UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIAS DE COMPUTAÇÃO

Trabalho 4

Aplicação gráfica interativa 3D com objetos, câmera, fontes de luz e algoritmos de iluminação e sombreamento

SCC0250 - Computação Gráfica

Profa. Maria Cristina Ferreira de Oliveira

PAE: Rafael Nakanishi

Grupo:

Flavio Vinicius Vieira Santana	Nº 9866552
Mateus Castilho Leite	Nº 9771550
Vinicius Henrique Borges	Nº 9771546

1. Introdução geral

Este trabalho de Computação Gráfica visa a implementação dos algoritmos de iluminação de Phong e de Gouraud realizada sobre uma cena com múltiplos objetos e uma câmera móvel, sendo a troca entre os algoritmos feita através do teclado.

Para tal, a aplicação deve ler informações sobre a cena descrita em um arquivo de texto *scene.txt*, onde estarão contidos, em cada linha, os objetos e suas respectivas posições no sistema de coordenadas do mundo; as fontes de luz pontuais e suas respectivas posições, intensidades de cores e fatores de atenuação (na ordem linear, constante e quadrática); e a câmera com sua respectiva posição, ponto de observação (*lookAt point*) e vetor *view up*.

As entradas da aplicação são {a,w,s,d} para mover a câmera para {esquerda, frente, trás, direita}. Ao segurar o shift, {a,w} se movimentam para {cima, baixo}. Ao pressionar Esc, a aplicação fecha. Ao pressionar L, troca-se entre o efeito de iluminação de Phong e Gouraud. Ao pressionar C, alterna-se a exibição de um cubo no meio da tela (detalhes na seção 3).

2. Introdução teórica

Para a realização desse trabalho foi utilizado o conhecimento teórico sobre as técnicas de iluminação de Phong e de Gouraud.

Na de Gouraud, calculamos a intensidade de iluminação em cada vértice do objeto, interpolamos essa intensidade de iluminação e depois calculamos a cor do pixel usando essa intensidade de iluminação junto à cor, ou junto à textura.

Já na técnica de Phong, obtemos o vetor normal de cada vértice, interpolamos esse vetor normal ao longo do objeto, calculamos a intensidade de iluminação em cada fragmento e então calculamos a cor do pixel usando essa intensidade de iluminação junto à cor, ou junto à textura.

Também é necessário saber o que fazer quando existem várias fontes de luz na cena. Nesse caso, somamos as contribuições de luz de cada fonte.

3. Resolução dos problemas

3.1. Leitura e posicionamento dos objetos

No método `Window::run()` do arquivo *graphicslib.cpp*, a partir da linha 131 é feita a leitura do arquivo *scene.txt*. Mais abaixo, o if da linha 200 verifica se a linha atual do *scene.txt* está definindo um objeto e, em caso afirmativo, obtém o caminho (`std::string path`) desse objeto e a posição final do seu centro de massa (`glm::vec3 finalPos`). Essas informações serão repassadas às classes responsáveis pelo processamento dos objetos, de forma similar ao Trabalho 3, porém utilizando-se de uma nova classe `ModelInformation` para guardar informações do objeto sendo carregado na linha atual do *loop*, de modo que cada objeto seja inicialmente transladado para a origem utilizando seu centro de massa calculado (por exemplo: `model.boundingBox.x.center`) e, em seguida, transladado para sua posição final (por exemplo: `finalPos.x`), isso é feito entre as linhas 446 e 457 do *graphicslib.cpp*.

3.2. Leitura e implementação das fontes de luz

A leitura das fontes de luz foi feita de forma similar à de objetos, detectando se a primeira palavra era "*light*" e então adicionando as informações à *struct Lighting Information.* Essas operações estão no arquivo *graphicslib.cpp* embaixo do comentário *READ LIGHTS FROM FILE*.

Na *struct Lighting Information* existe um *array bufferOfPointLights* que é passado para o *buffer* na função *loadPointLightsVAO* no arquivo *graphicslib.cpp*. Esse função é chamada embaixo do comentário *SETUP SHADERS*.

A exibição das luzes foi feita embaixo do comentário *DRAW POINT LIGHTS* no arquivo *graphicslib.cpp*.

3.3. Implementação de Phong e Gouraud

Os *shaders* estão da pasta *src* e são muito parecidos entre si. Se checarmos o *shader multipleLightsPhongColor.vs*, veremos que são aplicadas as transformações de *model, view* e *projection* à posição do vértice. Além disso, é aplicada a transformação de *model* de uma forma diferente para a *Normal*, pois operações de escala e rotação podem afetá-la, já operações de translação não, pois

a normal é um vetor. Para o *fragment shader* correspondente (*multipleLightsPhongColor.fs*) é passada a normal e a posição do fragmento. Nesse ponto, a normal é interpolada para calcular os framentos. Então calculamos o efeito da luz no fragmento, somando os efeitos de cada fonte de luz (*pointLights[]*).

