 Normal : NORMAL;};
struct Vert2Frag{
  float4 pos : POSITION;
  float4 Color: COLOR;};

void vert(uniform float4x4 mvp,...,
          in VertIn IN, out Vert2Frag OUT){ ... }

float4 frag(in Vert2Frag IN):COLOR{ return IN.Color; }

technique teste <int it = 10; string name = "hoho";> {
 pass P0{
  VertexProgram = compile arbvp1 vert(ModelViewProjMatrix,...);
  FragmentProgram = compile arbfp1 frag();
 }
}
```

Semântica definida pelo programador

CGFX

Exemplo

```
float4x4 ModelViewProjMatrix : ModelViewProjectionMatrix;
...
struct VertIn{
  float4 pos : POSITION;
  float4 Normal : NORMAL;};
struct Vert2Frag{
  float4 pos : POSITION;
  float4 Color: COLOR;};

void vert(uniform float4x4
          in VertIn IN, ou                    . }

float4 frag(in Vert2Frag I                   Color; }

technique teste <int it = 10; string name = "hoho";> {
 pass P0{
  VertexProgram = compile arbvp1 vert(ModelViewProjMatrix,...);
  FragmentProgram = compile arbfp1 frag();
 }
}
```

Semântica definida pela API Cg
- POSITON
- NORMAL
- TEXCOORD0
- ...

CGFX

Exemplo

```
float4x4 ModelViewProjMatrix : ModelViewProjectionMatrix;
...
struct VertIn{
  float4 pos : POSITION;
  float4 Normal : NORMAL;};
struct Vert2Frag{
  float4 pos : POSITION;
  float4 Color: COLOR;};

void vert(uniform float4x4 mvp,...,
          in VertIn IN, out Vert2Frag OUT){ ... }

float4 frag(in Vert2Frag IN):COLOR{ return IN.Color; }

technique teste <i                              'hoho";> {
 pass P0{
  VertexProgram =                               ewProjMatrix,...);
  FragmentProgram = compile arbfp1 frag();
 }
}
```

Qualificador do parâmetro
- uniform
- in
- out

CGFX

Exemplo

```
float4x4 ModelViewProjMatrix : ModelViewProjectionMatrix;
...
struct VertIn{
  float4 pos : POSITION;
  float4 Normal : NORMAL;};
struct Vert2Frag{
  float4 pos : POSITION;
  float4 Color: COLOR;};

void vert(uniform float4x4 mvp,...,
          in VertIn IN, out Vert2Frag OUT){ ... }

float4 frag(in Vert2F                    Color; }

technique teste <int it = 10; string name = "hoho";> {
 pass P0{
  VertexProgram = compile arbvp1 vert(ModelViewProjMatrix,...);
  FragmentProgram = compile arbfp1 frag();
 }
}
```

Anotação
O compilador ignora

CGFX

Exemplo

```
float4x4 ModelViewProjMatrix : ModelViewProjectionMatrix;
...
struct VertIn{
  float4 pos : POSITION;
  float4 Normal : NORMAL;};
struct Vert2Frag{
  float4 pos : POSITION;
  float4 Color: COLOR;};

void vert(uniform float4x4 mvp,...,
          in VertIn IN, out Vert2Frag OUT){ ... }

float4 frag                                    Color; }

technique teste <int it = 10; string name = "hoho";> {
 pass P0{
  VertexProgram = compile arbvp1 vert(ModelViewProjMatrix,...);
  FragmentProgram = compile arbfp1 frag();
 }
}
```

Definição da técnica
Composta de um ou mais passos

CGFX

Exemplo

```
float4x4 ModelViewProjMatrix : ModelViewProjectionMatrix;
...
struct VertIn{
  float4 pos : POSITION;
  float4 Normal : NORMAL;};
struct Vert2Frag{
  float4 pos : POSITION;
  float4 Color: COLOR;};

void vert(uniform float4x4 mvp,...,
          in VertIn IN, out Vert2Frag OUT){ ... }

float4 frag(in V                                    }

technique teste                                     {
 pass P0{
  VertexProgram = compile arbvp1 vert(ModelViewProjMatrix,...);
  FragmentProgram = compile arbfp1 frag();
 }
}
```

Definição dos algoritmos utilizados
- Processamento de vértice
- Processamento de fragmento

CGFX

Exemplo

```
float4x4 ModelViewProjMatrix : ModelViewProjectionMatrix;
...
struct VertIn{
  float4 pos : POSITION;
  float4 Normal : NORMAL;};
struct Vert2Frag{
  float4 pos : POSITION;
  float4 Color: COLOR;};

void vert(uniform float4x4 mvp,...,
          in VertIn IN, out Vert2Frag OUT){ ... }

float4 frag(in Vert2Frag IN):COLO

technique teste <int it = 10; str
 pass P0{
  VertexProgram = compile arbvp1 vert(ModelViewProjMatrix,...);
  FragmentProgram = compile arbfp1 frag();
 }
}
```

Definição do profile que será utilizado

CGFX

Exemplo

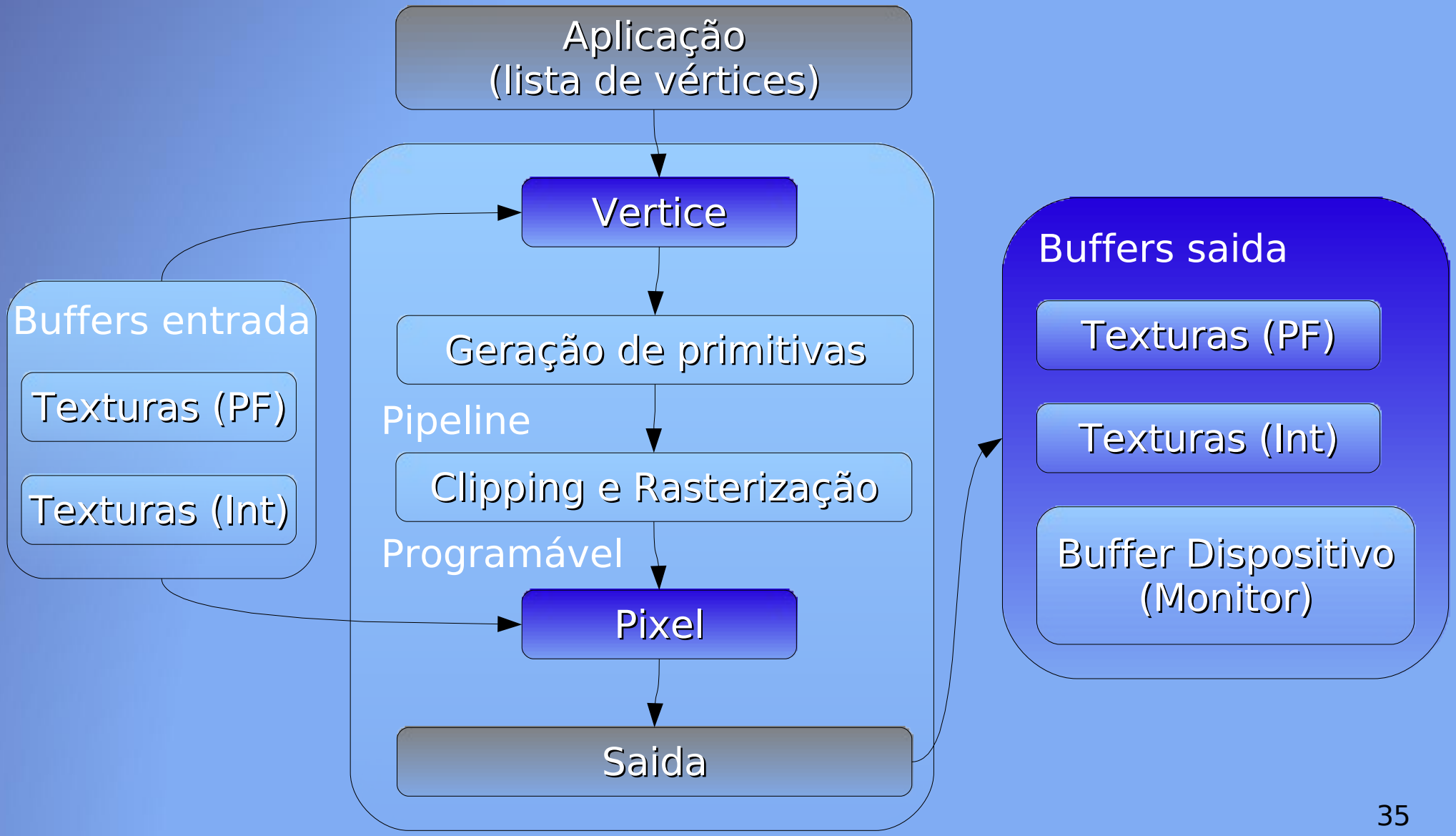
```
float4x4 ModelViewProjMatrix : ModelViewProjectionMatrix;
...
struct VertIn{
  float4 pos : POSITION;
  float4 Normal : NORMAL;};
struct Vert2Frag{
  float4 pos : POSITION;
  float4 Color: COLOR;};

void vert(uniform float4x4 mvp,...,
          in VertIn IN, out Vert2Frag OUT){ ... }

