Atividade Prática 5 - *Ray Tracing*

GDSCO0051 - Introdução à Computação Gráfica - 2021.1

Data de entrega: 29/11/2021, 23h59min.

1 Atividade

Nesta atividade os alunos farão duas extensões ao *ray tracer* fornecido com o exercício: 1) inclusão do termo especular ao modelo de iluminação fornecido originalmente com o *ray tracer*; e 2) inclusão de suporte ao rendering de triângulos. Após a implementação destas extensões, os alunos deverão compor e renderizar uma ou mais cenas que demonstrem o funcionamento do *ray tracer* e das novas extensões.

2 Objetivo

O objetivo deste trabalho é familiarizar os alunos com as técnicas de geração de imagens baseadas em *ray tracing*.

3 O *Framework*

O trabalho deve ser desenvolvido em JavaScript utilizando-se, como ponto de partida, o *framework* disponível no endereço:

`https://codepen.io/ICG-UFPB/pen/BaRqvpR`

O template fornecido com este exercício renderiza uma esfera vermelha iluminada com um modelo local de iluminação composto por um termo ambiente e um difuso, como ilustrado na Figura 1.

Importante: O código template utiliza a biblioteca `three.js` apenas para tirar proveito de suas funções de cálculos com vetores (*e.g.* produto interno, produto vetorial, operações aritméticas com vetores, etc.). Desta forma, o *ray tracer* do template não faz nenhum uso de OpenGL, *shaders*, Direct3D, etc., realizando todos os seus cálculos apenas em JavaScript. Da mesma forma, as atividades a serem implementadas devem utilizar apenas JavaScript, limitando o seu uso do `three.js` ao uso das funções de cálculos com vetores.

4 Desenvolvimento

As seções a seguir descrevem os três exercícios a serem desenvolvidos nesta atividade.

4.1 Exercício 1: Inclusão do termo especular no modelo de iluminação local do *ray tracer*.

O modelo de iluminação de Phong, como visto em aula, é composto pela combinação de três modelos distintos de iluminação local: os modelos ambiente, difuso e especular. A Equação 1 descreve o modelo de Phong.

$$I = I_a \kappa_a + I_p \kappa_d \left(\mathbf{n} \cdot \mathbf{l}\right) + I_p \kappa_s \left(\mathbf{r} \cdot \mathbf{v}\right)^n \tag{1}$$

onde

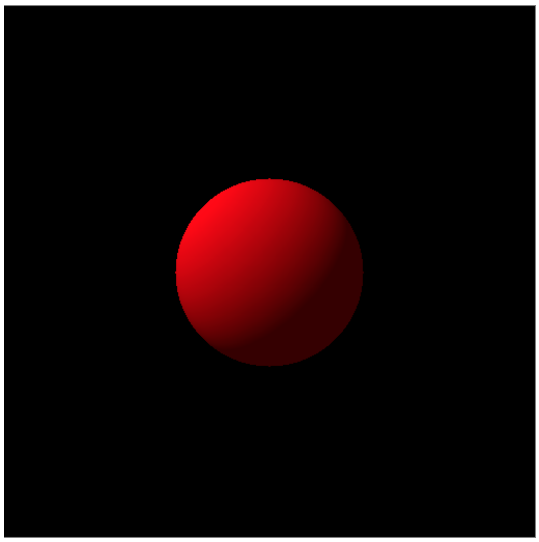

Figure 1: Esfera vermelha iluminada através de um modelo local de iluminação que considera apenas os termos ambiente e difuso.

- I : intensidade (cor) final do pixel.
- I_a : intensidade da luz ambiente.
- κ_a : coeficiente de reflectância ambiente.
- I_p : intensidade da luz pontual/direcional.
- κ_d : coeficiente de reflectância difusa.
- $\mathbf{n}$: vetor normalizado perpendicular à superfície.
- $\mathbf{l}$: vetor normalizado que aponta para a fonte de luz pontual/direcional.
- κ_s : coeficiente de reflectância especular.
- $\mathbf{r}$: vetor normalizado que representa a reflexão de $\mathbf{l}$ em relação à $\mathbf{n}$.
- $\mathbf{v}$: vetor normalizado que aponta para a câmera.
- n : tamanho do brilho especular.

