Atividade Prática 4 - Implementação de Modelos de Iluminação

GDSCO0051 - Introdução à Computação Gráfica - Turma 02 - 2019.4

Data de entrega: 10/08/2020, 23h59min.

1 Objetivo

O objetivo deste trabalho é familiarizar os alunos com os modelos de iluminação tradicionalmente utilizados na rasterização: *ambiente*, *difuso* e *especular* (ou *Phong*). Para isto, os alunos implementarão os três modelos utilizando o *vertex shader* do OpenGL.

2 Desenvolvimento

2.1 Código Template

Esta atividade de implementação será feita a partir de um código template `C++17` disponibilizado pelo professor no repositório da disciplina:

```
https://github.com/capagot/icg/tree/master/04_shading
```

2.2 Dependências

O código template disponibilizado depende de software de terceiros. As seções a seguir apresentam cada uma destas dependências.

2.2.1 `glm`

Este programa depende da biblioteca `glm` (`https://glm.g-truc.net/0.9.9/index.html`). Para facilitar a instalação desta dependência, ela foi inclusa no projeto do Github na forma de um submódulo. Assim, ao se fazer um clone do projeto, a `glm` pode ser instalada automaticamente.

Para clonar o projeto, e fazer *dowload* automático do submódulo da `glm`, o seguinte comando deve ser executado:

```
$ git clone -recurse-submodules https://github.com/capagot/icg.git
```

A opção `clone` fará uma cópia local do repositório enquanto a opção `-recurse-submodules` fará com que o conteúdo de todos os submódulos do projeto sejam inclusos na cópia local.

2.2.2 `GLEW`

Enquanto a linguagem `C` aceita declarações implícitas de funções, a linguagem `C++` não aceita. Assim, para que um programa `C++`, que utilize o OpenGL moderno, possa ser compilado com sucesso, todas as extensões OpenGL devem estar declaradas de forma explícita. A biblioteca `GLEW` tem a função de gerar as referências para todas as extesões OpenGL. O programa template disponibilizado faz uso da `GLEW` para a geração destas referências. Os procedimentos para a instalação da `GLEW` variam de acordo

com o sistema operacional. Maiores detalhes sobre como instalar a `GLEW` podem ser obtidos em seu site oficial: `http://glew.sourceforge.net`.

Existem outras bibliotecas para carga das extensões OpenGL, como por exemplo a `GLAD`, disponível em `https://glad.dav1d.de`. Os alunos estão livres para utilizarem outras bilbioteca para a carga das extensões OpenGL. Lembramos apenas que, neste caso, como o código disponibilizado pelo professor foi baseado na `GLEW`, ele deverá ser adaptado de acordo.

2.3 Compilação e Execução do Código Template

Uma vez que todas as dependências estejam satisfeitas, pode-se passar para a compilação do código template. O comando de compilação em sistemas Linux deverá ser:

```
$ g++ -Wall -Wextra -Wpedantic -std=c++17 -O0 -g3 -DDEBUG main.cpp -lglut -lGLEW -lGLU
-lGL -o shading_gl
```

Como este comando já se encontra descrito no arquivo `Makefile` disponibilizado no projeto, basta apenas emitir o seguinte comando a partir do diretório raíz deste programa:

```
$ make
```

Durante a compilação alguns *warnings* poderão aparecer na tela. Uma vez compilado, ao ser executado, o programa deverá apresentar uma janela contendo a imagem do *Utah Teapot* (`https://en.m.wikipedia.org/wiki/Utah_teapot`), em azul escuro, como ilustra a Figura 1:

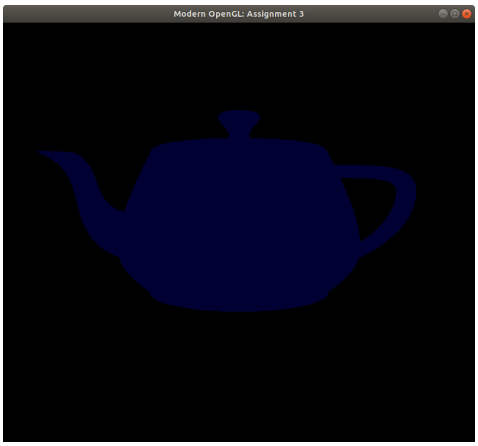

Figure 1: Renderização do *Utah Teapot* utilizando apenas o modelo de iluminação ambiente: $I = I_a \kappa_a$, onde $I_a = (0.2, 0.2, 0.2)$ e $\kappa_a = (0, 0, 1)$.

