Iluminação de objetos 3D

Alexandre Diniz de Souza, Giovane H. Iwamoto, Leandro José G. Pereira

Universidade Federal de Mato Grosso do Sul (UFMS) - FACOM

1. Projeto Base

Para a realização dessa etapa do trabalho utilizamos como base o projeto enviado anteriormente que consistia no desenvolvimento de 3 objetos 3D distintos que rotacionam em torno do próprio eixo e nos quais foram aplicados uma projeção do tipo perspectiva. Dando continuidade a essa nova parte do trabalho, explicamos a seguir as alterações realizadas a fim de aplicar os 4 tipos de iluminação estudados durante as aulas.

2. Iluminação Ambiente

Para aplicar uma iluminação ambiente aos objetos da cena foi necessário a adição do seguinte trecho de código no shader de fragmento (localizado em res/main.fs) responsável por definir a intensidade da luz ambiente (0.1) e a cor da fonte de luz (branca) além de realizar o cálculo necessário da cor de cada fragmento aplicando esta luz ambiente:

```
C/C++
float ambientStrength = 0.1;
vec3 lightColor = vec3(1.0f, 1.0f, 1.0f);
vec3 ambient = ambientStrength * lightColor;

vec3 result = ambient * ourColor;
FragColor = vec4(result, 1.0f);
```

3. Iluminação Difusa

Para aplicar uma iluminação difusa aos objetos da cena foi necessário, primeiramente, o cálculo dos vetores normais dos vértices. As normais foram então adicionadas nos vetores que já guardavam as coordenadas dos vértices e as cores de cada face de cada objeto.

Utilizando as funções `glVertexAttribPointer` e `glEnableVertexAttribArray` indicamos ao OpenGL as posições na qual estão armazenadas as normais dentro dos vetores. Além disso, passamos também para o shader a posição da fonte de luz. Como

nossos 3 objetos são independentes e a fim de explorar diferentes impactos de uma fonte de luz, posicionamos uma fonte de luz distinta para cada objeto, sendo que uma luz de um objeto não interfere em outro. Dito isso, posicionamos a fonte de luz do cubo no centro superior desse objeto, da pirâmide no centro da face e do prisma hexagonal no canto inferior esquerdo

No shader de vértices foi necessária a adição de trechos de códigos para o recebimento das normais adicionadas e repasse das mesmas para o shader de fragmento levando em consideração a matriz de transformação utilizada para realizar a rotação dos objetos.

No shader de fragmento foi necessário a adição de trechos de código para o recebimento das normais vindas do shader de vértices além do seguinte código responsável pelo cálculo correto das cores de cada fragmento e definição da cor da fonte de luz:

```
C/C++
vec3 lightColor = vec3(1.0f, 1.0f, 1.0f);

vec3 norm = normalize(Normal);
vec3 lightDir = normalize(lightPos - FragPos);

float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * lightColor;

vec3 result = diffuse * ourColor;

FragColor = vec4(result, 1.0f);
```

4. **Iluminação Especular**

Assim como na iluminação difusa, na iluminação especular também foi necessário incluir as normais aos vetores que contém as coordenadas e cores de cada face de cada objeto, instruir o OpenGL sobre onde estão localizadas as normais dentro dos vetores e passar para o shader a posição da fonte de luz. Além disso, assim como na iluminação difusa, na iluminação especular também definimos uma fonte de luz para cada objeto - sem que uma fonte de luz de um objeto impacte outro objeto. Ademais, foi necessário passar também a posição de visualização para o shader. Assim como na posição da luz, a posição de visualização foi definida independentemente para cada objeto e sem a interferência em outros objetos.

No shader de vértices foram realizadas as mesmas alterações que já haviam sido aplicadas ao shader de vértices para uma iluminação difusa.

Por fim, no shader de fragmento o trecho de código a seguir foi inserido. Sua função é definir a cor da fonte de luz (lightColor), a intensidade da luz pontual (specularStrength) e realizar o cálculo utilizando as normais definidas anteriormente junto ao coeficiente de brilho (utilizamos 32) da cor de cada fragmento para formar uma iluminação especular:

```
C/C++
vec3 lightColor = vec3(1.0f, 1.0f, 1.0f);
float specularStrength = 1.0;

vec3 norm = normalize(Normal);
vec3 lightDir = normalize(lightPos - FragPos);
vec3 viewDir = normalize(viewPos - FragPos);
vec3 reflectDir = reflect(-lightDir, norm);

float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
vec3 specular = specularStrength * spec * lightColor;

FragColor = vec4(specular, 1.0);
```

5. **Iluminação de Phong (Ambiente + Difusa + Especular)**

Para aplicar a iluminação de phong nos 3 objetos da cena bastou juntar os trechos de código adicionados para cada iluminação descrita anteriormente. Ou seja, incluímos as normais nos vetores que contém as coordenadas e cores das faces de cada objeto, passamos para os shaders a posição da fonte de luz e a posição de visualização de cada objeto da cena e por fim, adicionamos no shader de fragmento os trechos responsáveis pelo cálculo de cada tipo de iluminação. Para concluir bastou somar o resultado desses cálculos e multiplicar pela cor de cada fragmento para obter o resultado final, como podemos ver a seguir:

```
C/C++
vec3 lightColor = vec3(1.0f, 1.0f, 1.0f);
float ambientStrength = 0.1;
float specularStrength = 0.9;

vec3 ambient = ambientStrength * lightColor;
```

```
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(lightPos - FragPos);
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * lightColor;

vec3 viewDir = normalize(viewPos - FragPos);
vec3 reflectDir = reflect(-lightDir, norm);
float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
vec3 specular = specularStrength * spec * lightColor;

vec3 result = (ambient + diffuse + specular) * ourColor;

FragColor = vec4(result, 1.0f);
```

6. Execução

Cada tipo de iluminação foi colocado em uma pasta diferente. Para executar basta utilizar o Makefile de cada pasta para gerar o executável:

1. Na pasta raiz de cada tipo de iluminação execute o comando *make*
2. O arquivo executável está localizado na pasta *bin*, portanto para executar o programa basta digitar *bin/main* ainda na pasta raiz do projeto que contém a iluminação alvo.

OBS.: Caso queira compilar e executar o programa em um único comando use *make && bin/main* na pasta raíz da iluminação que deseja visualizar.