Uma implementação híbrida de raytracing em processadores gráficos programáveis ou processadores de propósito geral

Alessandro Ribeiro da Silva
Carlos Augusto Paiva da Silva Martins (Orientador)
Pontifícia Universidade Católica de Minas Gerais – PUC MINAS

Conteúdo

- Objetivos / Motivação
- Trabalhos relacionados
- Raytracing
- Pipeline programável e GLSL
- Raytracer híbrido
- Resultados
- Conclusão e trabalhos futuros

Objetivos / Motivação

- Implementação de um Raytracer híbrido no sentido de poder ser executado tanto no GPP quanto na GPU
  - Para a implementação na GPU utiliza a linguagem GLSL através da API OpenGL2.0

Objetivos / Motivação

- Raytracing gera imagens foto-realistas
- Raytracing é uma técnica que exige demanda computacional grande
- GPUs mais recentes apresentam alto poder de processamento com possibilidade de programação de parte de seu pipeline de renderização

Trabalhos Relacionados

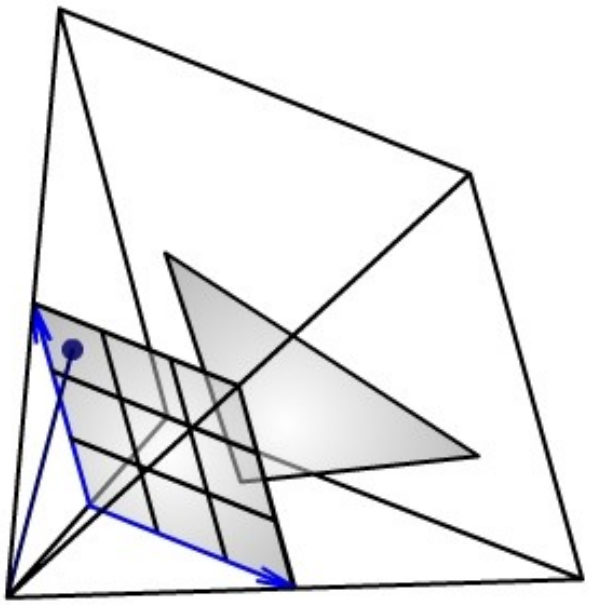
- Implementação de raytracers:
  - Integralmente na GPU
  - Parcialmente na GPU
  - Utilizando biblioteca Sh

- Este trabalho se baseia na implementação integral na GPU

Raytracing

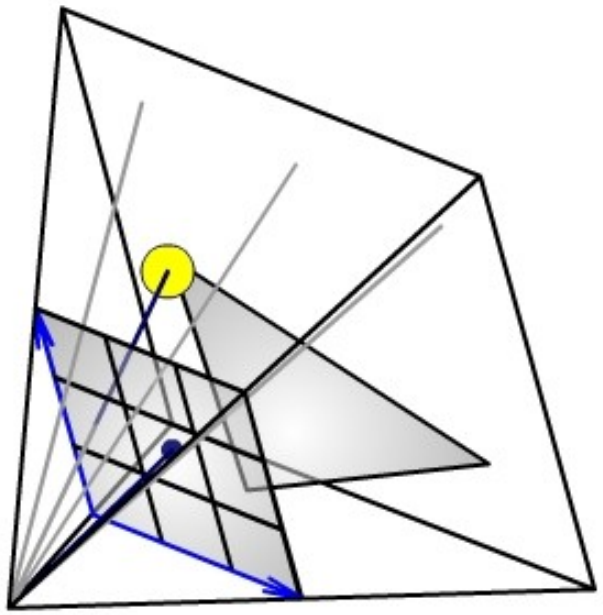
- Renderiza imagens foto-realistas
- Estuda o caminho que a luz percorre no espaço e as interações com objetos:
  - Reflexões
  - Refrações
  - transmissões
- Exige alta demanda de recursos computacionais

