Centro de Informática – João Pessoa
Disciplina: Computação Gráfica
Klismann de Oliveira Barros - 20200085284
Thaís Gabrielly Marques - 20180135293

ATIVIDADE PRÁTICA 5 - RAY TRACING

Descrição:

Nesta atividade cinco, foi pedido que fizéssemos dois exercícios de extensões ao ray tracer, com objetivo de nos familiarizarmos com essa técnica de geração de imagens.

Estratégias adotadas:

Primeiramente revisamos o conteúdo disponibilizado pelo professor e realizamos algumas pesquisas para começar a sanar as principais dúvidas. Logo após, passamos para o template disponível com os primeiros redirecionamentos, também fornecidos pelo professor.

Exercício 1: Inclusão de termo specular no modelo de iluminação local do ray tracer.

Durante esse primeiro exercício teríamos que completar a equação do modelo de iluminação de Phong e integrá-la ao ray tracer, pois no template disponível já havia o cálculo dos termos ambiente e difusos. Abaixo a equação original com o termo pedido:

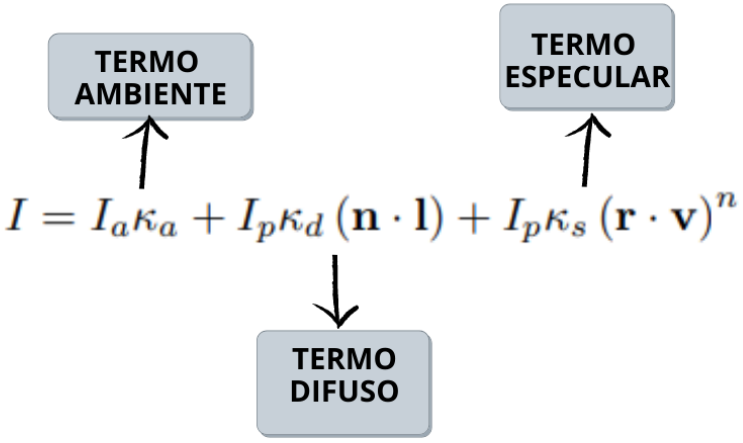


Imagem 01 - equação completa

Tínhamos relido algumas considerações da atividade três, porém adaptar para o ray tracer por um momento se tornou um problema considerável pela forma que é executado, além de designar o ks, seguindo a implementação dos primeiro termos, desenvolvemos o termo especular. Trecho do código:

```
        let r = L.clone().reflect(interseccao.normal);

        let v = interseccao.posicao.normalize();

        let termo_specular = Ip.cor.clone().multiply(ks).multiplyScalar((Math.max(0.0,
r.dot(v)))**32)
```

Tivemos que adicionar toda a constituição do termo especular, para enfim chegar ao resultado.

Exercício 2: Inclusão de suporte a rendering de triângulos.

O ray tracer disponibilizado pelo professor apenas trabalhava com esferas, então foi incumbido a nós incluir triângulos. Começamos analisando as sugestões e durante os testes escolhemos a segunda sugestão dada: o algoritmo de Möller e Trumbore(Fast, Minimum Storage Ray/Triangle Intersection), pelo motivo de maior compatibilidade com o código do template, facilidade de leitura e adaptação por nós. Trecho do código implementado:

```
class Triangle {
  constructor(v1,v2, v3) {
    this.v1 = v1;
    this.v2 = v2;
    this.v3 = v3;
  }
////////////////////////////////////////////////////////////////////////
////////
   // Metodo que testa a interseccao entre o raio e o triangulo,
adaptando o algoritmo de Möller e Trumbore.

////////////////////////////////////////////////////////////////////////
////////

  interseccionar(raio, interseccao) {

    let edge1 = this.v2.clone().sub(this.v1);
    let edge2 = this.v3.clone().sub(this.v1);
    let h = raio.direcao.clone().cross(edge2);
    let a = edge1.clone().dot(h);

    if (Math.abs(a) < 0.001) // Este raio é paralelo a este triângulo.
      return false;

    let f = 1.0 / a;

    let s = raio.origem.clone().sub(this.v1);
    let u = s.clone().dot(h) * f;

    if (u < 0.0 || u > 1.0)
      return false;

    let q = s.clone().cross(edge1);
    let v = raio.direcao.clone().dot(q) * f;

    if (v < 0.0 || u + v > 1.0)
      return false;

     interseccao.t = edge2.clone().dot(q) * f; // Aqui podemos calcular
t para descobrir onde está o ponto de interseção na linha.

    let w0 = this.v1.clone().multiplyScalar(1 - u - v);
```

```
    let u1 = this.v2.clone().multiplyScalar(u);
    let v2 = this.v3.clone().multiplyScalar(v);
    interseccao.posicao = w0.add(u1).add(v2);

     // Aqui acontece o calculo do vetor normal de interseccao com o
triangulo.
    let p2 = this.v3.clone().sub(interseccao.posicao);
    let p1 = this.v2.clone().sub(interseccao.posicao)
    interseccao.normal = p2.cross(p1).normalize();

    return true;
  }
}
```

Obs: Apesar dos códigos(exercícios) estarem separados, os dois podem funcionar juntos, basta incluir suas configurações na mesma função Render.

