Desenvolvimento de uma Ferramenta que Realize a Verificação da Performance entre Rotinas Implementadas em CPU e GPU

Pablo Sidney Corrêa

Orientador: Dalton Solano dos Reis

Roteiro

- Introdução
- Objetivos do trabalho
- Conceitos e técnicas
- Trabalhos correlatos
- Requisitos/Especificação
- Operacionalidade
- Resultados/Conclusão/Extensões

Introdução

- Processamento de grande quantidade de dados
- Hardwares extras, melhor desempenho
- Divisão de tarefas

Objetivos do trabalho

- Analisar as principais características da GPU e da linguagem Cg
- Implementar em GPU uma rotina de dinâmica de fluídos já desenvolvida para CPU
- Desenvolver rotinas para comparação de tempo de processamento, memória e aspectos visuais

Conceitos e técnicas

- Hardware gráfico
  - Processamento paralelo
  - Memória para texturas
  - Evolução da placa
- Pipeline gráfico
  - Estágios do pipeline
  - Vertex Shader – substitui o primeiro estágio
  - Fragment Shader – substitui terceiro estágio (cadidato a pixel)

Conceitos e técnicas

- Linguagens
  - GLSL – 3DLabs\OpenGL
  - HLSL - Microsoft
  - Cg - NVidia
- Ambientes de desenvolvimento
  - FX Composer – HLSL e Cg
  - Render Monkey – GLSL e HLSL
  - ASHLI – GLSL e HLSL

Conceitos e técnicas

- Dinâmica de fluídos
  - Estuda o movimento de um fluído
  - Possui três etapas para realizar a simulação
  - Gera uma quantidade grande de dados
- Técnicas implementadas
  - Colormap
  - Vectorfield