float4 frag(in Vert2Frag IN):COLOR{ return IN.Color; }

technique teste <int it = 10; string name = "hoho";> {
 pass P0{
  VertexProgram = compile arbvp1 vert(ModelViewProjMatrix,...);
  FragmentProgram = compile arbfp1 frag();
 }
}
```

CGFX

- Os uniforms são passados da aplicação para o shader, assim como as semânticas
- TODOS os dados passados do proc. de vértices para o proc. de pixels são interpolados linearmente
- IMPORTANTE: O controle de buffers de saida deve ser feito na aplicação

CGFX

Pipeline de renderização abstrato



Projeto Peixis

- Programação de efeitos de Pos-Processamento

- Peixis utiliza a Torque Game Engine 1.5
  - Com suporte a D3D e OGL
  - Possui um mecanismo de renderização complexo
    - Não é bom mexer no que está funcionando

Projeto Peixis

- Acrescentando efeitos à engine:
  - Criação de DLL
  - Especificação dos efeitos desejados
  - Implementação em CGFX
- Interface DLL
  - Inicialização
  - Antes de renderizar
  - Depois de renderizar
  - Finalização
  - Algumas chamadas para configuração

Projeto Peixis

- Efeitos:
  - Distorção de agua
  - Alteração de cor (contraste e brilho)
  - Desfoque de movimento (blur radial)
    - O efeito evoliu com observações feitas a outros jogos
  - Diminuir aliasing da cena (bloom)
    - A água deu a impressão de ser turva (comentário do texturizador Paulo)

Projeto Peixis

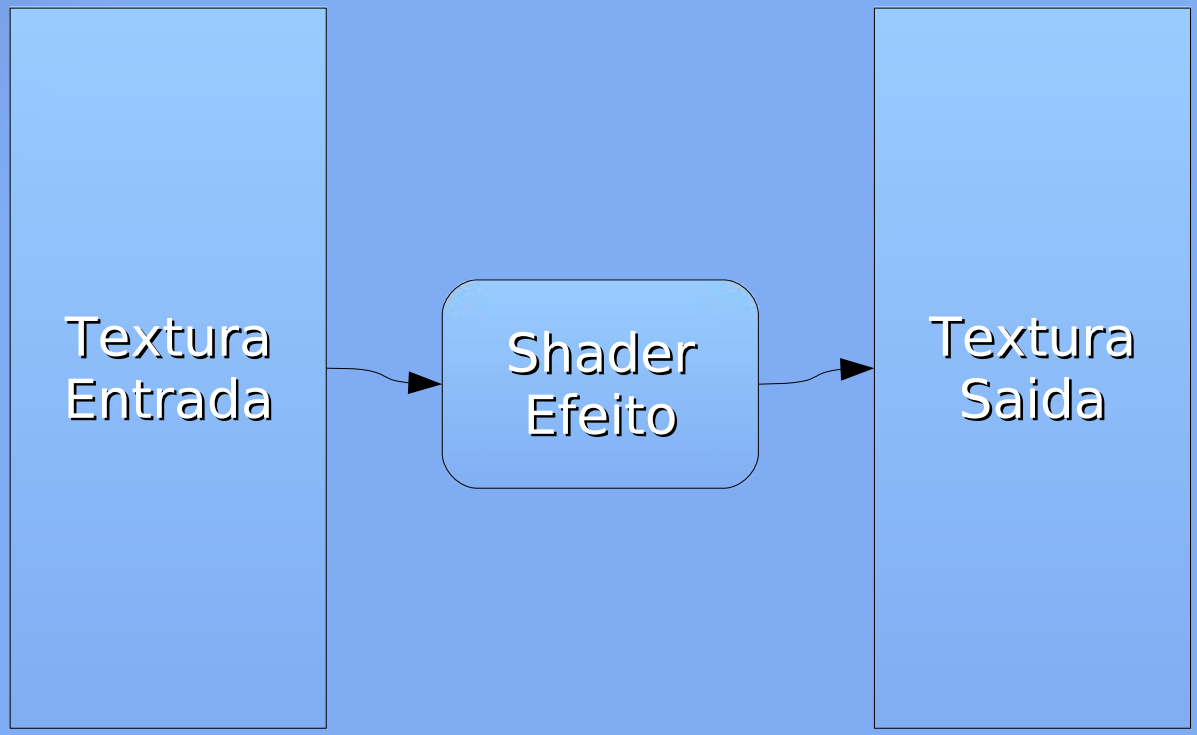
- Cada efeito geralmente é abordado de forma isolada em documentações de linguagens, SDKs, etc...
  - Mas preciso de alguma forma de interconecta-los!!!!

Projeto Peixis

- Como dito anteriormente
  - A aplicação deve definir os buffers
- Alguns efeitos precisam de mais que um passo de renderização como o Bloom e Blur
- Foi criada uma arquitetura que permita o fluxo de execução contínuo dos efeitos

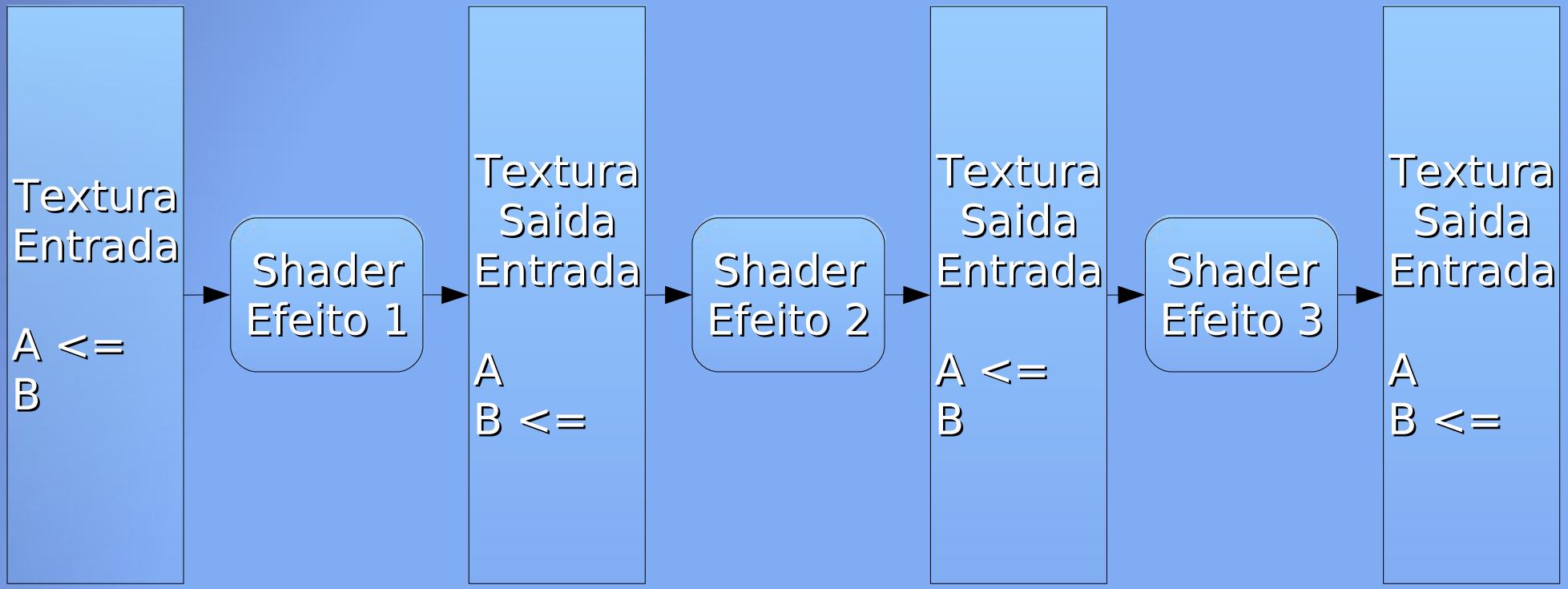
Projeto Peixis
Arquitetura

- Cada efeito individualmente é aplicado de forma sequencial



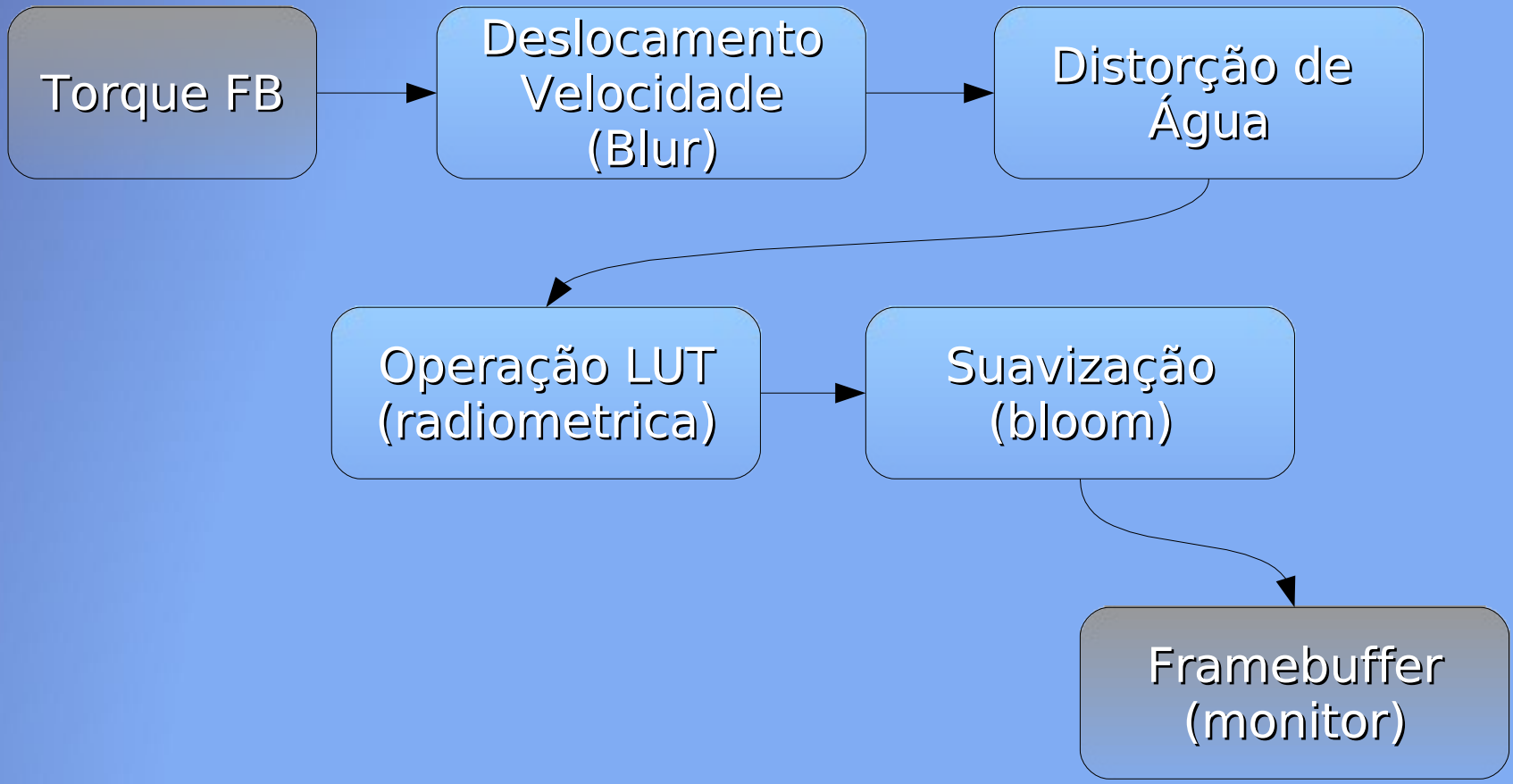
Projeto Peixis
Arquitetura

- É preciso de 2 buffers com resolução da tela para montar o pipeline de efeitos



Projeto Peixis
Arquitetura

- **Pipeline de pos-processamento do Peixis**



Projeto Peixis

- Ver o jogo funcionando!!!

Projeto Peixis – sem shader

Projeto Peixis – Bloom

Projeto Peixis – LUT (look up table)

Projeto Peixis – Distorção água

Projeto Peixis – Deslocamento velocidade

Processamento de vertice

- É possível alterar o processamento de vertices para efetuar alguma operação de transformação elaborada
  - Distorções/oscilações
  - Computação e conversão de espaços de vertices
  - etc...

Materiais

- A primeira utilização para os shaders esta ligada a utilização de Materiais

- São propriedades da superfície de um objeto utilizados para sua renderização
  - Plástico
  - Roupa
  - Asfalto
  - etc...

Materiais

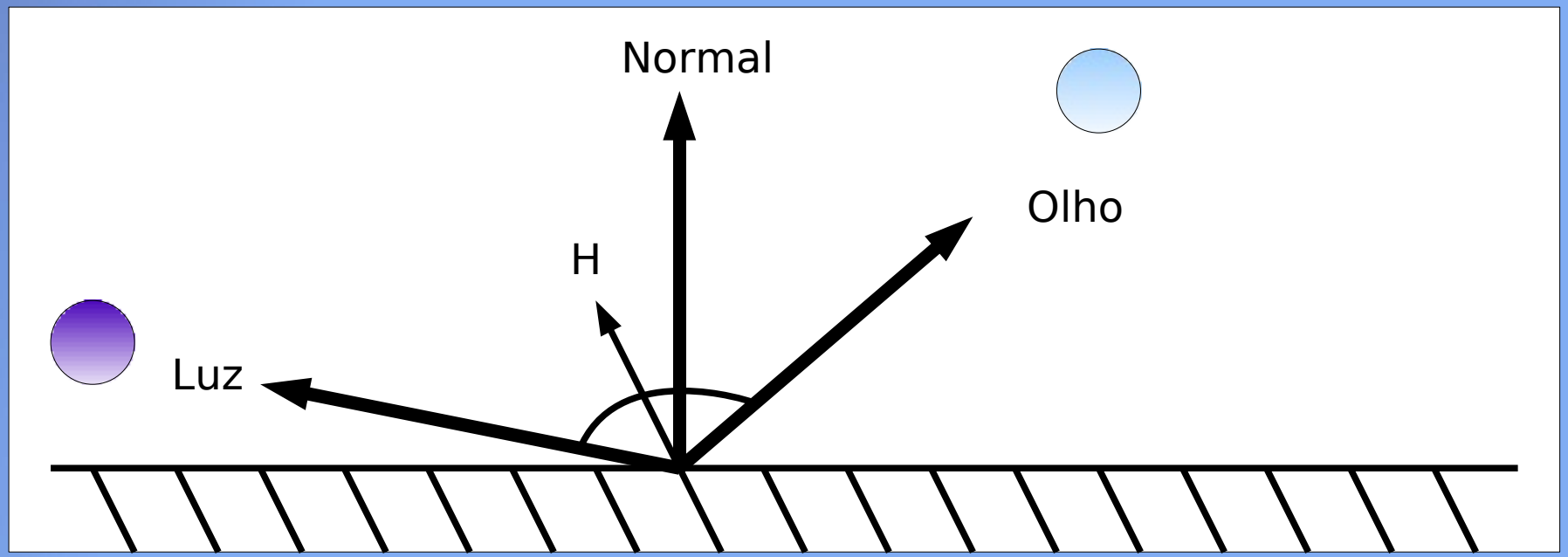
- Definição básica
  - Utilizar a própria cor do vértice como saida
  - Neste shader utiliza-se o pipeline fixo de vértice e implementa somente o processamento de pixel