2.4 A Cena

A cena a ser renderizada nesta atividade é composta por uma malha de triângulos que representa o *Utah Teapot* e uma fonte de luz pontual. O *teapot* encontra-se centrado na origem enquanto a fonte de luz pontual encontra-se localizada na posição $(-2, 2, 1.5)$. A câmera encontra-se localizada na posição $(0, 0, 1.5)$, com o vetor $Up = (0, 1, 0)$ e direção $= (0, 0, -1)$. Todas as posições são informadas no *Espaço do Universo*.

A configuração geométrica da cena renderizada na Figura 1 pode ser visualizada na Figura 2.

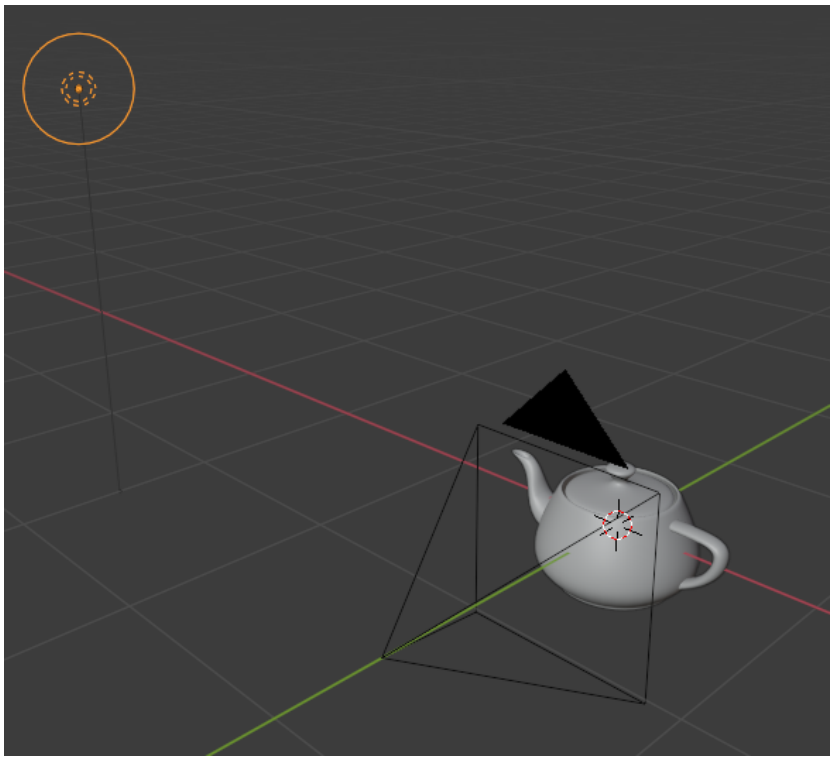

Figure 2: Configuração geométrica da cena, representada no *Espaço do Universo*, a ser utilizada nesta atividade. O ponto laranja no canto superior esquerdo representa a posição da fonte de luz pontual. A pirâmide preta abaixo da imagem representa o volume de visualização da câmera, que se encontra posicionada sobre o eixo Z, no vértice do volume. O triângulo preto acima do volume de visualização representa a direção *Up* da câmera. O sistema de coordenadas é mão-direita, onde o eixo vermelho representa o eixo X e o verde o eixo Z.

2.5 A Atividade

O programa template, em sua configuração original, renderiza o *teapot* utilizando apenas o modelo de reflexão *ambiente*. Esta atividade consiste em se alterar, no programa template, o conteúdo do *vertex shader* (arquivo `vertex_shader.glsl`) de forma a se adicionarem os modelos de reflexão *difuso* e *especular*.