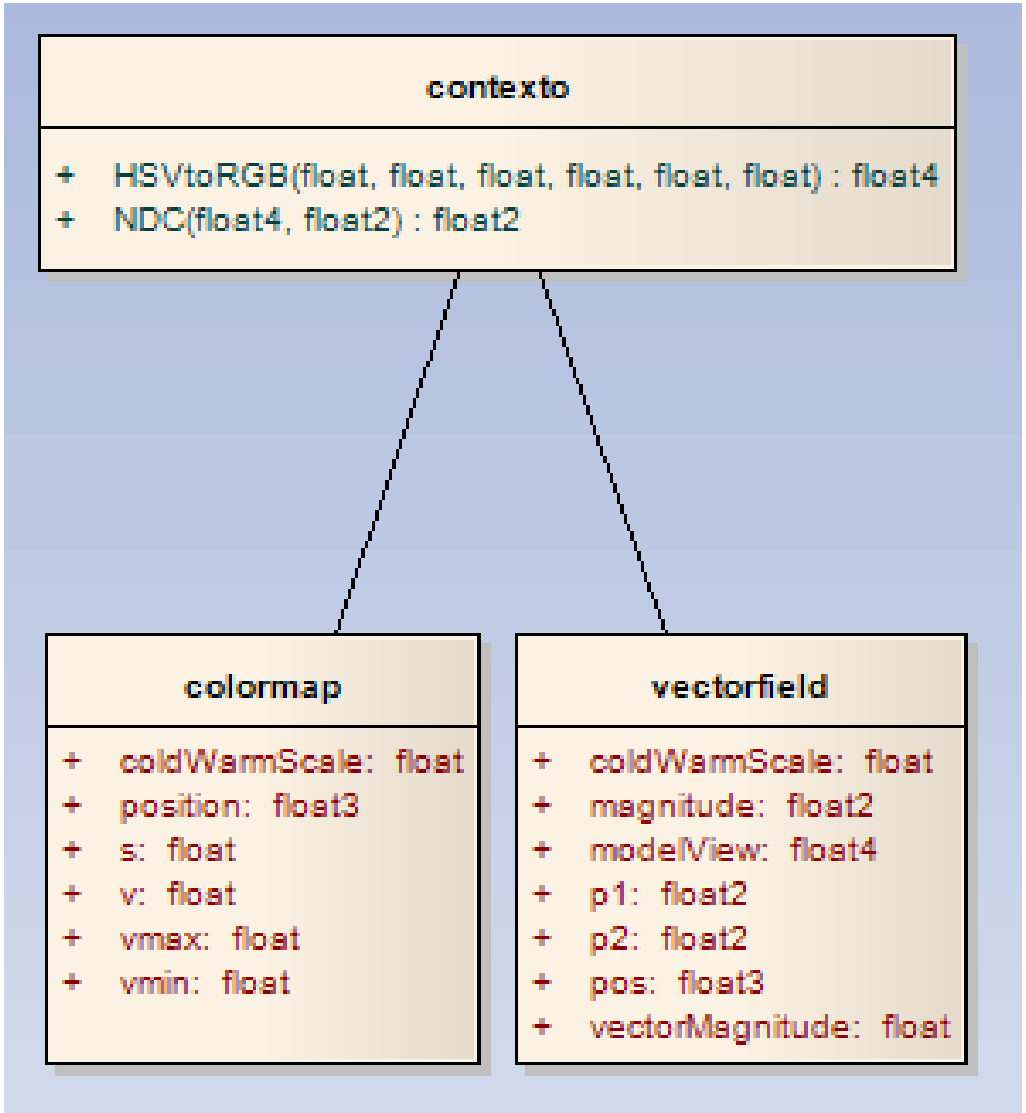
Trabalhos correlatos

- Processamento de imagens
  - Alteração de contraste, tentando preservar as cores originais
  - Desenvolvimento de nova técnica em CPU e GPU
  - GPU processa em menor tempo
- Visualização, manipulação de dados geométricos
  - Visualização da anatomia humana
  - Ferramentas para manipulação

Requisitos

- Permitir abrir arquivo com informações iniciais
- Permitir ao usuário iniciar ou parar as simulações
- Permitir ao usuário visualizar as simulações graficamente
- Ser implementado em C++
- Utilizar ambiente FX Composer e linguagem Cg

Especificação



- Diagrama de objetos para Cg
- Representação por efeito