Raytracing



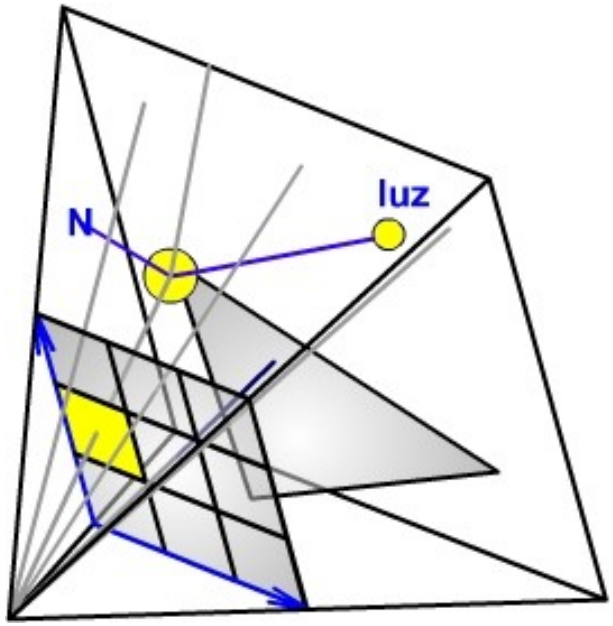
olho

Raytracing



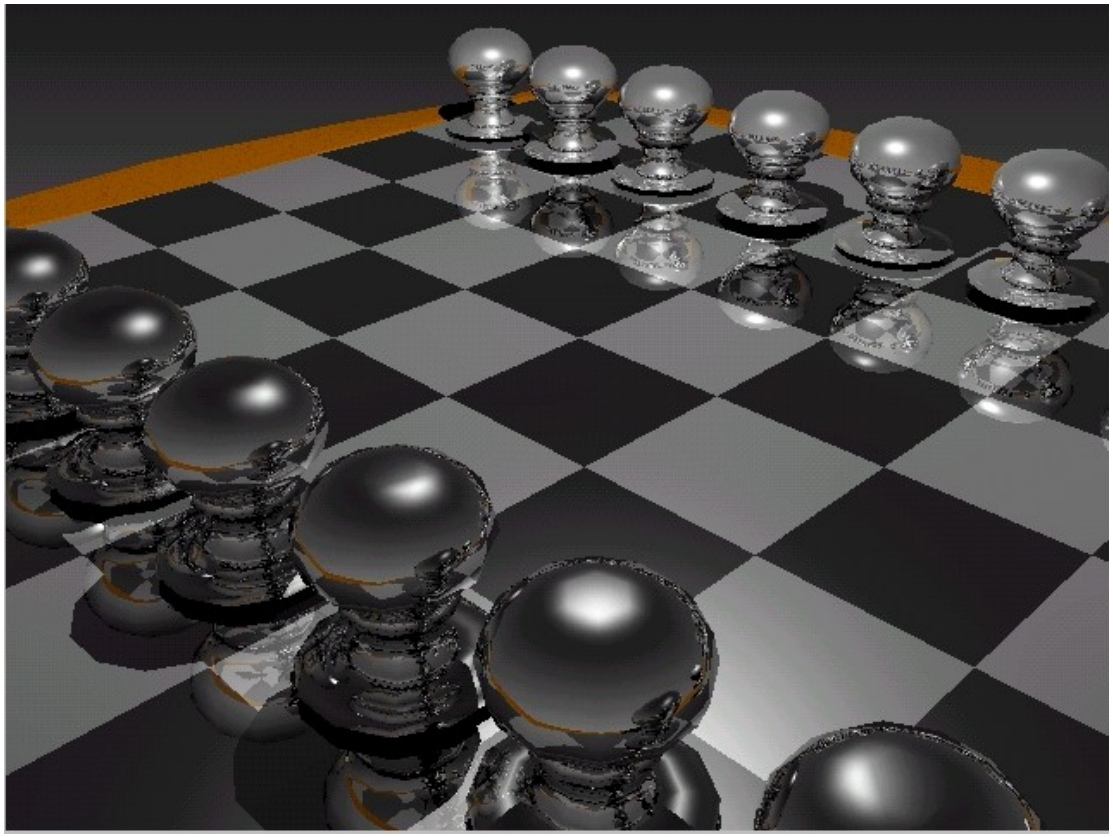
olho

Raytracing



olho

Raytracing – Exemplo



Renderização com base na técnica de rayracing no modelador Blender (www.blender.org)

