Universidade Federal da Paraíba
Centro de Informática

Introdução à Computação Gráfica
Atividade Prática 3 - Implementação de Modelos Locais de Iluminação

Francisco Siqueira Carneiro da Cunha Neto[1]
20190029015
[1]franciscneto@eng.ci.ufpb.br

4 de novembro de 2021

1 Atividade Desenvolvida

Baseando-se no *template* disponível em codepen.io/ICG-UFPB/pen/LYygXgq, nesta atividade foram escritos *shaders* que implementam o modelo de iluminação de Phong [1], utilizando tanto a interpolação de Gouraud (detalhado na seção 2), quanto a interpolação de Phong (detalhado na seção 3).

O código dos *shaders* foi armazenado por meio de *strings* nas variáveis `gouraud` e `phong` (equivalentes as suas respectivas interpolações), e dessa forma podem ser facilmente trocados na definição do material usado pelo ThreeJS.

O código fonte também pode ser encontrado em seus repositórios no GitHub e CodePen.

2 Interpolação Gouraud

Essa interpolação realiza o cálculo da cor final apenas dos vértices, interpolando entre a cor de cada vértice para definir a cor dos fragmentos. Por isso, o código descrito nesta seção está contido, em sua maioria, no *Vertex Shader*, com o *Fragment Shader* contendo apenas uma variável do tipo *varying* que armazena a cor dos vértices interpolada para aquele fragmento e então definindo a cor do fragmento para aquela armazenada nesta variável.

De acordo com a equação do modelo de Phong, são necessários três termos para o cálculo da cor de um vértice/fragmento: ambiente, difuso e especular. Em sua implementação foram declarados três vetores tri-dimensionais, cada um guardando o resultado de um desses termos.

Todas as variáveis necessárias para o cálculo dos termos ambiente e difuso já estavam disponíveis no *template*, seja por meio de *uniforms*, ou de derivações realizadas na *main*. Portanto só foi necessário utilizar funções e operadores do GLSL nessas variáveis para achar seus valores, levando às seguintes linhas de código.

```
vec3 ambient_term = Ia.xyz * k_a.xyz;
vec3 diffuse_term = Ip_diffuse_color.xyz *
    k_d.xyz * max(0.0, dot(N_cam_spc,
    L_cam_spc));
```

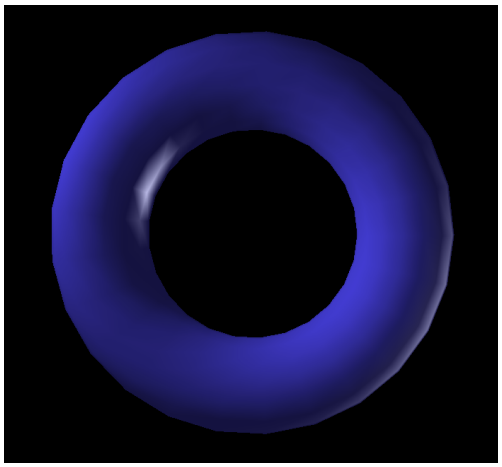

Figura 1: *Primeira tentativa de renderização utilizando interpolação de Gouraud.*

Para o termo especular ainda era necessário um elemento que não estava disponível no *template*: um vetor $\boldsymbol{v}$ que apontasse do vértice para a câmera. Para tal, é necessário conhecer a posição da câmera $\boldsymbol{c}$ e do vértice $\boldsymbol{p}$. A posição do vértice foi disponibilizada no *template* e, como todos os vetores utilizados estão no espaço da câmera, a câmera se encontra na origem. Assim, o vetor é derivado por $\boldsymbol{v} = \boldsymbol{c} - \boldsymbol{p} = -\boldsymbol{p}$, normalizado.

Após adotar uma variável adicional para representar o expoente do termo especular, o valor deste último termo é encontrado de forma similar aos outros dois.

```
vec3 specular_term = Ip_diffuse_color.xyz *
    k_s.xyz * pow(max(0.0, dot(R_cam_spc,
    V_cam_spc)), n_highlight);
```

A cor final do vértice é então definida como a soma desses três termos. O resultado inicial dessa renderização pode ser visto na Fig. 1.