No shader explicado, em específico, após calcularmos a luz, multiplicamos pela cor e a atenuação. Já nos shaders equivalentes de textura (*multipleLightsPhongTex*), multiplicamos pela atenuação e pelas texturas difusas e especular. A diferença entre os *shaders* de Gouraud para os de Phong, é que nos de Gouraud calculamos a intensidade de iluminação para cada vértice no *vertex shader* e ao passar para o *fragment shader* essa intensidade é interpolada. Isso é mais barato computacionalmente, principalmente porque a quantidade de fragmentos é muito superior à de vértices, então calcular a iluminação utilizando o modelo de iluminação de phong para cada fragmento leva muito mais tempo que calcular para cada vértice.

3.4. Troca de shaders de Phong e Gouraud

A troca de shaders para os objetos foi implementada utilizando a informação do vetor de *structs ModelInformation* declarado embaixo do comentário *MODEL INFORMATION VECTOR* no arquivo *graphicsLib.hpp*.

Essa *struct* é preenchida na leitura dos modelos, onde são atribuídos os shaders de textura ou cor, de acordo com o número de texturas que o modelo tem. Isso é feito embaixo do comentário *SHADER SELECTION* no arquivo *graphicsLib.cpp*.

Então para exibir os objetos na tela, verifica-se se a variável *mPhong* é verdadeira, embaixo do comentário *SHADING SELECTION* no arquivo *graphicsLib.cpp*. Essa variável é atualizada na função *updateInput* no mesmo arquivo.

3.5. Leitura e posicionamento da câmera

No método `Window::run()` do arquivo *graphicslib.cpp*, o if da linha 190 verifica se a linha atual do *scene.txt* está definindo a câmera e, em caso afirmativo, obtém sua posição (`glm::vec3 position`), o vetor *view up* (`glm::vec3 up`) e o *lookAt point* (`glm::vec3 up`). Com essas informações, é criada uma câmera através do construtor da classe `Camera` (arquivo *camera.cpp*, a partir da linha 4), tal

classe armazena e incrementa os ângulos de *Yaw* e *Pitch* a cada interação com o mouse. Inicialmente, no construtor (linhas 9 e 10), esses ângulos são calculados através das seguintes fórmulas:

yaw = arcotangente ((lookAt.x - position.x) / (position.z - lookAt.z)) - 90º

pitch = arcotangente ((lookAt.y - position.y) / (position.z - lookAt.z))

No caso do *yaw*, foi necessário subtrair 90 graus pois descobriu-se que o ângulo inicial padrão estava alinhado com o eixo x, de modo que essa operação alinha o vetor de observação com o eixo z. Seu valor será armazenado em graus e possuirá qualquer valor permitido para um *float*.

O *pitch* será dado em graus e estará sempre no intervalo [-90º, +89º] para impedir que a cena seja invertida verticalmente.

3.6. Cubo Extra

Ao pressionar a tecla “C” pode-se controlar a exibição de um cubo no centro da tela. Esse cubo foi implementado nas fases iniciais do projeto para se verificar mais nitidamente as diferenças de efeito das técnicas de Phong e de Gouraud. Ao final do projeto, atribuímos o controle dessa exibição à tecla “C” e deixamos como *feature* extra. Isso explica porque o cubo tem shaders separados para ele, pois esses shaders foram criados na fase inicial e serviram de base para criar os shaders para os modelos do formato *Wavefront obj*.

4. Contribuições

4.1. Flavio

- Leitura e implementação das fontes de luz;
- Implementação de Phong e Gouraud;
- Troca de shaders de Phong e Gouraud;

4.2. Mateus

- Criação de arquivos para testes;
- Implementação do cubo extra;

4.3. Vinicius

- Leitura e tratamento das informações contidas no *scene.txt*;
- Cálculo dos ângulos *Yaw* e *Pitch* a partir das informações da câmera;
- Obtenção de objetos para testes.

Conclusão

A execução da aplicação está dentro do esperado para as cenas em que testamos. Foram testadas diversas posições para a câmera, além de seu ponto de visualização e seu vetor *view up*, não tendo sido encontrado nenhum problema. O mesmo se aplicou aos objetos e as fontes de luz.

Uma dificuldade relevante que tivemos no início foi entender como era a implementação dos algoritmos de Phong e Gouraud. Após entender isso, ficou mais claro o funcionamento do pipeline do opengl e quando é feita a interpolação.

Outra dificuldade, menos importante, foi não saber como fazer “fontes de luz pontuais de tamanho 10”. Porém após pedirmos ajuda ao monitor, entendemos o que estava sendo pedido e a solução foi bem simples.

Acreditamos que, apesar de os trabalhos terem despendido bastante esforço e a curva de aprendizado de *OpenGl* ser grande, o trabalho foi condizente com conteúdo dado na disciplina.