```
void RenderJustColorFS(in  float4 inputColor :COLOR0,
                       out float4 outputColor:COLOR0){
  outputColor = inputColor;
}

technique RenderJustColor {
 pass P0{
  FragmentProgram = compile arbfp1 RenderJustColorFS();
 }
}
```

Materiais

Cálculo de iluminação

- Blinn Phong (halfway vector)



Materiais

Cálculo de iluminação

- Blinn Phong (halfway vector)
  - Algoritmo implementado no pipeline fixo do OpenGL e DirectX
  - Iluminação calculadas por Vértices ou por Pixel

$$H = \frac{(Olho + L)}{|Olho + L|}$$

$$Kd = N.L\,(Lambert)$$

$$Ke = H.N\,(Simplificação\ Phong)$$

$$Cor = Ka * ambiente + Kd * corDifusa + Ke^{pow} * corRefletida$$

Materiais

Cálculo de iluminação

- Blinn Phong (halfway vector) Vertice

```
void BlinPhongVertexVS(...){
  ...
  float3 h = normalize(eye + light );
  float kd = saturate(dot(normal, light));
  float ke = saturate(dot(normal, h));
  outputColor = 0.02*color + kd*color*0.5 + pow(ke,256)*float3(1);
}

void BlinPhongVertexFS(in  float3 inputColor :COLOR0,
                       out float4 outputColor:COLOR0){
  outputColor = float4(inputColor,1);
}
```

Materiais

Cálculo de iluminação

- Blinn Phong (halfway vector) Pixel

```
void BlinPhongVertexVS(uniform mat4x4 MVP,
                       in  float4 position:POSITION,
                       in  float4 normal:NORMAL,
                       in  float3 color:COLOR0,
                       out float4 outputPosition:POSITION,
                       out float3 NormalVarying,
                       out float3 HVarying,
                       out float3 LightVarying,
                       out float3 outputColor:COLOR0){
  ...
  HVarying = normalize(eyeVec + light);
  NormalVarying = normalVec;
  LightVarying = light;
  outputColor = color;
}
```

Materiais

Cálculo de iluminação

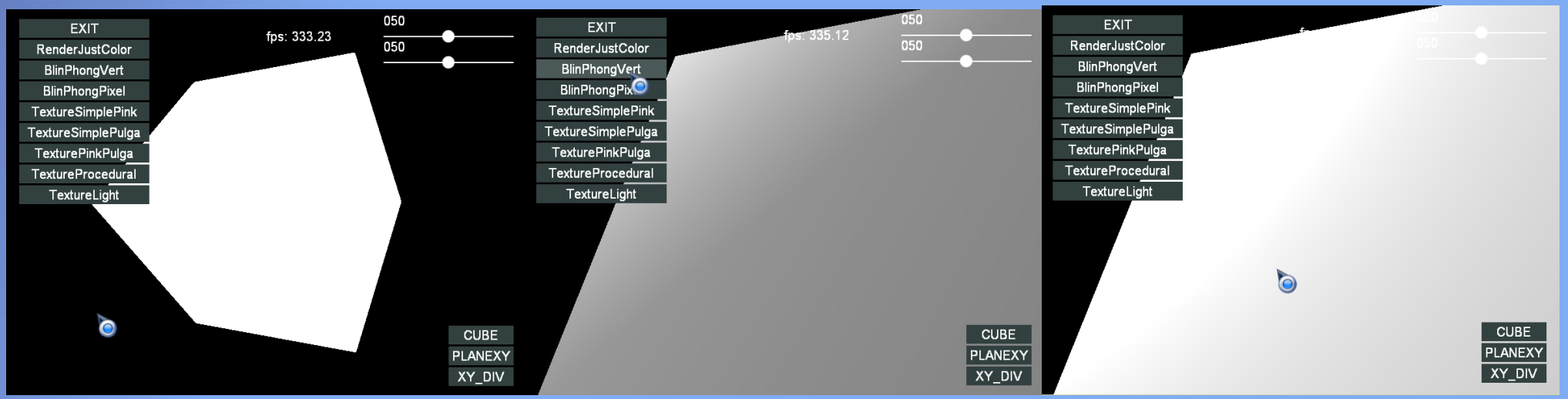
- Blinn Phong (halfway vector) Pixel