É notável que, nessa primeira renderização, o brilho especular age de maneira estranha. Ele se encontra exatamente nas regiões de sombra (definidas pelo brilho difuso), como

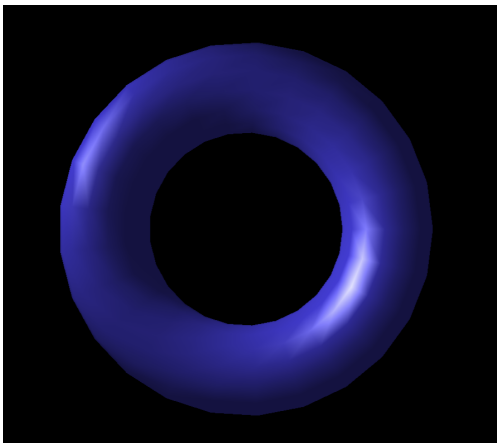

Figura 2: *Geometria renderizada utilizando interpolação de Gouraud.*

se a luz pontual estivesse na direção oposta à real. Dessa forma, foi-se deduzido que o erro estava em definir a direção da câmera como o negativo da posição do vértice normalizado. Adotando a direção apenas como a posição do vértice normalizada, essencialmente invertendo a direção do brilho, a renderização foi realizada como esperado, como mostra a Fig. 2.

3 Interpolação Phong

Para esta interpolação, o cálculo da cor final é realizado individualmente para cada fragmento, utilizando seus vetores normais. Por isso, foi necessário deixar uma parte do código no *Vertex Shader* e outra no *Fragment Shader*.

No *Vertex Shader* foram declaradas duas variáveis *varying* para vetores que precisam ser interpolados em cada fragmento, nominalmente os vetores de posição e normal, que foram subsequentemente derivados. Além disso, o vetor de posição da luz pontual no espaço de câmera também é declarado aqui como *varying* e derivado, já que apenas nesse *shader* temos acesso a matriz *Model-View*.

No *Fragment Shader* estão todos os *uniforms* (com exceção da posição da luz pontual). Utilizando essas variáveis *uniform* e *varying* derivam-se os vetores utilizados na equação do modelo: direção do fragmento para a luz pontual, a reflexão deste em relação ao normal do fragmento e a direção do fragmento para a câmera. Essa derivação é feita de forma similar à feita no *Vertex Shader* para a interpolação de Gouraud.

A partir desses vetores e parâmetros, são derivados os três termos do modelo de Phong da mesma forma do que na interpolação de Gouraud. Os três termos são então somados e a cor de cada fragmento é definida como resultado dessa soma. A renderização resultante é exibida na Fig. 3.

Essa primeira renderização também possui defeitos. O efeito das *Mach bands* é forte, e o brilho é muito estreito e comprido. Após uma revisão das aulas e análise do código, notou-se que ainda era necessário normalizar, no *Fragment Shader*, os vetores normais interpolados. Tendo corrigido isso a geometria foi renderizada exatamente como esperado, como mostra a Fig. 4.

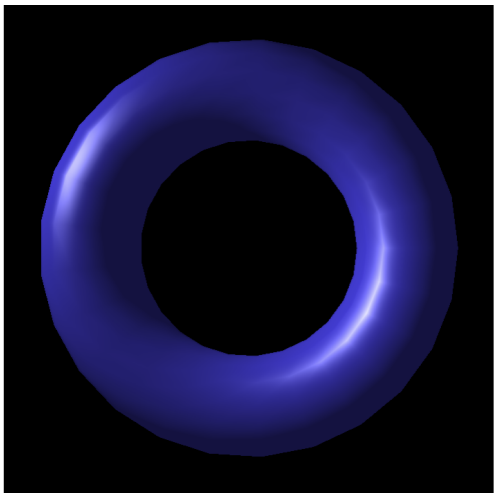

Figura 3: *Primeira tentativa de renderização utilizando interpolação de Phong.*

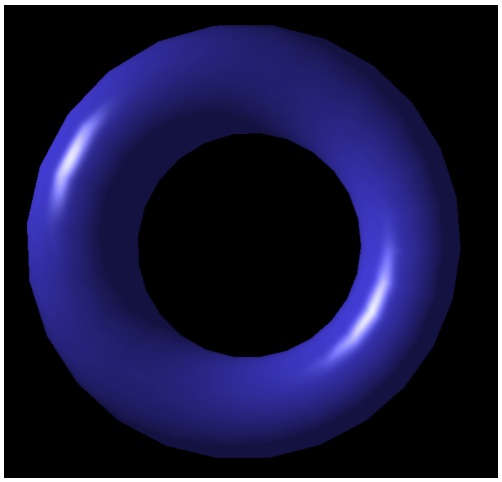

Figura 4: *Geometria renderizada utilizando interpolação de Phong.*

Referências

[1] B. T. Phong, “Illumination for computer generated pictures,” *Communications of the ACM*, v. 18, n. 6, pp. 311–317, 1975.