Pipeline programável e GLSL

- GPUs mais recentes permitem a programação dos estágios:
  - Processamento de vértices
  - Processamento de fragmentos
- A GPU programável funciona como um processador SIMD

Pipeline programável e GLSL

- Linguagens mais comuns:
  - HLSL (High Language Shading Language)
  - GLSL (OpenGL Shading Language)
  - Cg (“C” for Graphics)
- Biblioteca Sh (Embedded Metaprograming Language)

Pipeline programável e GLSL

- GLSL
  - Pode se escrever um programa para cada estágio programável (vértice e fragmento)
  - Presente na API OpenGL 2.0
  - Portável
  - Compilação em tempo de execução
  - Não exige exclusividade de hardware
  - Linguagem de alto nível

Pipeline programável e GLSL

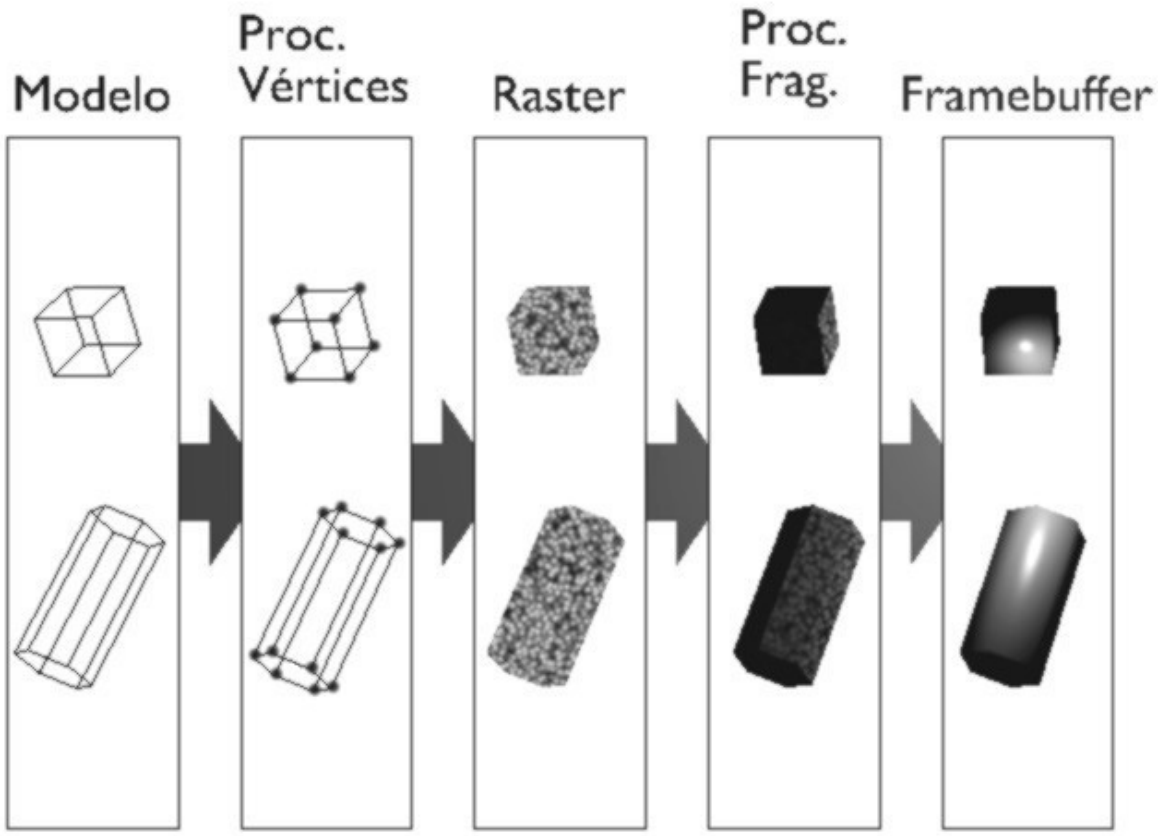


Diagrama de um pipeline de renderização

Raytracer híbrido

- Implementado para possibilitar a execução do raytracer tanto na GPU quanto na CPU
- Utiliza da linguagem GLSL
- Dividido em 3 etapas:
  - Geração de raios
  - Teste de interseções
  - Tonalização