```
void BlinPhongVertexFS(in float3 NormalVarying,
                       in float3 HVarying,
                       in float3 LightVarying,
                       in  float3 inputColor :COLOR0,
                       out float4 outputColor:COLOR0){
  float3 normal = normalize(NormalVarying);
  float kd = saturate(dot(normal, normalize(LightVarying)));
  float ke = saturate(dot(normal, normalize(HVarying)));
  outputColor = 0.2*inputColor + kd*inputColor*0.5 +
                pow(ke,256)*float3(1);
}
```

Materiais

Cálculo de iluminação

- Ver exemplo 01 – iluminação até pixel



Cor Blinn(vertice) Blinn(pixel)

Materiais

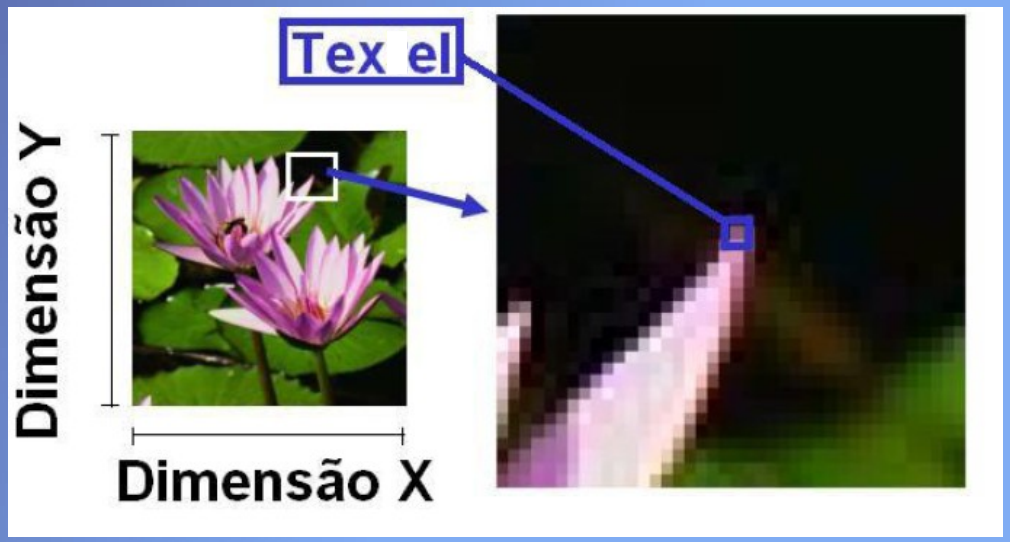
Texturização

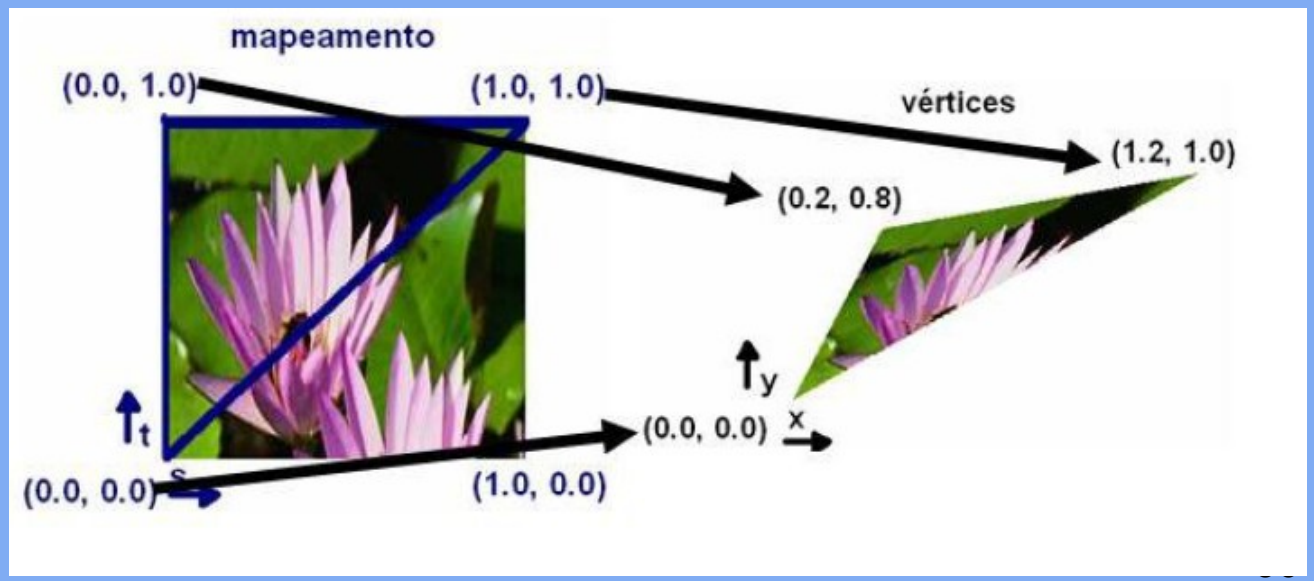
- Texturização é a técnica de utilizar uma imagem para preenche os pixels de um triângulo
  - Pode ser visto como uma função de mapeamento bidimencional

- Para utilizar uma textura é necessário definir como a mesma deve ser amostrada
  - Configuração de interpolação
  - Definição de acessos fora dos limites

Materiais

Texturização





Materiais
Texturização

- Definição da amostra

```
sampler2D nome <string filename = "minha anotação";>
= sampler_state {  WrapS = Clamp;
                   WrapT = Clamp;
                   MinFilter = LinearMipMapLinear;
                   MagFilter = Linear; };
```

- Acesso a uma textura

```
float4 textureSimple():COLOR0{
...
  float4 cor = tex2D(tex_color,IN.texCoord);
...
}
```

- Este processo pode ser utilizado para acessar várias texturas

Materiais
Texturização

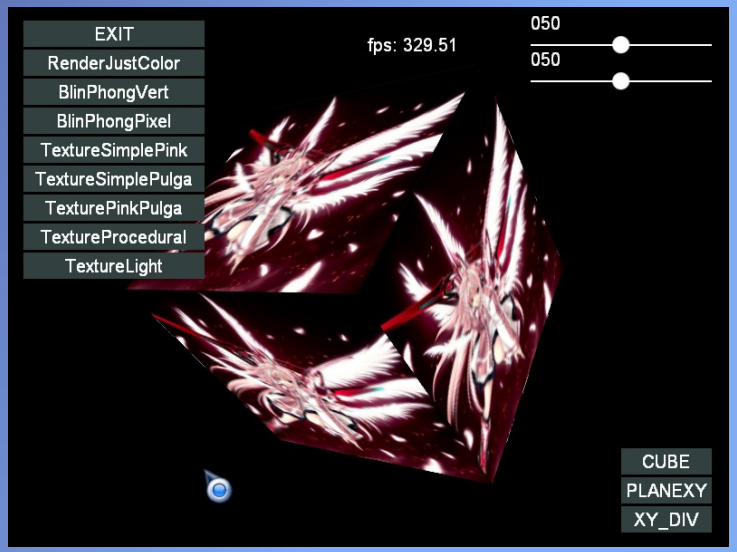
- Pode ser definida uma textura de forma procedural
  - Não ocupa memória de vídeo
  - Permite o mapeamento em todo o espaço do objeto
- Possível porque a função de textura deve retornar uma cor a partir de uma coordenada