Técnica colormap

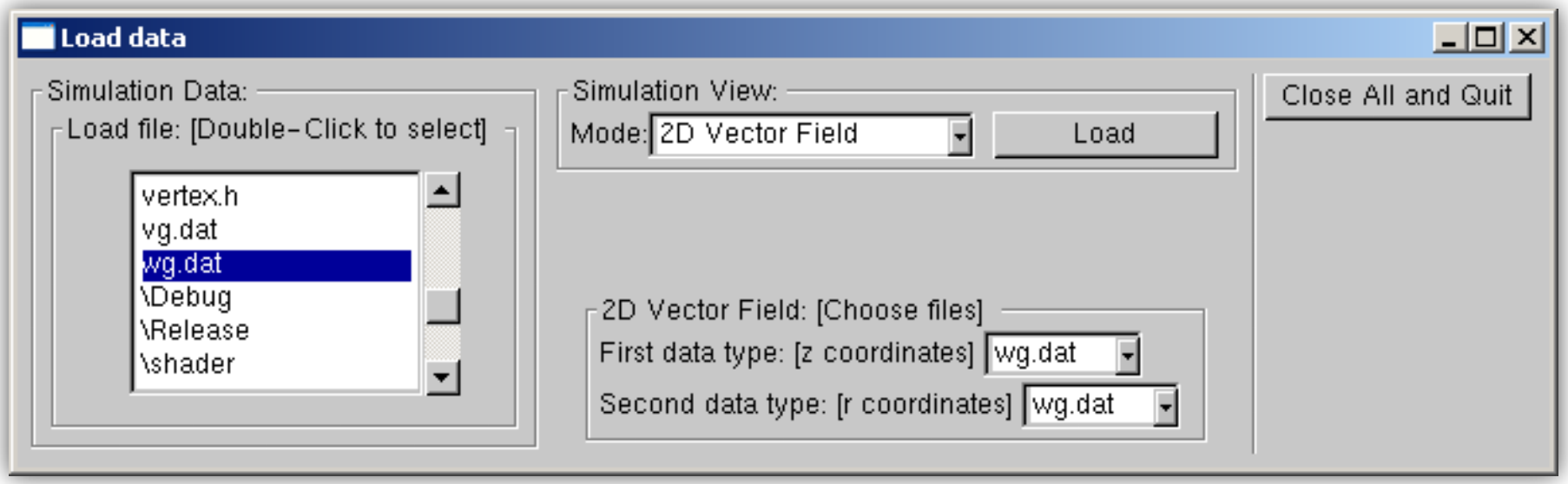
```
52    //função principal para o colormap
53    Ev_Output colormap (uniform float s,
54                        uniform float v,
55                        uniform float vmax,
56                        uniform float vmin,
57                        uniform float coldWarmScale ,
58                        uniform float4 modelView,
59                        float3 position: POSITION) {
60
61    Ev_Output OUT;
62
63    OUT.position = NDC(position.xy, modelView);
64    //a terceira dimensão é utilizada para armazenar o vertexvalue
65    OUT.color = HSVtoRGB(s, v, vmax, vmin, coldWarmScale, position.z);
66
67    return OUT;
68    }
```