2.5.1 Sobre o *Vertex Shader*

Abaixo segue o código fonte atual do *vertex shader* (arquivo `vertex_shader.glsl`):

```
1  #version 330 core
2
3  layout (location = 0) in vec3 obj_spc_vertex_pos;
4  layout (location = 1) in vec3 obj_spc_N;
5  layout (location = 2) in vec3 k_d;
6
7  uniform mat4 model_mat;
8  uniform mat4 view_mat;
9  uniform mat4 proj_mat;
10
```

```
11 out vec3 I;
12
13 void main() {
14   vec3 cam_pos = vec3(0.0f, 0.0f, 1.5f);
15
16     vec3 I_a = vec3(0.2f, 0.2f, 0.2f);
17     vec3 k_a = vec3(0.0f, 0.0f, 1.0f);
18     vec3 k_s = vec3(1.0f, 1.0f, 1.0f);
19     vec3 I_p_pos = vec3(-2.0f, 2.0f, 1.5f);
20     vec3 I_p = vec3(0.8f, 0.8f, 0.8f);
21
22     I = I_a * k_a;
23
24     gl_Position = proj_mat * view_mat * model_mat * vec4(↩
         obj_spc_vertex_pos, 1.0);
25 }
```

Listing 1: Conteúdo do arquivo `vertex_shader.glsl`

A seguir comentários acerca do código-fonte listado acima:

- Linha 1: Versão da linguagem GLSL: `330 core`
- Linhas 3-5: Atributos do vértice.

  Variável `obj_spc_vertex_pos`: coordenadas no espaço do objeto (atributo `0`);

  Variável `obj_spc_N`: vetor normal (atributo `1`);

  Variável `k_d`: coeficiente de reflectância difusa (atributo `2`).
- Linhas 7-9: Matrizes 4×4 do pipeline.

  Variável `model_mat`: matriz *Model*;

  Variável `view_mat`: matriz *View*;

  Variável `proj_mat`: matriz *Projection*.
- Linha 11: Variável `I`: atributo de intensidade (ou cor) a ser calculado para o vértice.
- Linha 14: Variável `cam_pos`: posição da câmera (*hard coded*) no espaço do *Universo*.
- Linha 16: Variável `I_a`: intensidade da luz ambiente.
- Linha 17: Variável `k_a`: coeficiente de reflectância ambiente (κ_a).
- Linha 18: Variável `k_s`: coeficiente de reflectância especular (κ_s).
- Linha 19: Variável `I_p_pos`: posição da fonte de luz pontual (*hard coded*) no espaço do *Universo*.
- Linha 20: Variável `I_p`: intensidade da luz pontual.
- Linha 22: Cálculo do modelo de iluminação ambiente.
- Linha 24: Atualização da posição do vértice, originalmente no espaço do *Objeto*, para o espaço de *Recorte*.

2.5.2 Exercício 1: Implementação do Modelo de Reflexão Difuso

Modificar o *vertex shader* (arquivo `vertex_shader.glsl`), de forma que ele passe a incluir, além do modelo de iluminação *ambiente*, também o modelo *difuso*. Para isto, será necessário calcular o novo valor do vetor normal `N`, após as transformações feitas pela matriz *Model*, bem como o vetor `L` que aponta do vértice para a fonte de luz. Abaixo seguem as sugestões de cálculo destes dois vetores:

```
vec3 L = normalize(I_p_pos - (model_mat * vec4(obj_spc_vertex_pos, 1.0)).xyz);
vec3 N = normalize(mat3(transpose(inverse(model_mat))) * obj_spc_N);
```

Após os cálculos dos vetores `N` e `L`, o modelo de iluminação difuso a ser avaliado deverá ser:

$$I = I_a \kappa_a + I_p \kappa_d \cos\theta$$

onde $\cos\theta = (L \cdot N)$.

Após a implementação, se tudo estiver correto, o resultado obtido deverá ser igual ao da Figura 3 (ver os parâmetros utilizados na legenda da figura!).

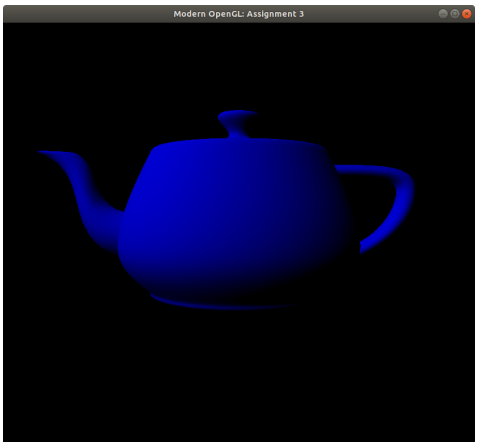

Figure 3: Renderização do *Utah Teapot* utilizando o modelo $I = I_a \kappa_a + I_p \kappa_d \cos\theta$ (ambiente + difuso), onde $I_a = (0.2, 0.2, 0.2)$, $\kappa_a = (0, 0, 1)$, posição da fonte de luz igual a $(-2, 2, 1.5)$, $I_p = (0.8, 0.8, 0.8)$ e $\kappa_d = (0, 0, 0.8)$