```
float3 proceduralTextura(float3 coord){
  float freq = lerp1;//parametro
  float3 retorno = 1;
  if(freq>0)
    retorno = float3(saturate(sign(fmod(coord.x,freq)-freq*0.5 ) ));
  return retorno;
}
float4 textureSimple(...):COLOR0{
  return float4(proceduralTextura(IN.texcoord),1);
}
```

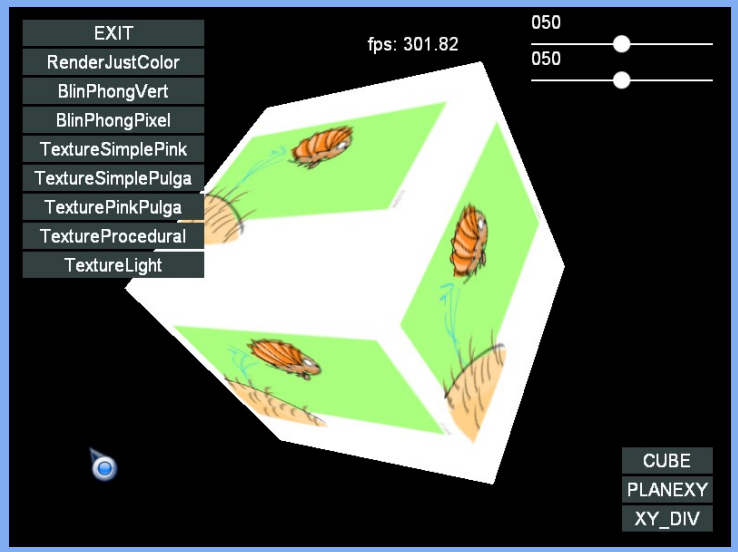
Materiais

Texturização

- Exemplo 01 – iluminação – textura

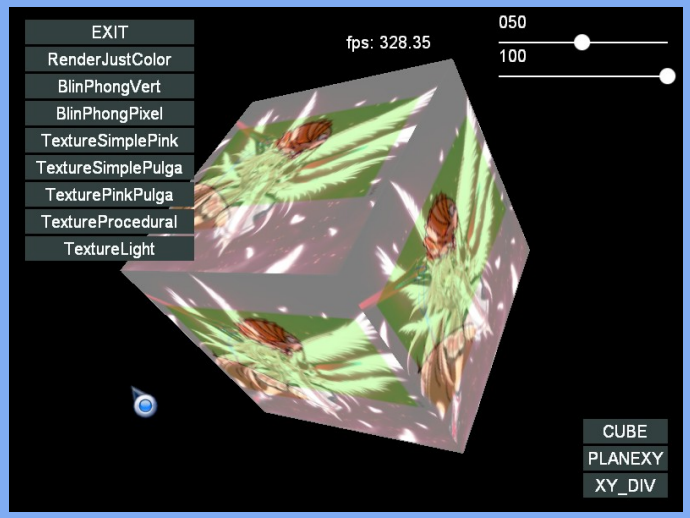


Tex1



Tex2

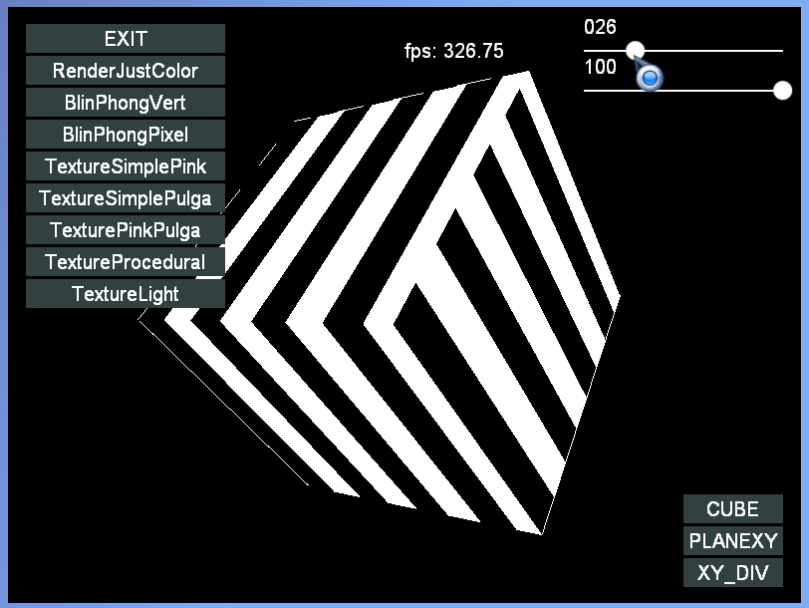
Tex1+Tex2



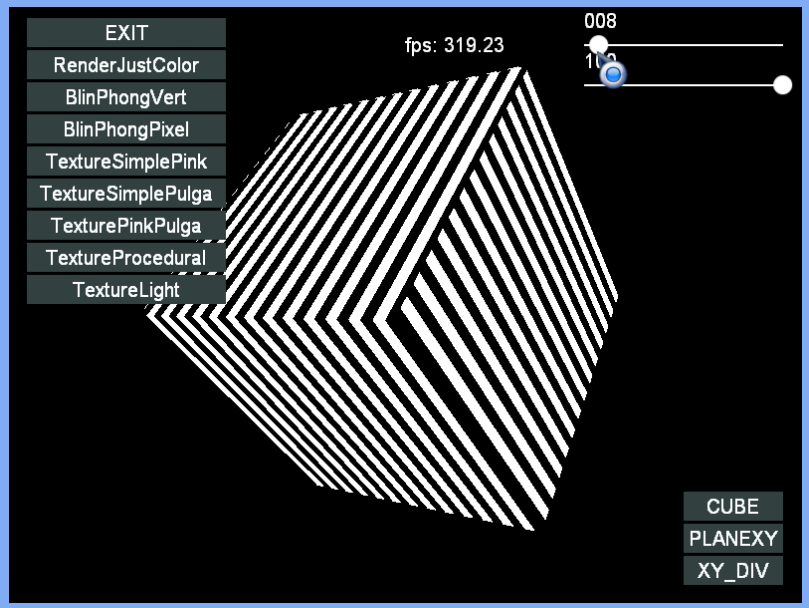
Materiais

Texturização

- Exemplo 01 – iluminação – textura



Parametro 028



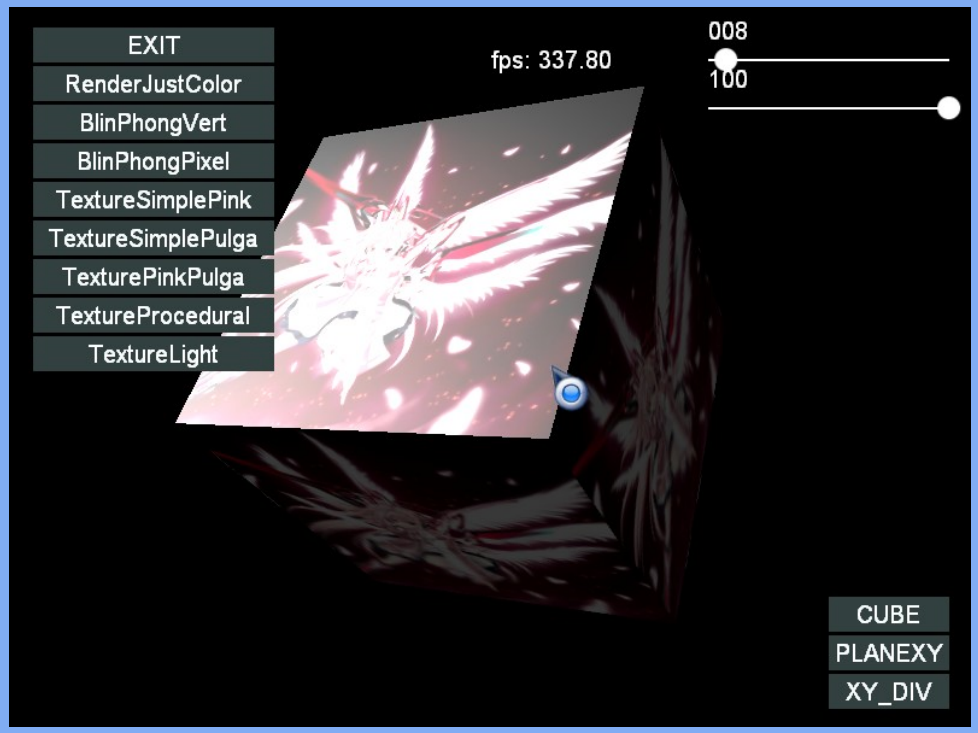
Parametro 009

Textura procedural

Materiais

Existem infinidades de efeitos

- É possível criar novos efeitos para objetos juntando os que você ja tem ou a partir de novas fórmulas e mapeamentos
  - Exemplo de textura + luz



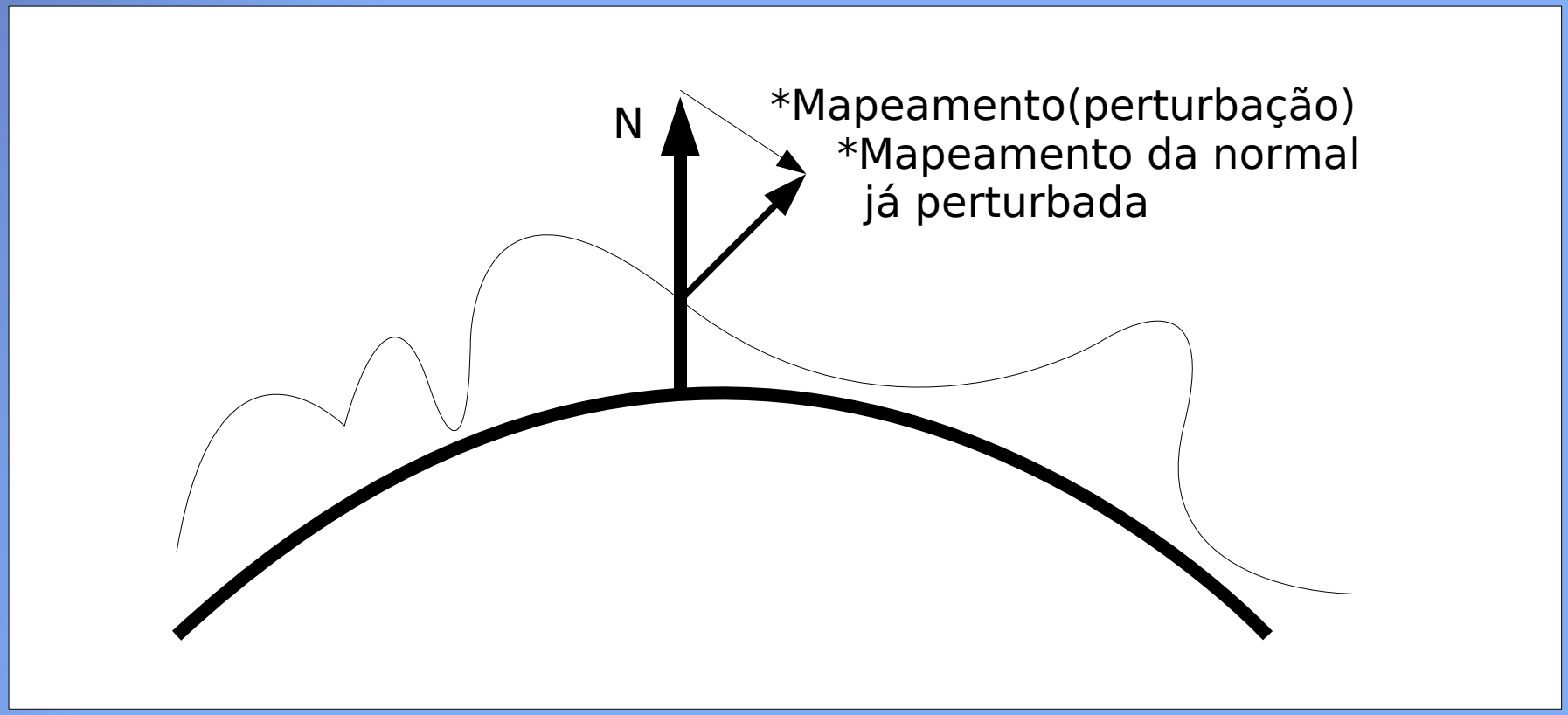
Materiais

Renderização de detalhes

- Com shader é possível implementar técnicas mais elaboradas
  - *Bump mapping
  - *Cone step mapping
  - Enviroment mapping
  - Shadow mapping
  - etc...

Materiais
Renderização de detalhes

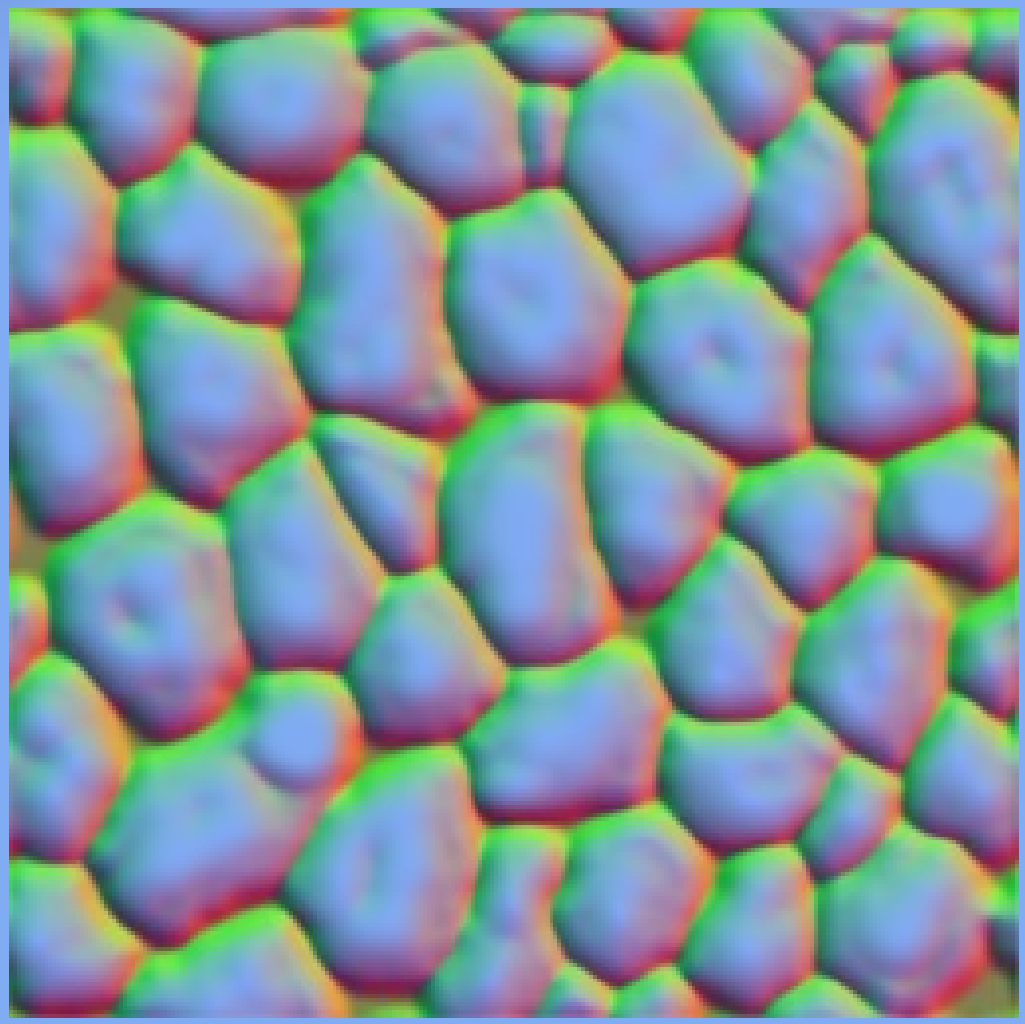
- Bump



Materiais

Renderização de detalhes

- Bump(textura com dados ao invés de cores)



Materiais

Renderização de detalhes

- Cone step mapping (CSM)
  - Utiliza mapa de bump mas também implementa um mini-raytracer no Processador de fragmentos

Materiais

Renderização de detalhes

- Ambas as técnicas podem ser calculadas no espaço da tangente
  - ??????????????

Materiais

Renderização de detalhes

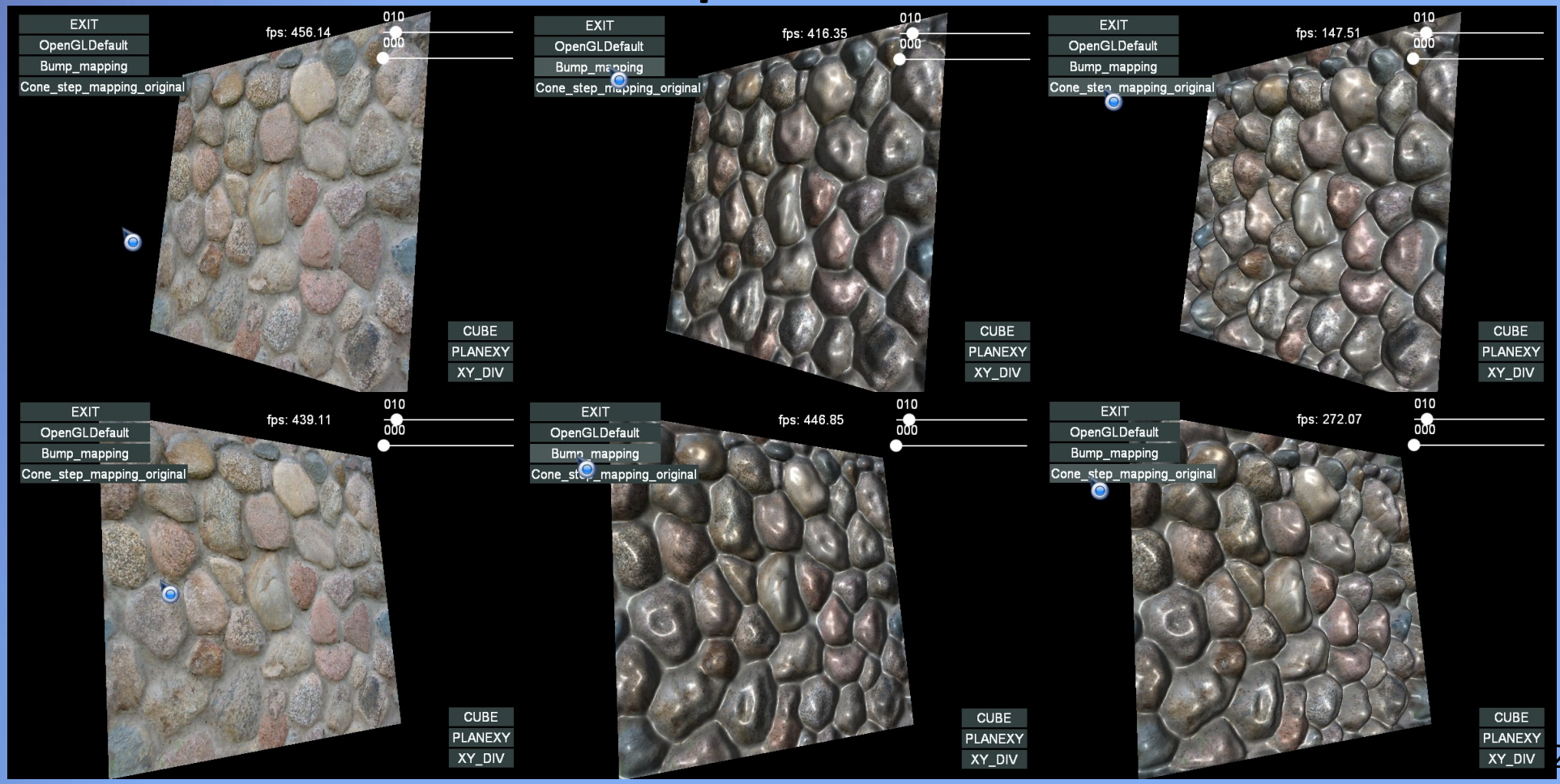
- Ambas as técnicas podem ser calculadas no espaço da tangente
  - Facilita o calculo de perturbação
  - A informação das tangentes e binormais devem fazer parte do modelo como as normais

Materiais

Renderização de detalhes

- Ver exemplo 02