Algumas falhas durante o processo:

No começo do desenvolvimento do primeiro exercício ocorreu um erro na combinação dos termos no PutPixel, por causa de um mau cálculo no termo especular.

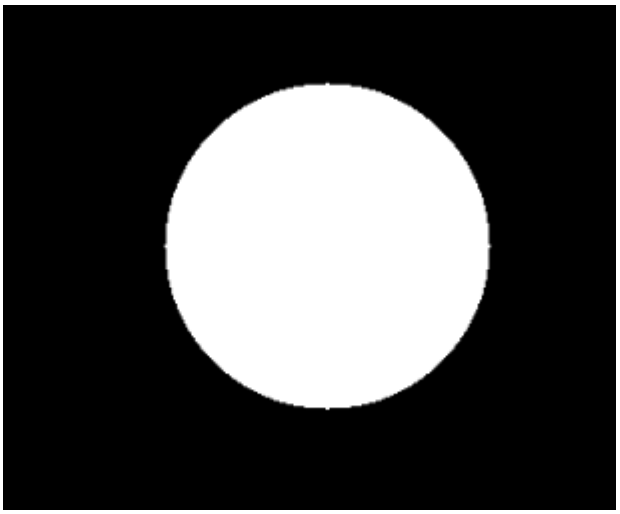
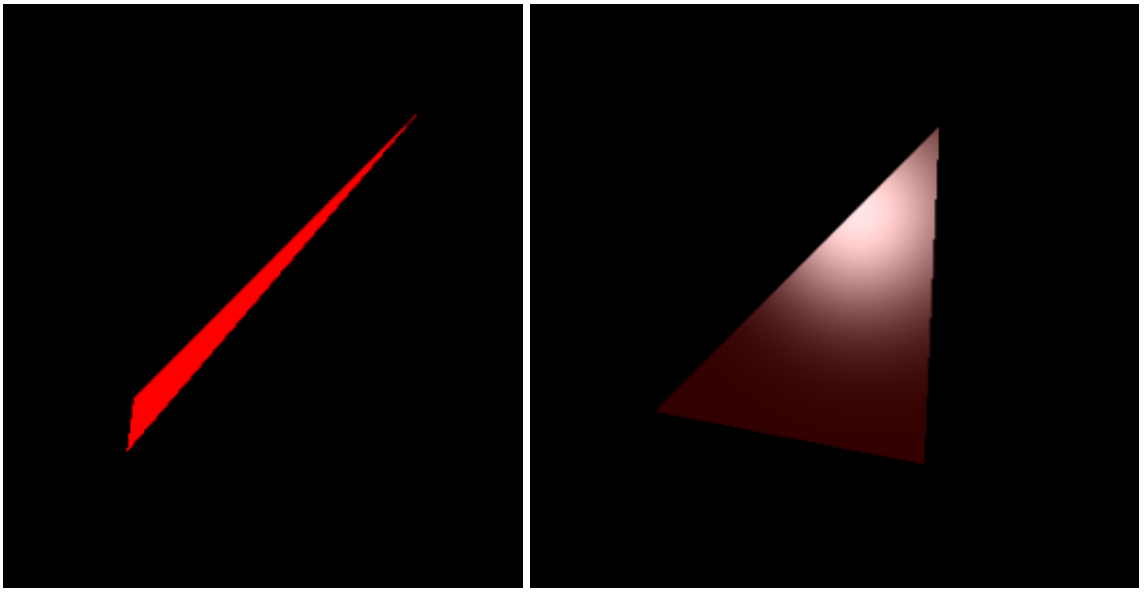


Imagem 02

Durante o desenvolvimento do segundo exercício passamos por alguns erros:



Imagens 03 e 04

Na imagem 03 aconteceu o bug abordado no vetor (-0.75, -1.0, -2.5), que foi corrigido por 0.75. Já na imagem 04 ocorreu um erro no cálculo da intersecção e por fim tivemos que alterar o segundo vetor de Ip para 0.9, assim podendo atingir o resultado esperado.

