Aluno: Renilton Ribeiro de Oliveira Matrícula: 11328395

Atividade 5

Introdução

Esse é um relatório sobre a quinta atividade da cadeira de computação gráfica, onde na primeira parte devemos fazer a implementação do termo especular no modelo de iluminação local do ray tracer, e na segunda parte devemos fazer a implementação do suporte ao rendering de triângulos. O objetivo deste trabalho é familiarizar os alunos com as técnicas de geração de imagens baseadas em ray tracing, essa disciplina é ministrada pelo professor Christian Pagot, na UFPB, no período 2020.2. Para esse trabalho foi utilizada a linguagem de programação Javascript juntamente com o framework disponibilizado pelo professor, o programa template que renderiza uma esfera vermelha iluminada com um modelo local de iluminação composto por um termo ambiente e por um difuso, como ilustrado na Figura 1.

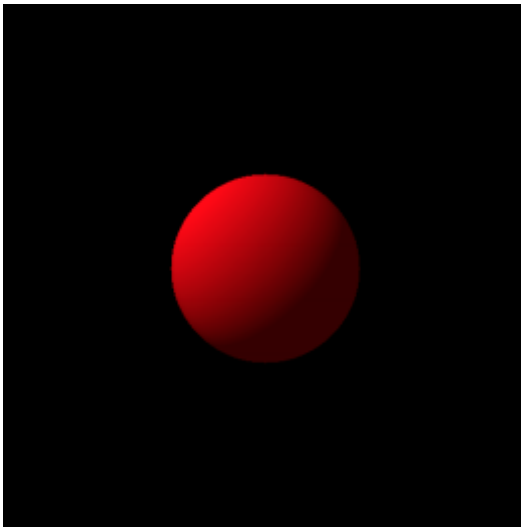

figura 1. Esfera vermelha iluminada através de um modelo local de iluminação que considera apenas os termos ambiente e difuso.

Estratégia

Na primeira parte do trabalho usei basicamente a mesma implementação utilizada na atividade 3, só foi preciso fazer algumas modificações na forma de utilizar algumas funções, como por exemplo a função reflect(), já na segunda parte optamos pela utilização do método de Fast, Minimum Storage Ray/Triangle Intersection (Möller, T.; Trumbore, B.) apresentado em um dos materiais passado pelo professor, este é um dos métodos mais rápidos para o cálculo de intersecção raio/triângulo, e se destaca também pelo baixo consumo de memória. É o método recomendado no renderizador Mitsuba (versão 0.6) no caso do rendering de malhas com muitos triângulos. O artigo referente a este método está disponível nas referências deste relatório.

Resultado - Parte 1.

A primeira parte consistia em implementar o termo especular no modelo de iluminação local do ray tracer, para isso foi necessário adicionar ao código na linha 171 o vetor com o coeficiente de reflectância especular da esfera, informado pelo professor, na linha 177 foi adicionado o vetor R, que é o vetor que representa a reflexão do vetor L sobre N(vetor normal), na linha 178 foi adicionado o vetor V, que representa o vetor que aponta para câmera, feito isso se tem os vetores necessários para o cálculo do termo especular, esse adicionado na linha 184, representado por $Ip*Ks\ (r \cdot v)^{n}$, onde o valor de n=32 foi dado pelo professor.

```
168
169         let ka = new THREE.Vector3(1.0, 0.0, 0.0);  // Coeficiente de reflectancia ambiente da esfera.
170         let kd = new THREE.Vector3(1.0, 0.0, 0.0);  // Coeficiente de reflectancia difusa da esfera.
171         let ks = new THREE.Vector3(1.0, 1.0, 1.0);  // Coeficiente de reflectancia especular da esfera.
172         let Ia = new THREE.Vector3(0.2, 0.2, 0.2);  // Intensidade da luz ambiente.
173
174         let termo_ambiente = Ia.clone().multiply(ka); // Calculo do termo ambiente do modelo local de iluminacao.
175
176         let L = (Ip.posicao.clone().sub(interseccao.posicao)).normalize(); // Vetor que aponta para a fonte e luz pontual.
177         let R = L.clone().reflect(interseccao.normal); //Vetor que representa a reflexão de L sobre N.
178         let V = (interseccao.posicao).normalize(); // Vetor que aponta para a câmera
179
180         // Calculo do termo difuso do modelo local de iluminacao.
181         let termo_difuso = (Ip.cor.clone().multiply(kd)).multiplyScalar(Math.max(0.0, interseccao.normal.dot(L)));
182
183         // Calculo do termo especular do modelo local de iluminacao.
184         let termo_especular = (Ip.cor.clone().multiply(ks)).multiplyScalar(Math.max(0.0, R.dot(V))**32);
185
186         // Combina os termos difuso, ambiente e especular e pinta o pixel.
187         PutPixel(x, y, termo_difuso.add(termo_ambiente).add(termo_especular));
```