normal bump CSM



Pos-processamento(PDI)

- Imagem em PDI(Processamento Digital de Imagens) :
  - Função bidimensional
  - Matriz 2D com intensidades

- Operações
  - Transformações radiométricas
  - Filtros
  - Composição
  - ...

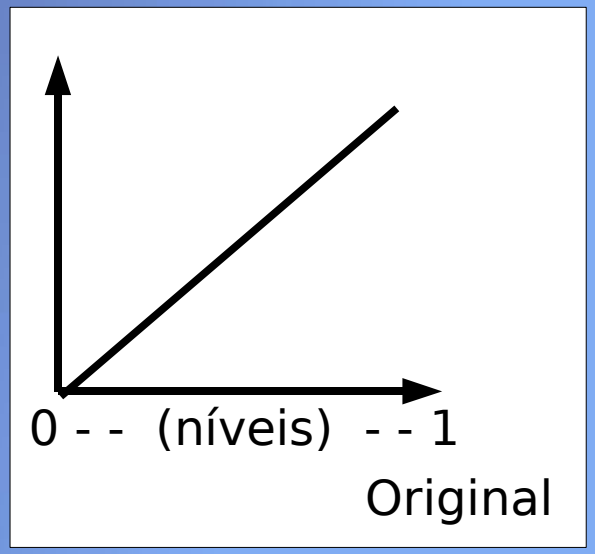
Pos-processamento(PDI)
Transformações radiométricas

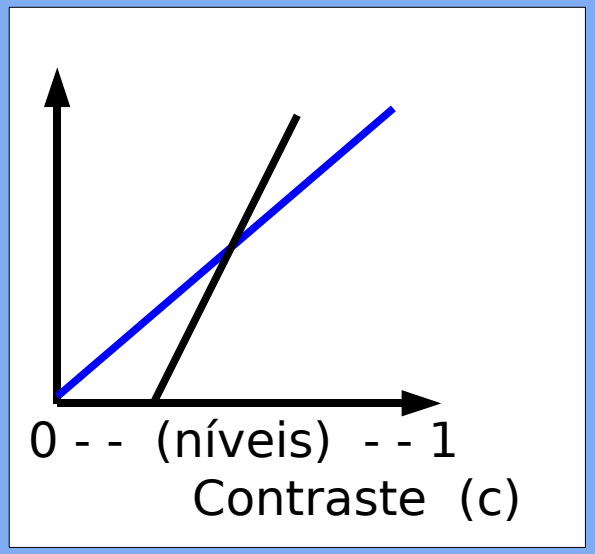
- Operações que refletem diretamente no histograma
  - Operações de LUT
    - Contraste
    - Brilho
    - Gamma
    - Conversão de sistema de cor
    - Composição
    - etc

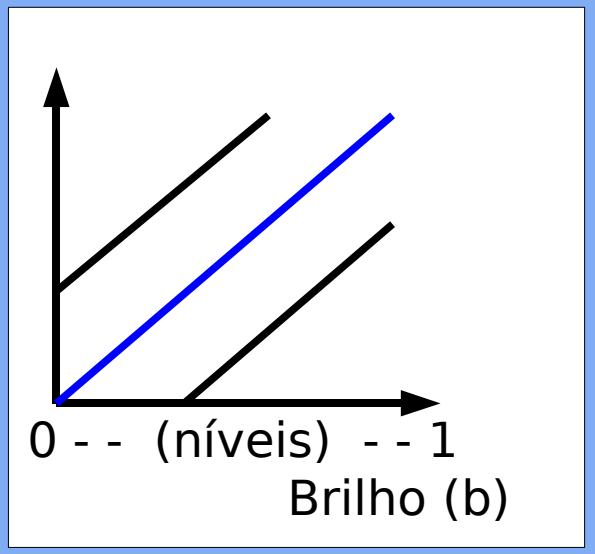
- Ver livro PDI para mais tipos de operações

Pos-processamento(PDI)
Transformações radiométricas

- Contraste e Briho







$(f(x,y)-0.5)*c + 0.5$

$f(x,y)+b$

Pos-processamento(PDI)
Transformações radiométricas

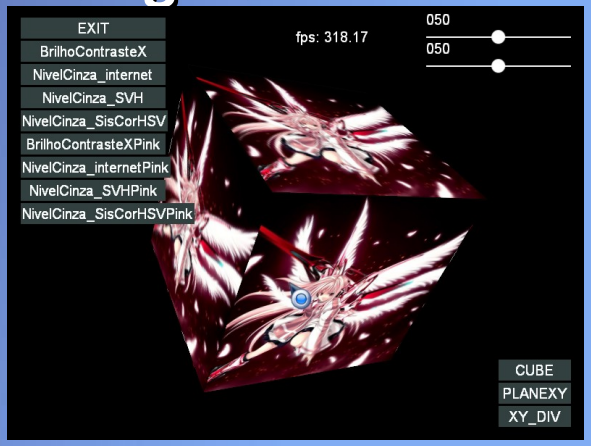
- Extração de tons de cinza

  - cinza = (vermelho+verde+azul)/3 (Value - HSV)
  - cinza = (vermelho * 0.299) +
    (verde * 0.587) +
    (azul * 0.114) (SVH)
  - cinza = máximo(vermelho, verde, azul)
    (encontrado na internet)

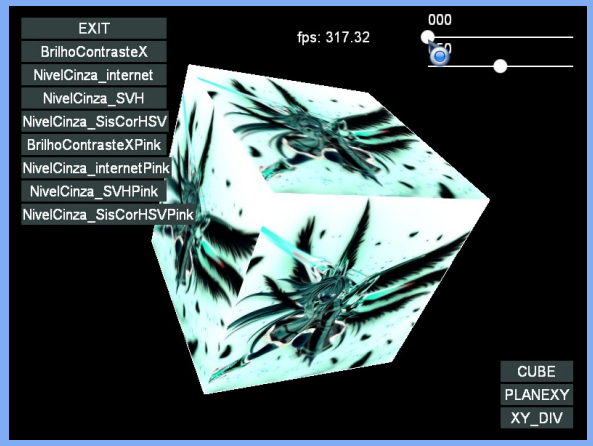
Pos-processamento(PDI)
Transformações radiométricas

- Ver exemplo 03

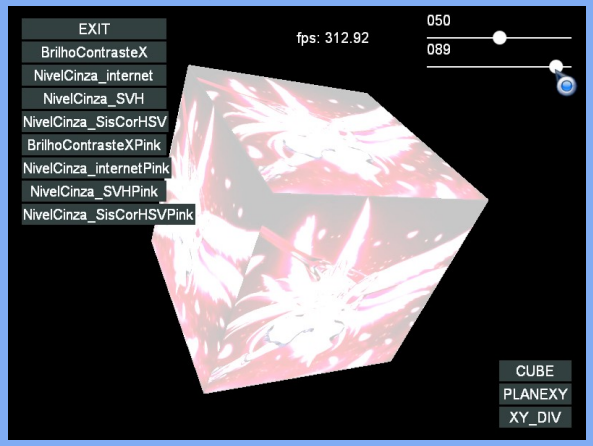
Original



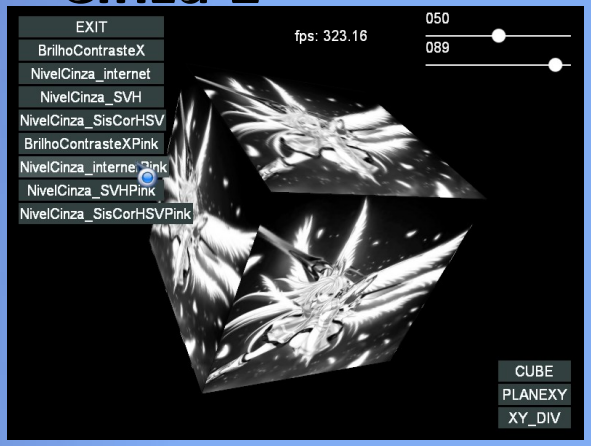
Contraste



Brilho



Cinza 1



Cinza 2

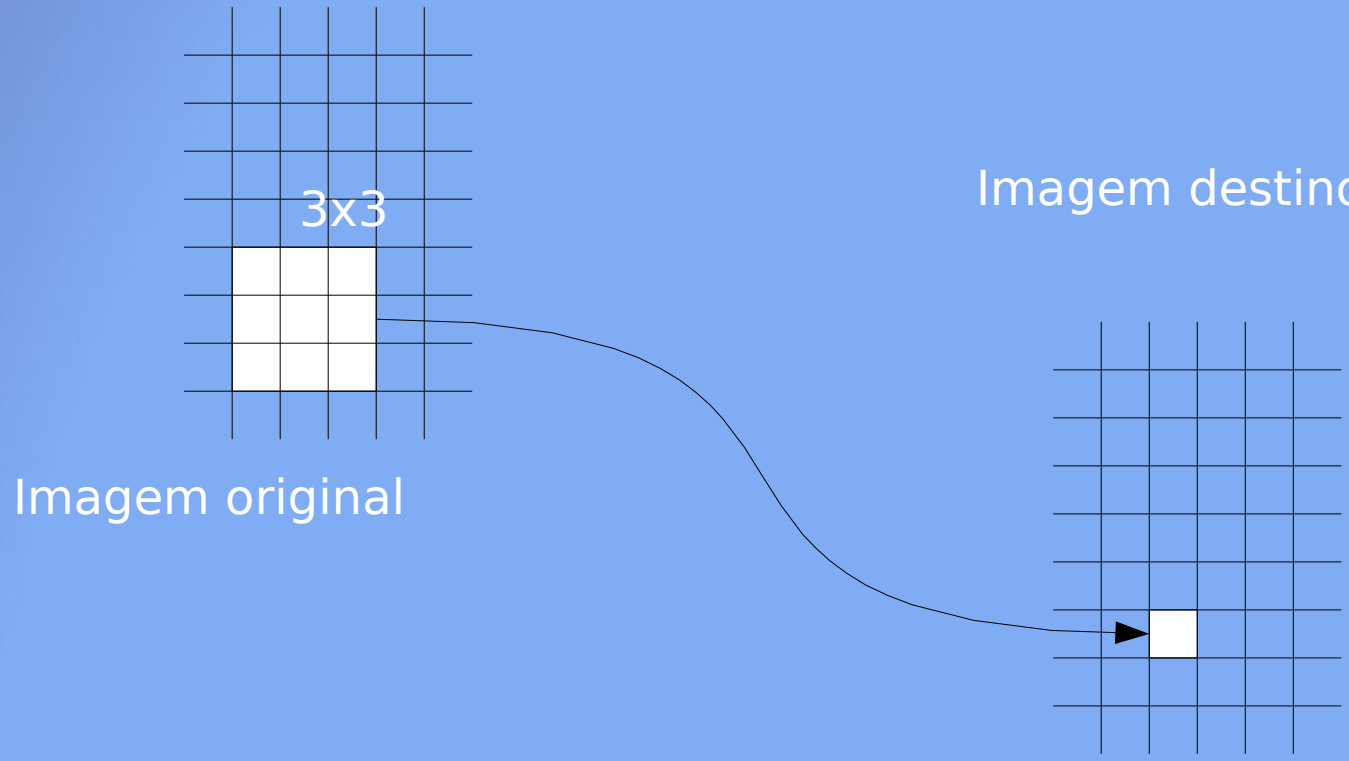
Pos-processamento(PDI)
Filtros

- Utilizados para:
  - Redução de ruidos e suavização (aliasing, blur)
  - Detecção de descontinuidade (gradientes e bordas)

- Também é possível implementá-los em shaders
  - Shader permite acesso aleatório a textura

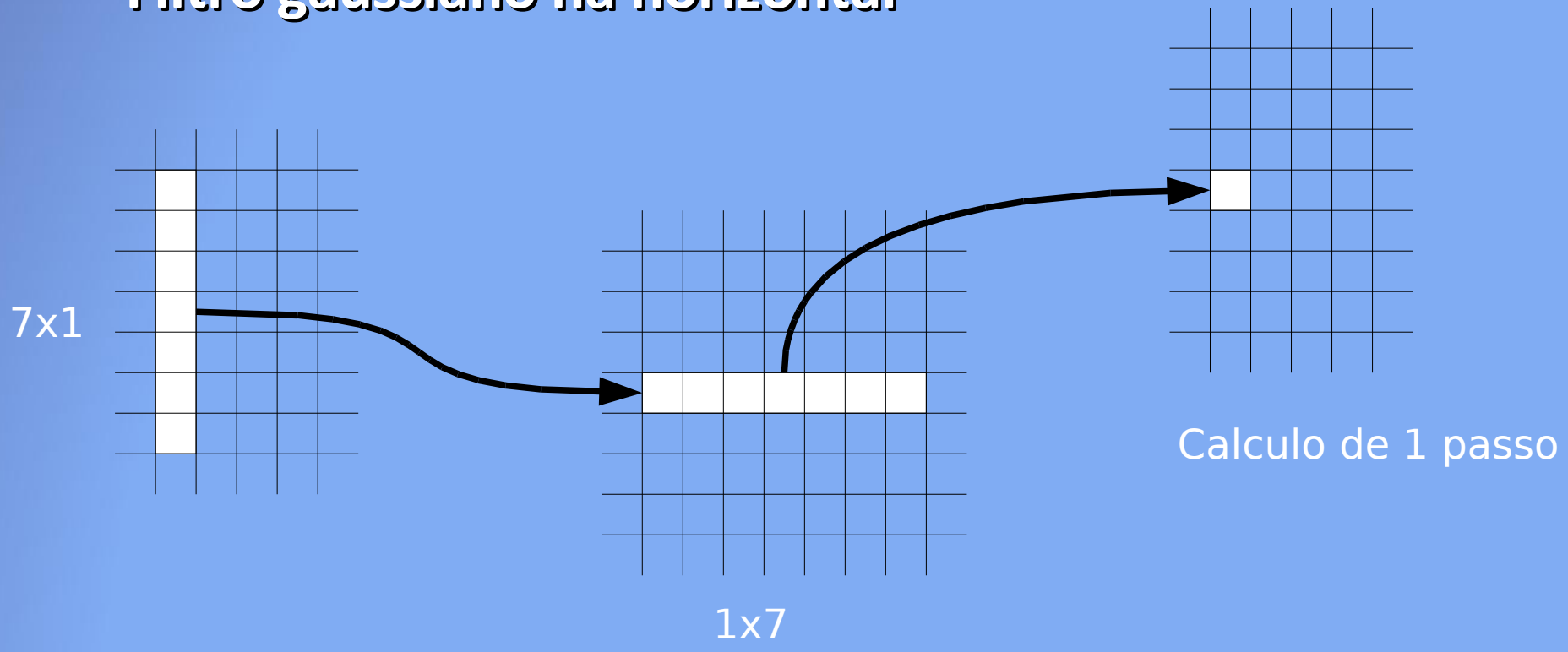
Pos-processamento(PDI)
Filtros

- Aplicar mascara e realizar operações sobre toda a imagem



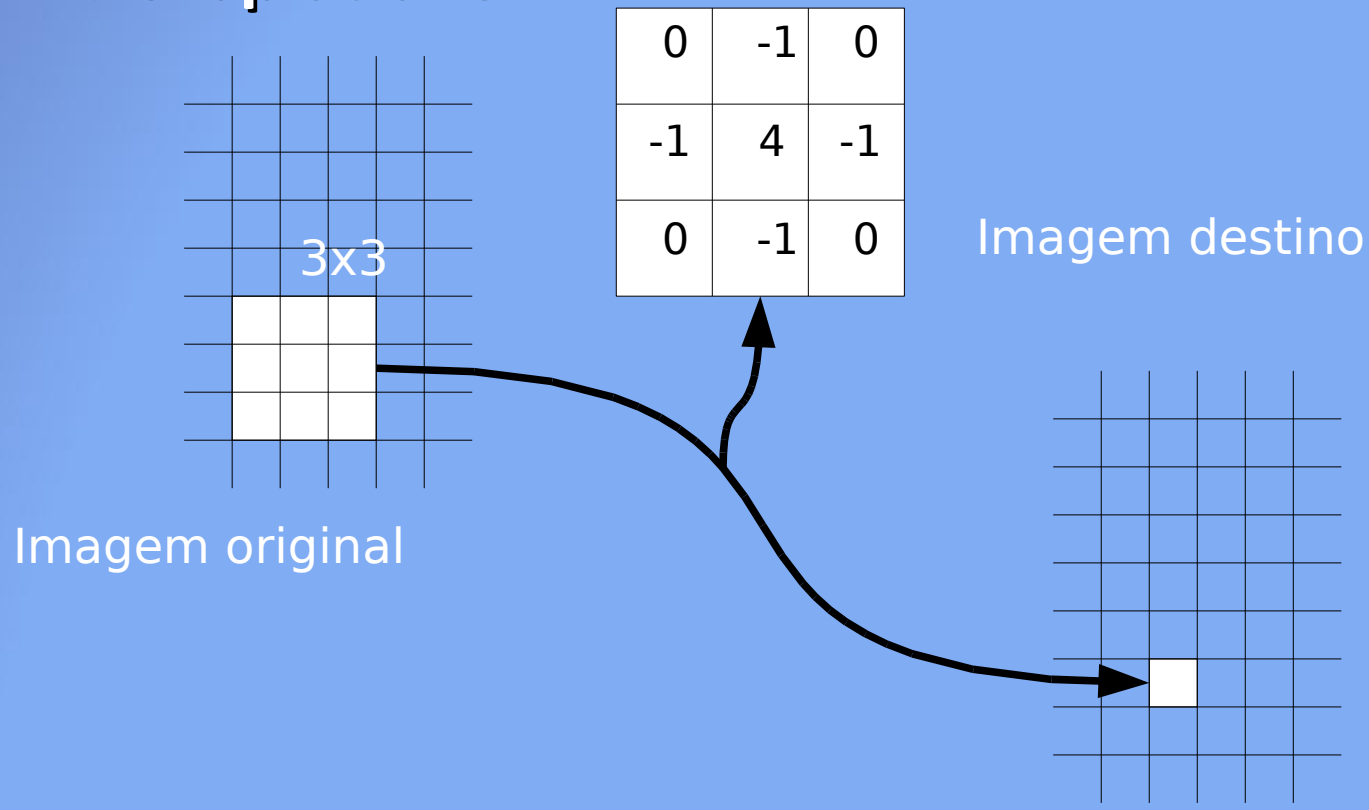
Pos-processamento(PDI)
Filtros

- Blur – implementação típica em GPU - O(n)
  - Filtro gaussiano na vertical