Resultados gerados:

Exercício 1:

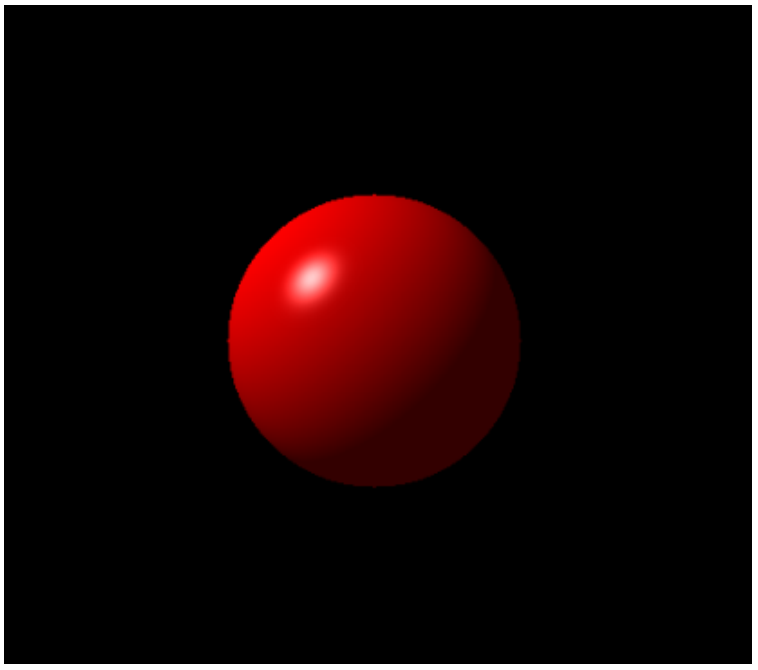


Imagem 05 - Resultado com equação completa.

Exercício 2:

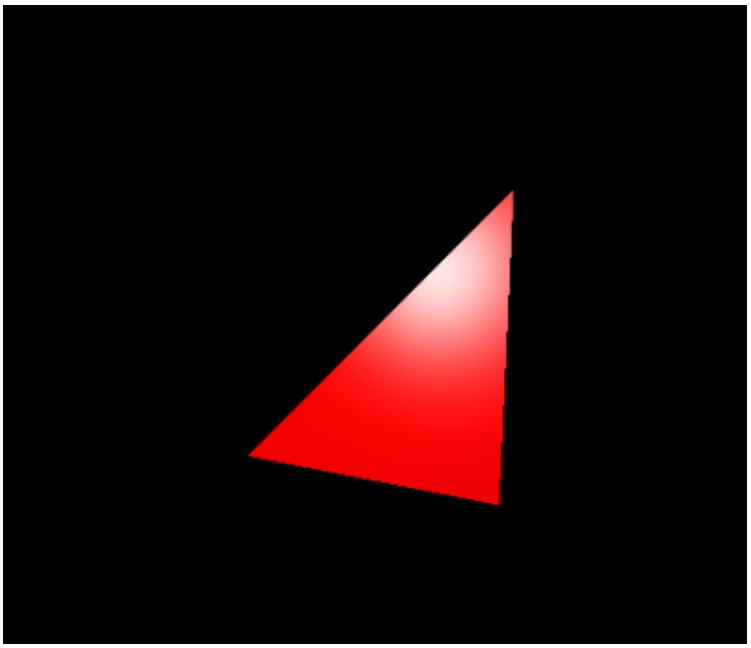


Imagem 06 - Resultado da inclusão do triângulo

Dificuldades e possíveis melhorias:

Tivemos bastante dificuldade em realizar os exercícios, no primeiro a maior tormenta foi desenvolver tudo relacionado ao specular do jeito que se pedia no código, pois do jeito que estávamos fazendo por mais certo que estivesse era incompatível ao template. No segundo a parte da função interseccionar deu problemas, mas com pesquisas e consulta ao material disponível conseguimos resolver. Uma possível melhoria seria renderizar outras formas geométricas e até mesmo cenários completos com e sem animação.

Referências bibliográficas:

- Notas e aulas do professor.
- http://index-of.es/Computer/Fundamentals%20of%20Computer%20Graphics%20--%20Peter%20Shirley.pdf (Seção 10.3.2).
- https://www.gsigma.ufsc.br/~popov/aulas/cg/plano/index.html
- https://cadxfem.org/inf/Fast%20MinimumStorage%20RayTriangle%20Intersection.pdf (pág. 4-5).
- https://en.wikipedia.org/wiki/M%C3%B6ller%E2%80%93Trumbore_intersection_algorithm (Java implementation).

Link repositório:

- https://codepen.io/gabriellymarques02/pen/poPjJrN (Exercício 1).
- https://codepen.io/Klismann_Barros/pen/XWRXVpL (Exercício 2).