Técnica

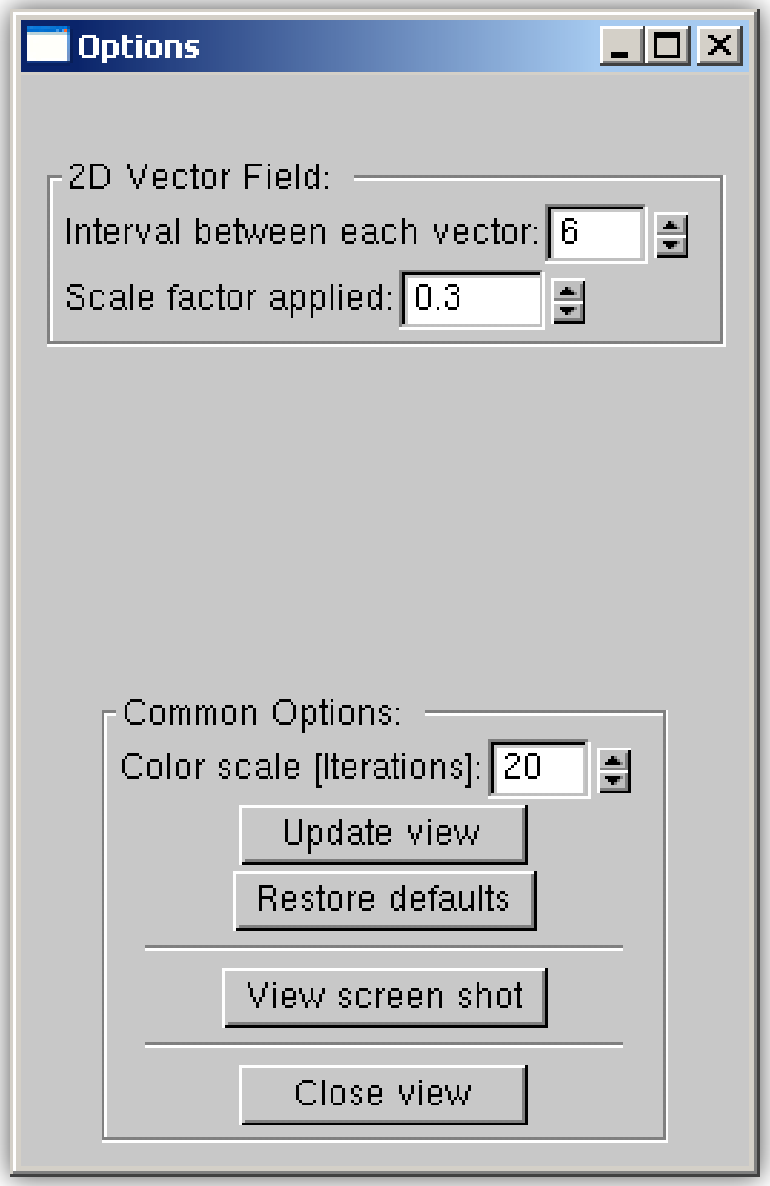
```
80      ...
81
82  if (pos.z == 1){
83
84      ...
85
86  } else if (pos.z == 2){
87     //parametric line equation
88     float2 pb;
89     pb.x = (1-t)*p1.x + t*p2.x;
90     pb.y = (1-t)*p1.y + t*p2.y;
91
92     //creates the normal vector
93     float2 originVector = p2.xy - p1;
94     float modulo = sqrt((originVector.x * originVector.x) +
95                         (originVector.y * originVector.y));
96
97     float2 normalVector = float2(-originVector.y, originVector.x);
98     normalVector.x /= modulo;
99     normalVector.y /= modulo;
100
101    float dx = sqrt((p2.x - pb.x) * (p2.x - pb.x) +
102                    (p2.y - pb.y) * (p2.y - pb.y)) / (3.0*0.67);
103
104    float2 palpha;
105    palpha.x = pb.x + dx*normalVector.x;
106    palpha.y = pb.y + dx*normalVector.y;
107    OUT.position = NDC(palpha, modelView);
108    OUT.color = HSVtoRGB(1.0, 1.0, magnitude.y, magnitude.x,
109                         coldWarmScale, vectorMagnitude);
110 }
111     ...
```