Raytracer híbrido – Diagrama GPU

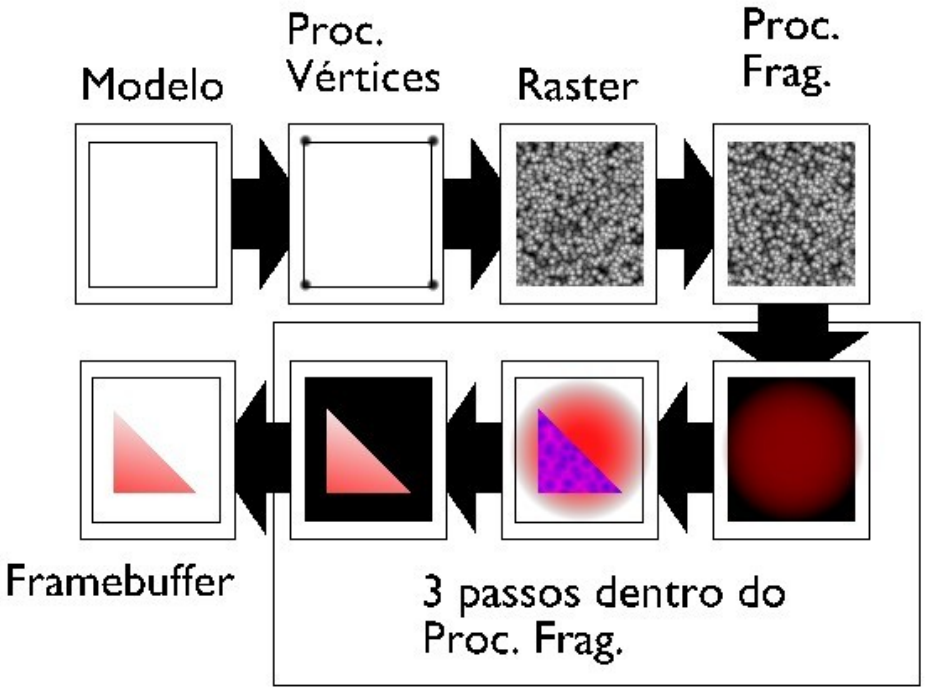


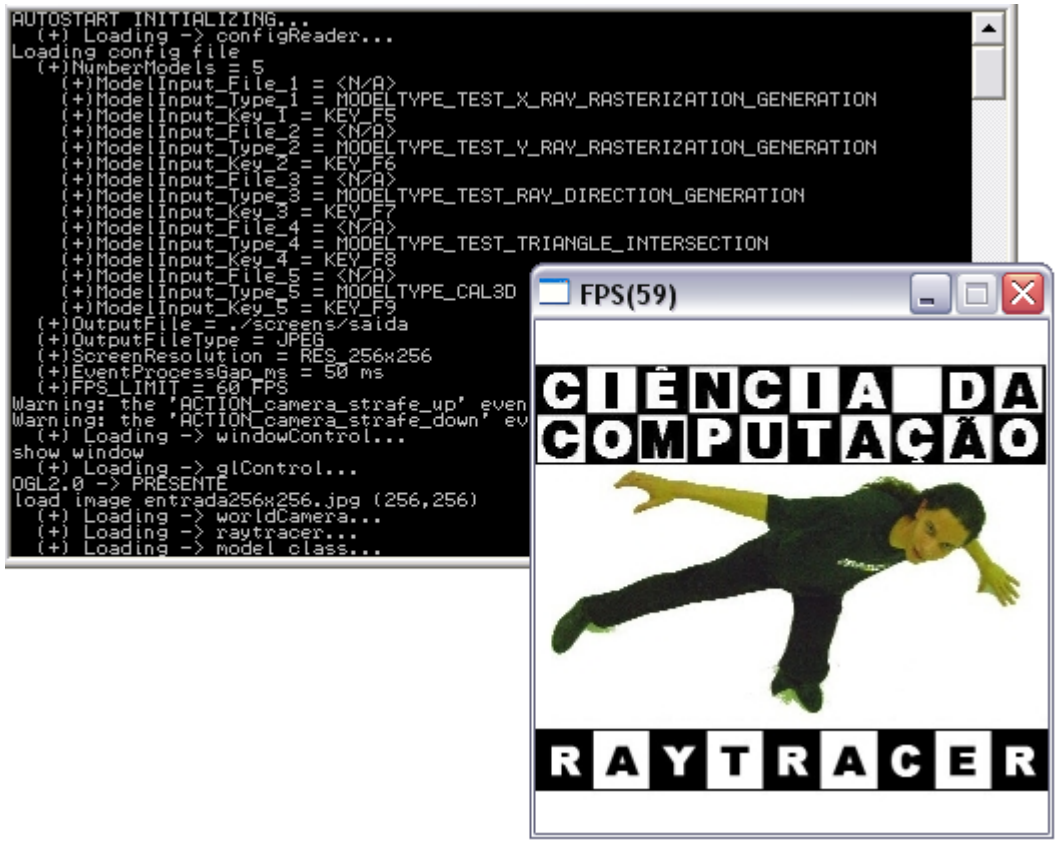
Diagrama do Raytracer dentro do pipeline da GPU

(O raytracer da GPP possui os mesmos passos exceto os estágios do pipeline da GPU)

Raytracer híbrido – Shader Principal

```
void main(){
   vec3 finalColor, n0, n1, n2;
   property material;
   ray currentRay = getCurrentRay();
   colisionResult colision = getColisionResult(
                                 currentRay, n0, n1, n2, material );
   if ( colision.L == INFINITE )
        discard; //background
   finalColor = shade( colision, currentRay,
                               n0, n1, n2, material );
   gl_FragColor = vec4( finalColor, 1.0 );
}
```