Entretanto, o *ray tracer* disponibilizado para a realização desta atividade implementa um modelo mais simples, composto apenas pelos termos ambiente e difuso, como mostra a Equação 2.

$$I = I_a \kappa_a + I_p \kappa_d \left(\mathbf{n} \cdot \mathbf{l}\right) \tag{2}$$

Esta primeira atividade consiste em se extender o modelo de iluminação do *ray tracer* fornecido com o template (Equação 2) de forma que ele passe a suportar o modelo completo de iluminação de Phong (*i.e.* Equação 1).

Após a extensão do modelo de iluminação do *ray tracer*, gere imagens que ilustrem os resultados obtidos com o novo modelo de iluminação. A título de verificação, a Figura 2 ilustra a mesma esfera da Figura 1, porém renderizada com o modelo completo de Phong. Neste caso, utilizou-se $n = 32$ e $\kappa_s = (1, 1, 1)$.

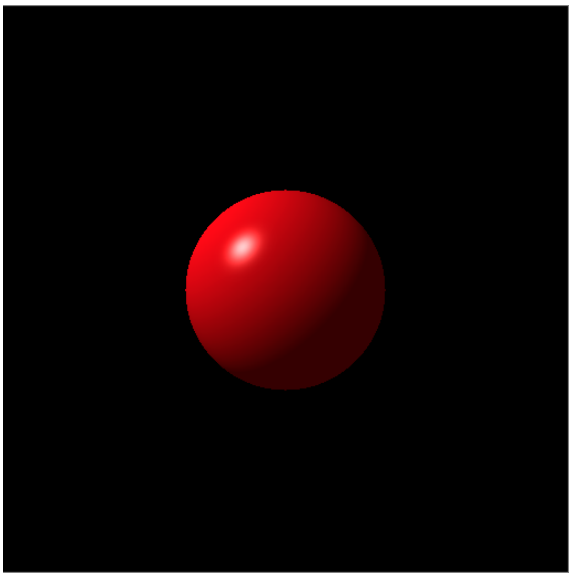

Figure 2: Esfera vermelha iluminada através do modelo de iluminação de Phong, com $\kappa_s = (1, 1, 1)$ e $n = 32$.

4.2 Exercício 2: Inclusão do suporte ao rendering de triângulos.

O template fornecido com este exercício suporta apenas o rendering de primitivas do tipo esfera. Esta segunda atividade consiste na inclusão do triângulo como uma outra possibilidade de primitiva do *ray tracer*. Os alunos tem liberdade de definir a classe que representará o triângulo da forma que quiserem, desde que esta classe conte com um método `interseccionar(raio, interseccao) {...}`, para manter a compatibilidade com o código do template.

Embora alunos estejam livres para implementarem o método `interseccionar()` da maneira que preferirem, abaixo seguem algumas sugestões de métodos que podem ser utilizados:

- Abordagem ingênua:

  O método tradicional consiste em se interseccionar o raio contra o plano que contém o triângulo, e então verificar se o ponto de intersecção está dentro da área do triângulo. Este é o método apresentado na Seção 10.3.2 do livro *Fundamentals of Computer Graphics 2nd Ed.*, e foi o método utilizado para gerar a imagem da Figura 3. Apesar de eficaz e didático, este algoritmo não é muito eficiente.

- *Fast, Minimum Storage Ray/Triangle Intersection* (Möller, T.; Trumbore, B.):

  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, disponível em `https://www.mitsuba-renderer.org/index_old.html`) no caso do rendering de malhas com muitos triângulos. O artigo onde este método foi originalmente publicado está disponível em `https://cadxfem.org/inf/Fast%20MinimumStorage%20RayTriangle%20Intersection.pdf`, e inclui código fonte em C.