figura 2. Implementando o modelo de iluminação Phong.

Após isso temos os três termos(ambiente, difuso e especular) necessários para implementar o modelo de iluminação Phong, modelo esse implementado na linha 187, nos dando o resultado abaixo na figura 3.

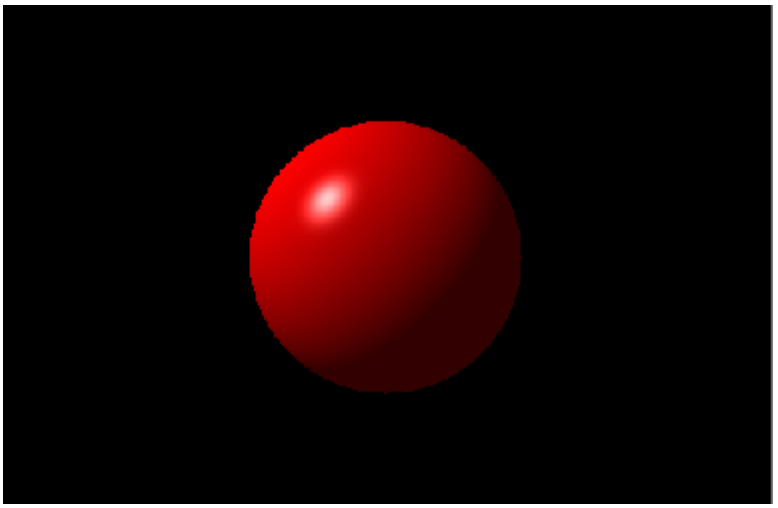

figura 3. Resultado da inclusão do termo especular no modelo de iluminação local do ray tracer.

Resultado - Parte 2.

Após a implementação da primeira parte tive que fazer algumas modificações para a Inclusão do suporte ao rendering de triângulos, como a criação do constructor com os vértices do triângulo na figura 4, onde o v1, v2 e v3 eram representados respectivamente por (−1, −1, −3.5), (1, 1, −3) e (0.75, −1, −2.5) dados pelo professor na linha 165.

```
106   interseccionar(raio, interseccao) {
107     let v1 = this.v1;
108     let v2 = this.v2;
109     let v3 = this.v3;
110
111     let v1v2 = v2.clone().sub(v1);
112     let v3v1 = v3.clone().sub(v1);
113
114     let invDet = 1.0 / v1v2.clone().dot(raio.direcao.clone().cross(v3v1));
115
116     let tvec = raio.origem.clone().sub(v1);
117     let qvec = tvec.clone().cross(v1v2);
118
119     // Calcula os parâmentro "u" e "v"
120     let u = tvec.clone().dot(raio.direcao.clone().cross(v3v1)) * invDet;
121     let v = raio.direcao.clone().dot(qvec) * invDet;
122
123     // Testa os resultado e "u" e "v"
124     if (u < 0.0 || u > 1.0)
125       return false;
126
127     if (v < 0.0 || u + v > 1.0)
128       return false;
129
130     // Distancia entre o ponto P de interseccao e a origem do raio.
131     interseccao.t = v3v1.clone().dot(qvec) * invDet;
132
133     // Ponto P de interseccao.
134     interseccao.posicao = v1.clone().multiplyScalar(1 - u - v).add(v2.clone().multiplyScalar(u)).add(v3.clone().multiplyScalar(v));
135
136     // Ponto de interseccao do raio com o triangulo.
137     interseccao.normal = v3.clone().sub(interseccao.posicao).cross(v2.clone().sub(interseccao.posicao)).normalize();
138
139     return true;
140   }
141 }
```

Figura 4. Implementação da parte 2.