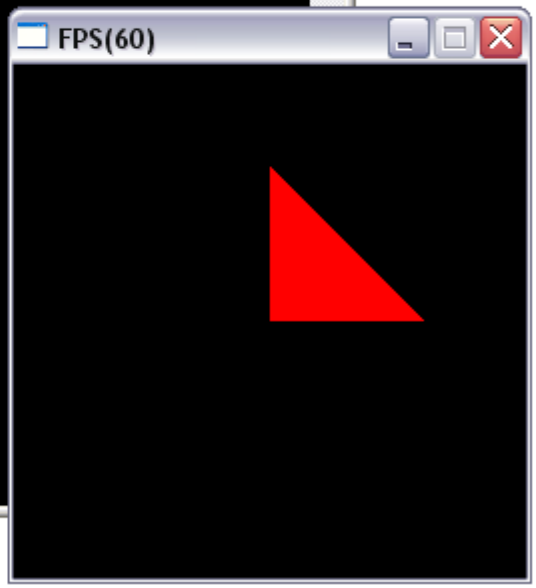
Raytracer híbrido – Tela Inicial



Raytracer híbrido – RenderizaçãoGPU

```
raytracer_Updated
setando GLSL inf!!
Gerando Arquivo ./rayShaders/StaticConfig.txt...
 ++++++++ Setando Numero de Triangulos
 ++++++++ Setando Numero de Vertices
Abrindo arquivo "./rayShaders/structs.txt       " ... OK.
Abrindo arquivo "./rayShaders/StaticConfig.txt  " ... OK.
Abrindo arquivo "./rayShaders/rayModule.txt     " ... OK.
Abrindo arquivo "./rayShaders/triangleModule.txt" ... OK.
Abrindo arquivo "./rayShaders/colisionModule.txt" ... OK.
Abrindo arquivo "./rayShaders/shadingModule.txt " ... OK.
Abrindo arquivo "./rayShaders/main.txt          " ... OK.
  (+)Compilando um Objeto 'Fragment Shader' no OpenGL ... OK
Abrindo arquivo "./rayShaders/VertexShadersMain.txt" ... OK.
 (+) Compilando: OK
  (+) Linkando O Programa Final ... OK
PROGRAMA FINAL PRONTO PARA SER UTILIZADO!!!
SETOU Vert DATA!!!!!
RenderMode changed!!! XD
raytracer_Updated
E a primeira vez ou o prorgama dee shader foi carregado!!!

CARREGANDO NOVAS INF SOBRE A CAMERA
Copiando backBuffer To texture!!!
```



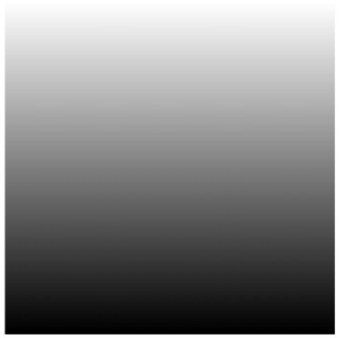
Resultados

- Foram feitos cinco testes
  - Duas rasterizações
  - Três renderizações que envolviam etapas do raytracing

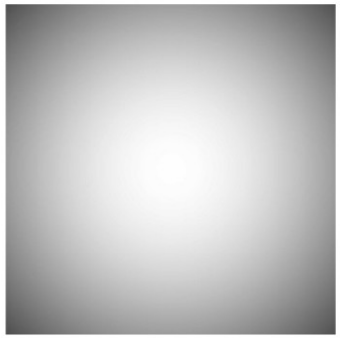
- Os processadores testados foram:
  - GPP1 – Pentium III 800 - 256Mb RAM
  - GPP2 – Pentium IV 2.4 Ghz – 512Mb RAM
  - GPU1 – GeForce 5200 – 128Mb RAM
  - GPU2 – GeForce 6600 – 128Mb RAM