O site *Scratchapixel* apresenta uma discussão bastante detalhada sobre o cálculo de interseccção raio/triângulo, incluindo o algoritmo de Möller-Trumbore e exemplo de implementação em C++. O link para a página é:
`https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/why-are-triangles-useful`.

A título de verificação, a Figura 3 ilustra os resultados esperados no caso do rendering de um triângulo com o mesmo material da esfera da Figura 2, e com vértices $(-1, -1, -3.5)$, $(1, 1, -3)$ e $(-0.75, -1, -2.5)$.

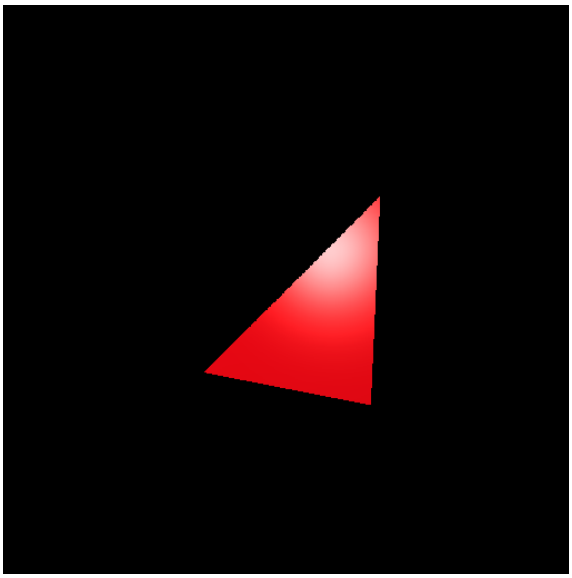

Figure 3: Triângulo vermelho iluminado através do modelo de iluminação de Phong, com vértices $(-1, -1, -3.5)$, $(1, 1, -3)$ e $(-0.75, -1, -2.5)$.

4.3 Exercício 3: Construção e renderização de uma ou mais cenas.

Uma vez que os Exercícios 1 e 2 tenham sido realizados, e as implementações correspondentes estejam funcionando, os alunos deverão construir uma ou mais cenas e renderizá-las no *ray tracer* extendido. O tema das cenas é livre, e devem enfatizar a capacidade do *ray tracer* de suportar diversos materiais e primitivas.

Importante: Usem a criatividade!

5 Entrega

Os trabalhos devem ser entregues, via atividade específica do SIGAA, até as **23 horas e 59 minutos** do dia **29/11/2021**. A entrega consistirá em um arquivo compactado (*i.e.* ZIP) contendo um relatório e o código fonte:

1. Relatório no formato PDF, contendo:
   (a) Nome e número de matrícula do(s) alunos(s).
   (b) Um parágrafo que descreva a atividade desenvolvida.
   (c) Breve explicação das estratégias adotadas pelo aluno na resolução da atividade.
   (d) *Printscreens* das imagens geradas e discussão sobre os resultados obtidos, dificuldades e possíveis melhoras.
   (e) Referências bibliográficas.

(f) O aluno pode, se assim desejar, disponibilizar seu código fonte também em um repositório *online* (*e.g.* `codepen.io`, `jsfiddle.net`, etc.) e incluir o *link* para este respositório em seu relatório. Entretanto, observa-se que esta disponibilização do código fonte em sites é opcional, não vale nota, e não substitui o envio do código fonte pelo SIGAA.

2. Arquivo contendo o código fonte em JavaScript.

Não serão aceitos trabalhos enviados por outro meio que não o SIGAA.

Este trabalho pode ser desenvolvido em duplas.

Importante:

1. Não serão aceitos trabalhos enviados por outro meio que não o SIGAA.
2. Trabalhos entregues até às **23h e 59min** do dia **29/11/2021** serão contabilizados como presença em aula. Trabalhos que **não forem entregues** até às **23h e 59min** do dia **29/11/2021** serão contabilizados como **falta em aula**, e receberão **nota zero**.