-Vector Field

Operacionalidade

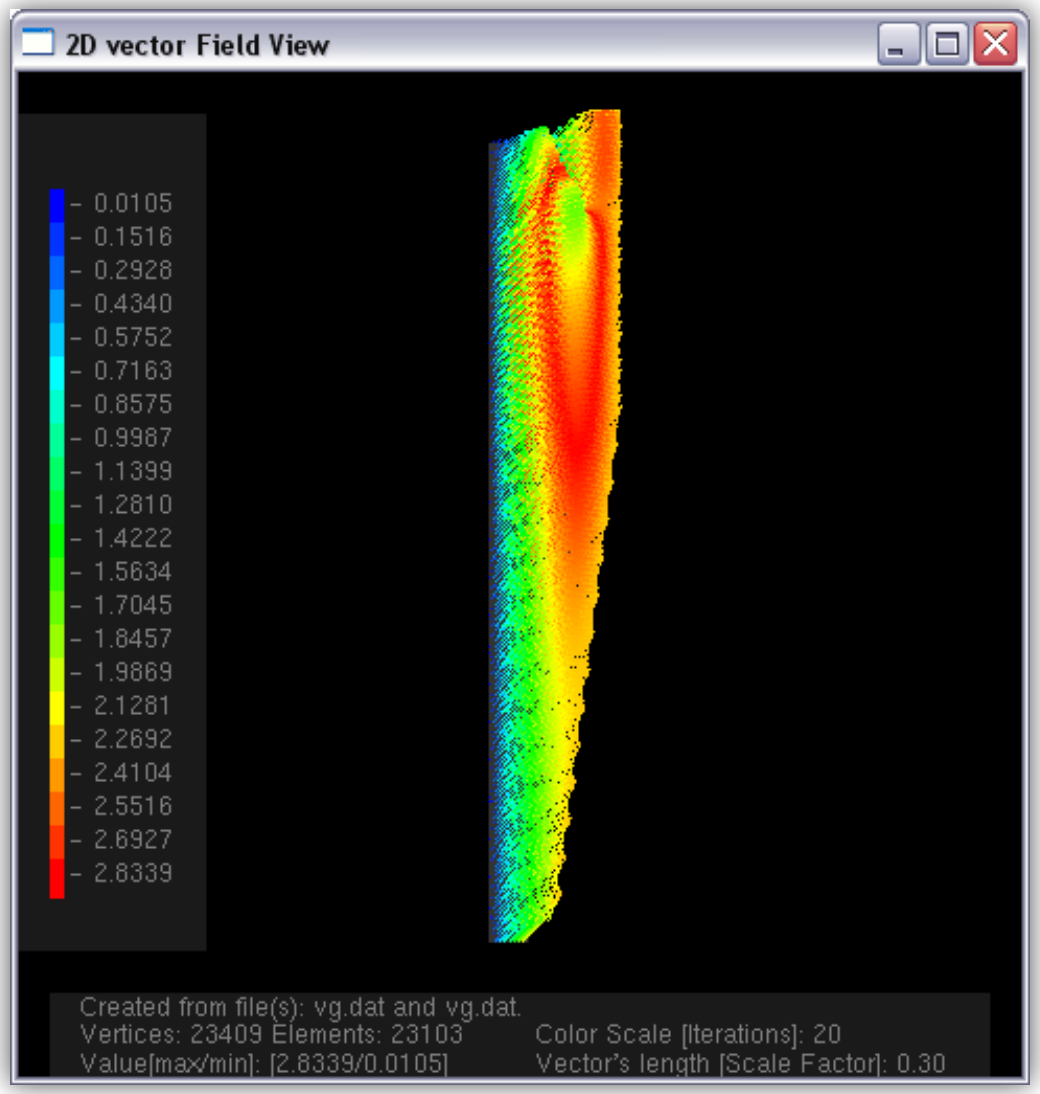


-Tela principal



-Tela opções

Operacionalidade



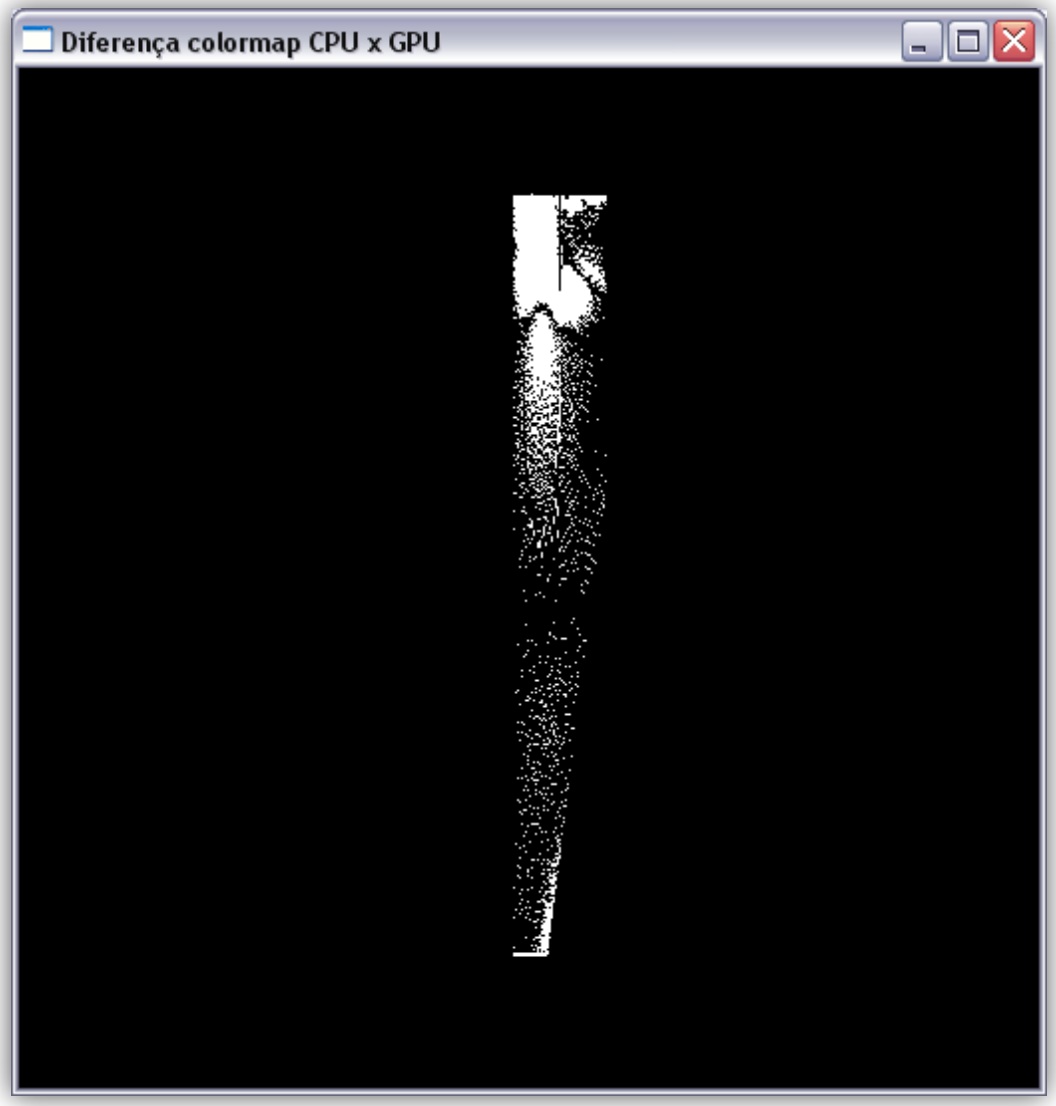
Operacionalidade

-Visualização

Resultados

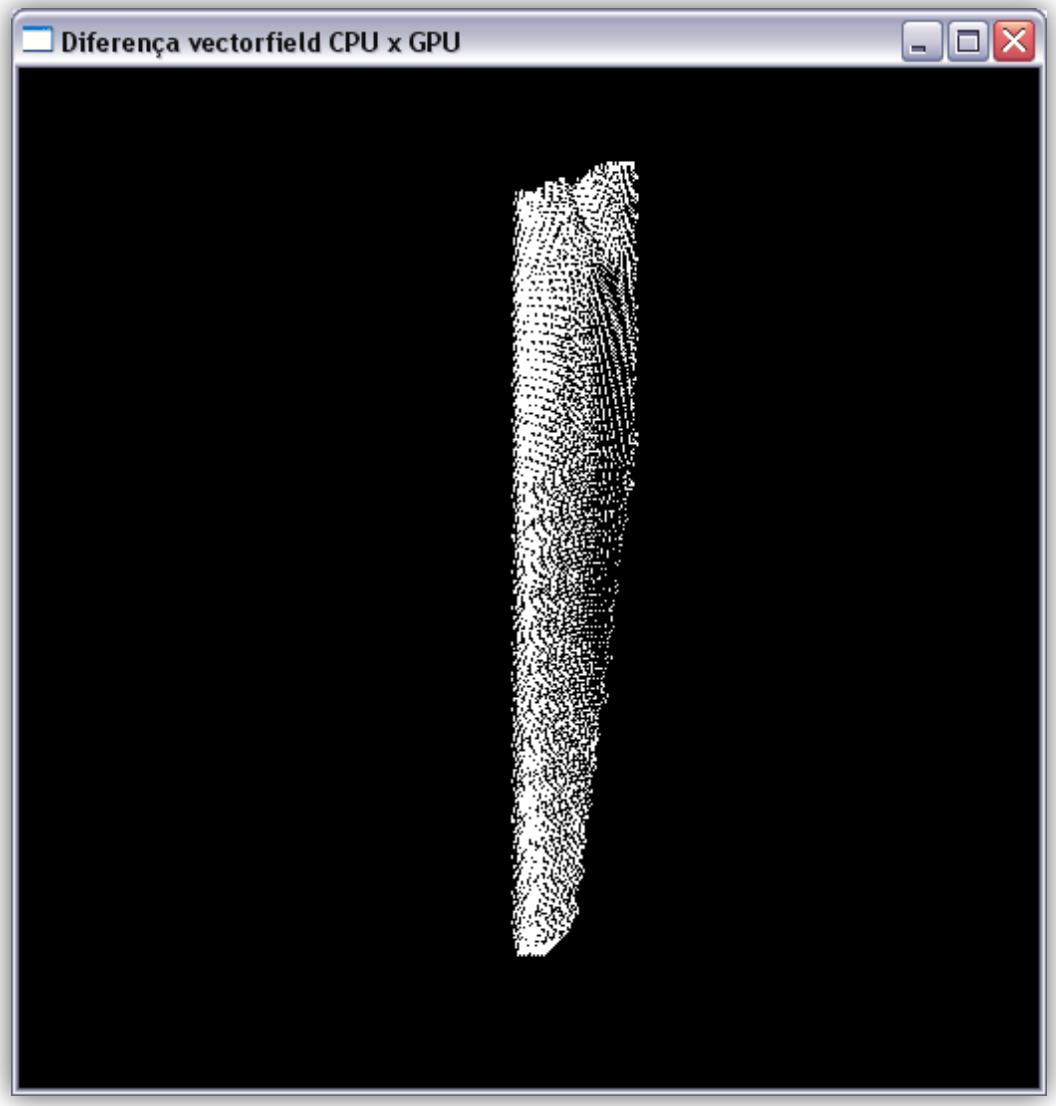
- Processamento
  - GPU mais eficaz
- Consumo de memória
  - Pouca diferença na maioria dos testes
- Representação gráfica
  - Diferenças imperceptíveis (colormap)
  - Vetor mantém cor constante na GPU (vectorfield)