Resultados – Rasterizações

F5



F6

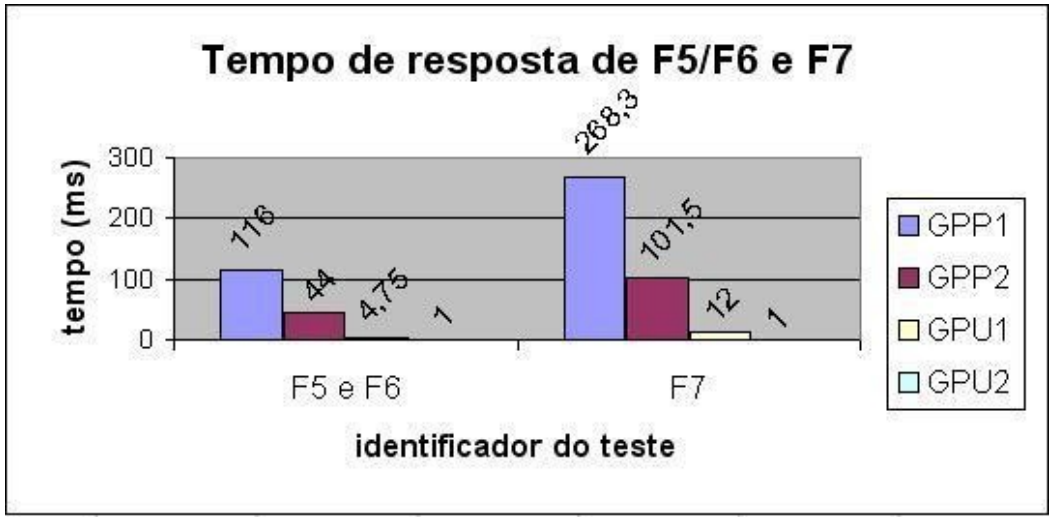


F7

(F5/F6) Rasterizações

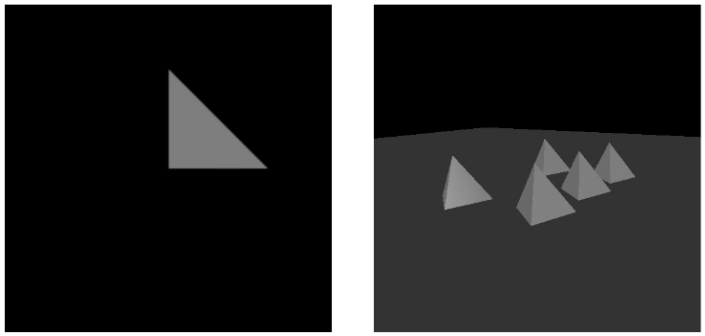
(F7) Geração de raios

Resultados – Rasterizações



Teste	Speedup em relação a GPP1		
	GPP2	GPU1	GPU2
F5/F6	2.6	24.4	116
F7	2.6	22.3	268.3

Resultados – Renderizações

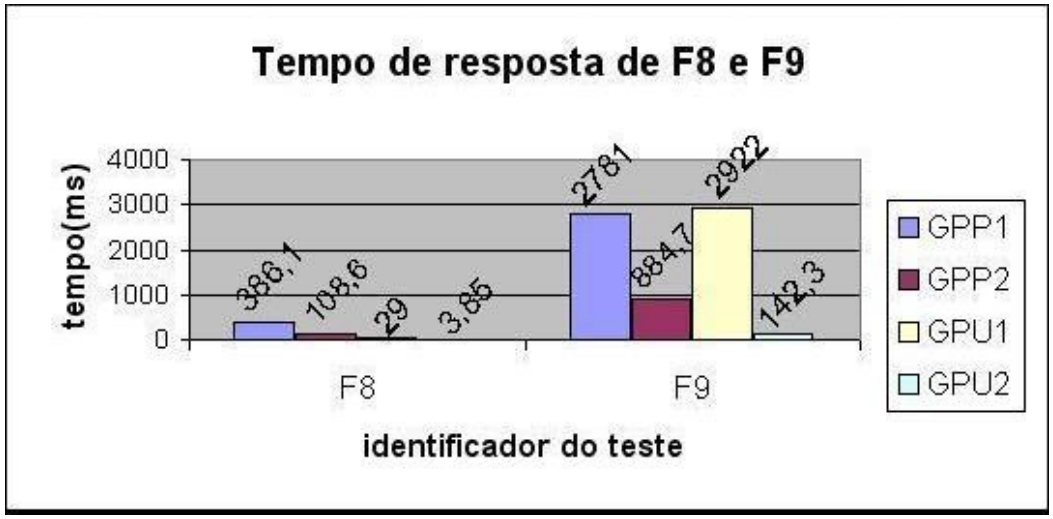


F8 F9

(F8) Interseção com um triângulo

(F9) Renderização com iluminação

Resultados – Rasterizações



Teste	Speedup em relação a GPP1		
	GPP2	GPU1	GPU2
F8	3.5	13.3	100.2
F9	3.1	0.95	19.5

Conclusão e trabalhos futuros

- Este trabalho apresenta os resultados parciais da implementação de um raytracer que funciona tanto em GPP quanto em GPU
- Na maioria dos testes as GPUs superaram a velicidade das CPUs
  - Exceto no teste F9 com a GeForce5200, onde o speedup foi de 0.95
- O software implementado utiliza a linguagem GLSL para a programação de GPU

Conclusão e trabalhos futuros

- Continuações que podem ser dadas a esta pesquisa:
  - Estudar a possibilidade de implementação do raytracer baseado em mais de um passo para a renderização
  - Efetuar mais testes que envolvam a complexidade da cena (resolução e modelo)
  - Adaptar esta implementação para que possa realizar o calculo de reflexões, sombras e textura
  - Utilizar mais de uma GPU unidas por algum meio de comunicação
  - Implementar alguma técnica de suavização de cena baseado em superamostragem
  - Implementar e testar este raytracer em outras GPUs do mesmo e de outros fabricantes

FIM