2.5.3 Exercício 2: Implementação do Modelo de Reflexão Especular

Modificar o *vertex shader* (arquivo `vertex_shader.glsl`) do exercício anterior de forma que ele passe a incluir, além do modelo de iluminação *ambiente* e *difuso*, também o modelo *especular*, ou de *Phong*. Para isto, será necessário calcular o vetor `R`, de reflexão da luz `L`, e o vetor da câmera `V`, que aponta do vértice em questão para a câmera. Abaixo seguem as sugestões de cálculo destes dois vetores:

```
vec3 R = -reflect(L, N);
vec3 V = normalize(cam_pos - (model_mat * vec4(obj_spc_vertex_pos, 1.0)).xyz);
```

Após os cálculos dos vetores `R` e `V`, o modelo de iluminação a ser avaliado deverá ser o de *Phong*:

$$I = I_a \kappa_a + I_p(\kappa_d \cos\theta + \kappa_s(\cos\alpha)^n)$$

onde $\cos\theta = (L \cdot N)$, $\cos\alpha = (R \cdot V)$ e n é a potência do termo especular (controla o tamanho do brilho especular).

Após a implementação, se tudo estiver correto, o resultado obtido deverá ser igual ao da Figura 4 (ver os parâmetros utilizados na legenda da figura!).

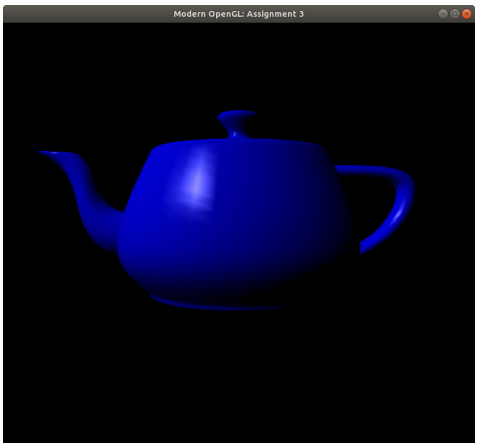

Figure 4: Renderização do *Utah Teapot* utilizando o modelo $I = I_a\kappa_a + I_p(\kappa_d \cos\theta + \kappa_s(\cos\alpha)^n)$ (*Phong*), onde $I_a = (0.2, 0.2, 0.2)$, $\kappa_a = (0, 0, 1)$, posição da fonte de luz igual a $(-2, 2, 1.5)$, $I_p = (0.8, 0.8, 0.8)$, $\kappa_d = (0, 0, 0.8)$, $\kappa_s = (1, 1, 1)$ e $n = 64$.

3 Entrega

Os trabalhos devem ser entregues até o dia **10/08/2020, impreterivelmente até as 23 horas e 59 minutos**. O trabalho será considerado entregue assim que estiver acessível, de forma pública, no repositório Git informado, via SIGAA, pelo aluno. Este trabalho pode ser desenvolvido em duplas.

No repositório deverão constar os *printscreens* dos resultados acompanhados de pequenos textos (frases ou parágrafos) explicativos. O repositório deve conter também o código fonte. Abaixo segue um detalhamento de cada um dos componentes que devem aparecer no *post*.

- ***Printscreens***: Devem demonstrar a evolução do processo de implementação, incluindo eventuais problemas encontrados, e os resultados obtidos após suas correções.
- **Texto**:
  - Deve iniciar com um parágrafo que descreva a atividade desenvolvida.
  - Deve explicar brevemente as estratégias adotadas pelo aluno na resolução da atividade.
  - Breve discussão sobre os resultados gerados, dificuldades e possíveis melhoras.
  - Listar as referência bibliográficas consultadas durante o desenvolvimento do trabalho (livros, artigos, endereços web), se for o caso.
  - OBS: A inclusão de trechos de código no corpo do post deve se limitar ao mínimo indispensável para o correto entendimento do texto!