  - Filtro gaussiano na horizontal



Pos-processamento(PDI)
Filtros

- Laplace
  - Filtro laplaciano



Pos-processamento(PDI)
Composição

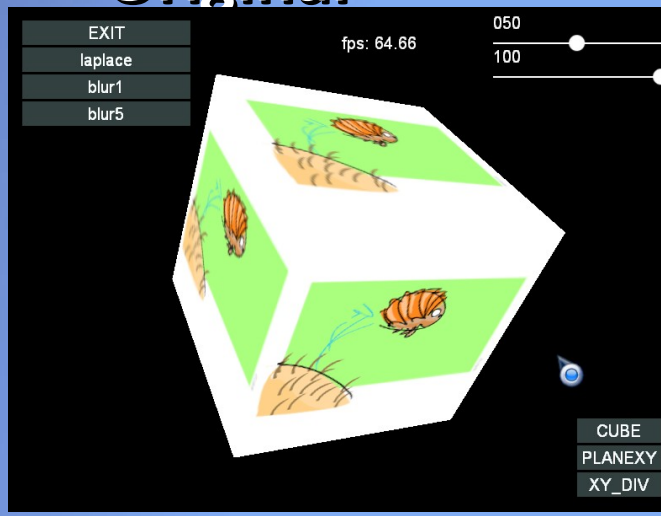
- Uma composição é a utilização de uma ou mais imagens e combiná-las de alguma forma
  - Exemplo da implementação de composição de texturas(01)

- Se a composição for entre 2 imagens, ela pode ser representada por uma combinação convexa de suas intensidades
  - ImagemFinal = (lrp)*ImagemA+(1-lrp)*ImagemB

Pos-processamento(PDI)
Composição

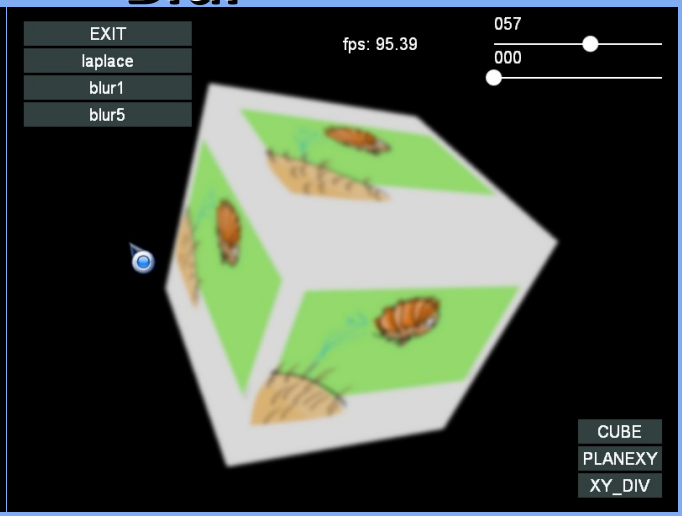
- Ver exemplo 03
  - Filtro Blur, Laplaciano e Composição

Original



Laplace

Blur



GPGPU

- Geralmente a implementação em GPU é mais eficiente que implementar em CPU
  - Exemplo Blur

- Então porque não utilizar a GPU para fazer computações matemáticas pesadas?

GPGPU

- Utilizar a GPU para renderizar gráficos é interessante, mas é possível programar a GPU para qualquer finalidade
  - Desde que o problema se assemelhe a computações de streams
- É interessante utilizar também a GPU para computações genéricas, pois o hardware gráfico programável é altamente paralelizado (para fazer renderizações a 60 QPS) e realiza de forma eficiente o calculo de ponto flutuante, além de possuir operações de matrizes implementadas em hardware

GPGPU

- APIs que podem ser utilizadas para programação de GPUs com a linguagem C:
  - Close to Metal(CTM) - ATI
  - Compute Unified Device Architecture(CUDA) – NVIDIA

- Para placas de video antigas ainda é necessário utilizar o pipeline gráfico programável
  - < Geforce série 8
  - < Radeon série R5x

GPGPU

Kernels

- Um kernel é o trecho de código executado para todos os dados que serão processados
  - Não sabem o que ja foi processado
  - Não podem utilizar resultados de processamentos do mesmo passo de outro kernel

GPGPU

Kernels

- Um kernel é a porção do código executada dentro do loop