Das linhas 111 a 114 criamos as bordas do triângulo, das linhas 116 a 117 definimos o ponto do raio, das linhas 120 a 128 calculamos e testamos os valores de "u" e "v" da equação na figura 5, já na linha 134 calculamos o ponto p de intersecção, com a equação da figura 5 retirada do artigo “Fast Minimum Storage Ray” nas referências, e finalmente na linha 137 calculamos o ponto de intersecção do raio com o triângulo.

$$T(u,v) = (1-u-v)V_0 + uV_1 + vV_2,$$

figura 5. equação.

Abaixo podemos observar o resultado obtido.

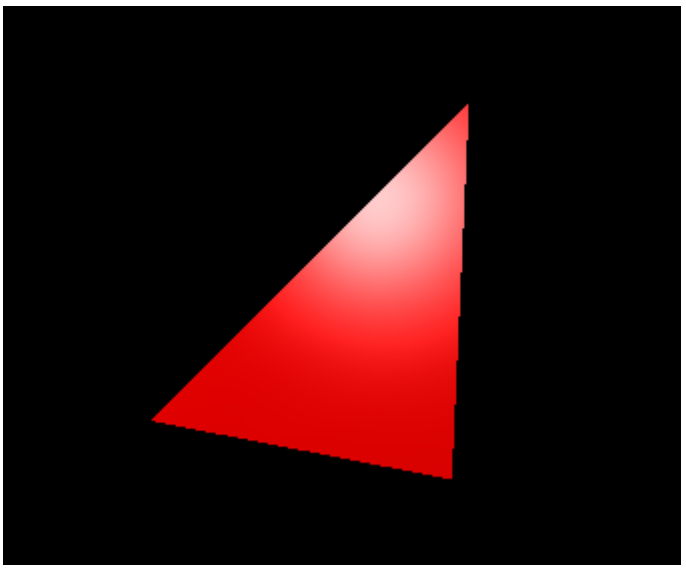

figura 6. Resultado da inclusão do suporte ao rendering de triângulos.

Dificuldade e Melhorias Possíveis

A dificuldade foi entender como implementar o R(reflexão de L sobre N), ao final vi que estava tentando complicar algo simples de ser feito, além disso tive que dedicar um tempo considerável olhando vários exemplos para entender como implementar o método de suporte ao rendering de triângulos, chegamos a conclusão de utilizar o método do artigo “Fast, Minimum Storage Ray/Triangle Intersection (Möller, T.; Trumbore, B.)”.

Trabalho poderia ser refeito em outra linguagem que o deixasse mais otimizado, poderia ter sido feita uma “figura” mais elaborada para desenvolver mais conhecimento, poderia ser feita a implementação de tipos de textura as quais fossem interessante utilizar outros tipos de modelos de iluminação.

Referências

1. Notas e vídeos de aulas do Prof. Christian Azambuja Pagot.
2. https://threejs.org/docs/index.html#api/en/
3. https://math.stackexchange.com/questions/13261/how-to-get-a-reflection-vector
4. Seção 10.3.2 do livro Fundamentals of Computer Graphics 2nd Ed.
5. https://www.realtimerendering.com/intersections.html
6. https://cadxfem.org/inf/Fast%20MinimumStorage%20RayTriangle%20Intersection.pdf

Link para o código

Parte 1 - https://codepen.io/Reniwtz/pen/eYWNegQ
Parte 2 - https://codepen.io/Reniwtz/pen/eYWNyYO