Resultados



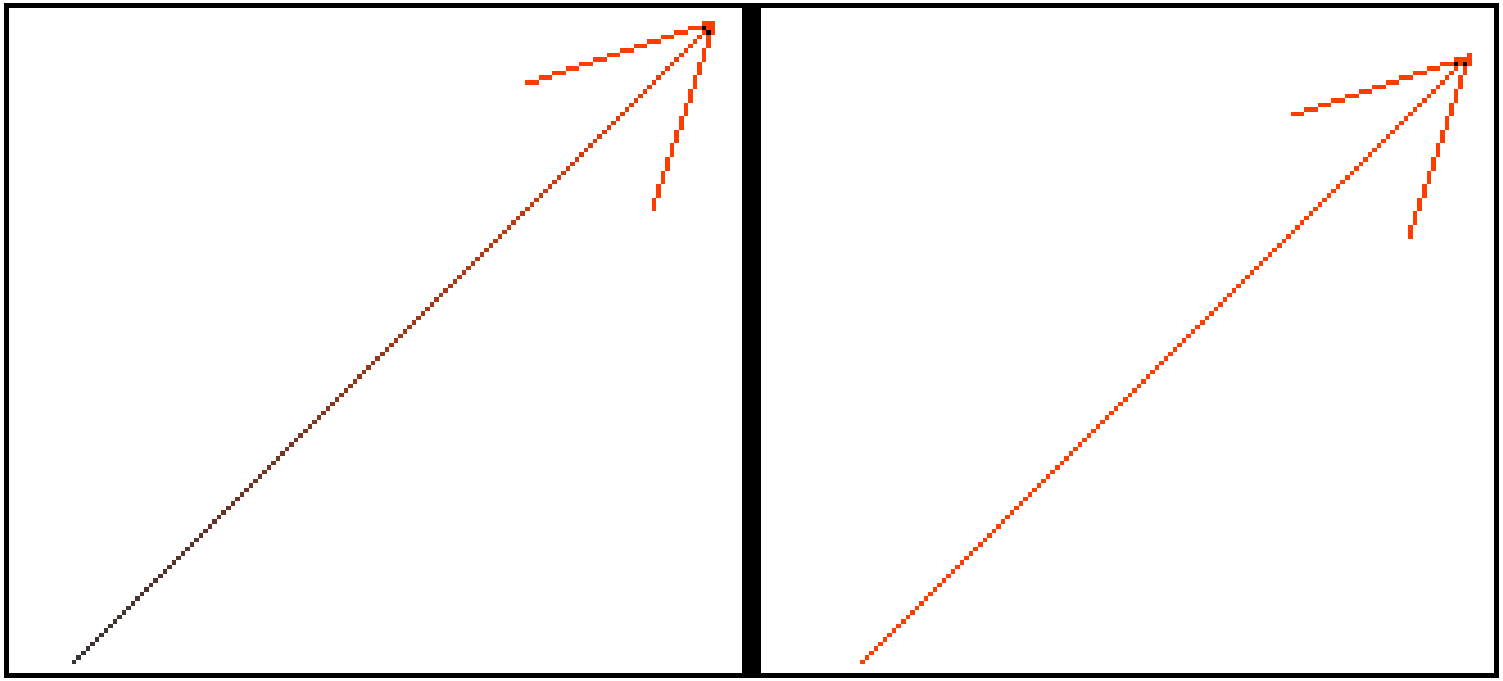
-Colormap

Resultados



-Vectorfield

Resultados - Vectorfield



CPU

GPU

Conclusão

- Pontos positivos
  - Maior processamento
  - Libera a CPU
- Negativos
  - Interação CPU x GPU
  - Maior complexidade

Extensões

- Melhorar técnica vectorfield
- Interação com os objetos em tempo real
- Implementar técnica 3D stream ribbons
- Desenvolver técnicas sobre o diagrama de objetos Cg
- Minimizar quantidade de pixels diferentes

FIM

Programa de comparação

```
1    //vertex shader
2    void C3E5v_twoTextures(float2 position : POSITION,
3                           float2 texCoord : TEXCOORD0,
4                           out float2 oPosition     : POSITION,
5                           out float2 oTexCoord  : TEXCOORD0)
6    {
7       oPosition = position;
8       oTexCoord = texCoord;
9    }
```

-Vertex shader

Programa de comparação

```
11    //fragment shader
12    void C3E6f_twoTextures(float2 TexCoord  : TEXCOORD0,
13                           out float4 color : COLOR,
14                           uniform sampler2D decalCPU,
15                           uniform sampler2D decalGPU)
16    {
17       float4 CPUcolor = tex2D(decalCPU, TexCoord);
18       float4 GPUcolor = tex2D(decalGPU, TexCoord);
19
20       color = abs(GPUcolor-CPUcolor);
21       if (color.r !=0.0 || color.g != 0.0 || color.b != 0.0)
22             color = float4(1.0, 1.0, 1.0, 1.0);
23    }
24
```

-Fragment shader