```
/* Pseudocode */

x = 10e8
y = 10e8

make array x by y

for each "x" {    // Loop this block 10e8 times
  for each "y" { // Loop this block 10e8 times
    KERNEL_EXECUTE(x, y) // This is done 10e16 times
                         // (10 000 000 000 000 000)
  }
}
```

GPGPU

Possibilidades

- Mapeamento
- Reducao
- Filtro da stream
- Dispersão
- Agrupamento
- Ordenação
- Buscas em paralelo
- Estruturas variadas

GPGPU

Possibilidades

- Limitação dos dados a 32Bits
- Não suporta funções recursivas
- Conversão de ponto flutuante para inteiros pode ser diferente da CPU
- Sincronização global de uma GPU é ineficiente
- O barramento de transmissão CPU-GPU pode se tornar o gargalo da aplicação

GPGPU

Possibilidades

- Infelizmente não trouxe nenhuma implementação que utilize a GPU para computações genéricas
  - Vertex skinning
  - Processamento de Imagens
  - Processamento de simulação de Agentes (IA)
  - Calcular FFT
  - Simulação de fluidos
  - Etc...

- developer.nvidia.com (exemplos do SDK)

Efeitos compostos

- A possibilidade de programar a GPU permite:
  - Montagem de seu pipeline de renderização
  - Definir operações que processam vertices
  - Definir operações que processam pixels

- Nada impede de misturar várias técnicas

Considerações finais

- Crie
- Invente
- Tente

- Faça um Shader diferente!!!!

Considerações finais

- Questão de utilização de placas de videos
  - ATI
    - Se for utilizar OpenGL tenha cuidado
  - Nvidia
    - Desenvolve ferramentas interessantes
  - Ambas possuem restrições com relação a buffers e texturas
  - Faça o máximo para não escrever no Z-buffer :-)
  - Pesar algoritmos para shader de renderização e processamento genérico, ainda mais se ambos forem pesados

Considerações finais

- Tentei dar uma visão geral sobre a utilização de shaders
  - Existem uma infinidade de algoritmos e estudos que visam o desenvolvimentos de novos shaders
- Espero que tenham gostado

Perguntas?

Alessandro Ribeiro da Silva

Site: www.alessandrosilva.com

Email: alessandro.ribeiro.silva@gmail.com

asilva@dcc.ufmg.br