Universidade Federal de Mato Grosso do Sul
Faculdade de Computação

Computação Gráfica 2019

Trabalho – Parte 3

Paulo A. Pagliosa

O objetivo principal da **Parte 3** é implementar o modelo de iluminação de Phong aplicado a dois modos diferentes de tonalização de triângulos: tonalização de Gouraud e de Phong. Conforme estudado no Capítulo 4, um modelo de iluminação é um conjunto de equações para determinação da cor de um ponto, P, de uma superfície, quando iluminado pelas fontes de luz de uma cena. O modelo de iluminação de Phong é *local*, isto é, considera somente a iluminação *direta* em P, sem levar em conta a inter-reflexão entre objetos e a transparência. De acordo com o modelo de Phong, a cor, I, do ponto P é

$$I = O_a I_a + \sum_{l=1}^{\mathsf{NL}} [O_d I_l (-\mathbf{N} \cdot \mathbf{L}_l)] + \sum_{l=1}^{\mathsf{NL}} [O_s I_l (-\mathbf{R}_l \cdot \mathbf{V})^{n_s}], \tag{1}$$

cujos termos, explicados em sala, referem-se à iluminação ambiente, reflexão difusa e reflexão especular da luz direta, respectivamente. A cor I_l e o vetor $\mathbf{L}_l$ (e em consequência $\mathbf{R}_l$) dependem do tipo da l-ésima luz, $1 \leq l \leq \mathsf{NL}$. Dentre os vários tipos de luz virtual em computação gráfica, vimos em sala três: luz pontual, direcional e *spot*.

A tonalização de uma superfície consiste na determinação da cor de cada pixel correspondente a um ponto visível da superfície na imagem sendo renderizada. Uma vez que estamos usando uma malha de triângulos como modelo geométrico de uma superfície, a tonalização consiste, em OpenGL, na determinação da cor de cada um dos pixeis (mais especicamente, *fragmentos*, que podem ser entendidos como candidatos a pixeis) resultantes da *rasterização* de cada um dos triângulos da malha, como comentado em sala (veremos mais sobre rasterização no Capítulo 6). Vamos considerar dois *modos* de tonalização de triângulos. O primeiro, chamado tonalização de Gouraud, implementa o modelo de iluminação de Phong (ou outro modelo qualquer) no estágio de processamento de vértices. Assim, a cor em cada fragmento de um triângulo é definida pela interpolação das cores calculadas em cada um de seus três vértices, o que é feito automaticamente pela OpenGL durante o estágio de rasterização do triângulo (já vimos o efeito resultante de tal interpolação um uma discussão sobre *shader* de vértices efetuada em sala). O segundo modo, chamado tonalização de Phong, implementa o modelo de iluminação de Phong diretamente no processamento de fragmentos. Neste caso, o processamento de vértices deve calcular não a cor, mas a posição global, P, e a normal, $\mathbf{N}$, de cada vértice, as quais serão interpoladas pelo rasterizador para o cômputo da Equação (1) em cada fragmento.

1 Passo 1: Código-fonte e geração do executável

O código-fonte específico para este exercício está no diretório `cg/p3` do repositório `https://git.facom.ufms.br/pagliosa/cg.git`. O código comum a todas as partes está no diretório `cg/common` do repositório.

Após atualizar o código da parte comum e baixar o código da **Parte 3** em seu diretório de trabalho, cg conterá os diretórios common, p0, p1, p2 e p3.O conteúdo do diretório p3 será:

```
assets
  meshes
    bunny.obj
    f-16.obj
  shaders
    p3.fs
    p3.vs
build
  vs2019
    imgui.ini
    p3.sln
    p3.vcxproj
    p3.vcxproj.filters
Assets.cpp
Assets.h
Camera.cpp
Camera.h
Component.h
GLRenderer.cpp
GLRenderer.h
Light.h
Main.cpp
Material.h
P3.cpp
P3.h
Primitive.h
Renderer.cpp
Renderer.h
Scene.h
SceneEditor.cpp
SceneEditor.h
SceneNode.h
SceneObject.cpp
SceneObject.h
Transform.cpp
Transform.h
```

O diretório assets/meshes foi copiado da **Parte 2** e vem com exemplos de arquivos de malhas de triângulos no formato OBJ Wavefront. Como no exercício anterior, o diretório assets/shaders contém arquivos com o código GLSL dos *shaders* de vértice e fragmento e build/vs2019 o projeto do Visual Studio 2019. Os arquivos listados em verde foram copiados de p2 e não precisam ser alterados. Os arquivos listados em azul também foram copiados de p2, mas deverão ser substituídos com sua implementação da **Parte 2**. Desses, o único modificado foi Primitive.h (agora, um primitivo não contém mais uma cor, mas sim um material). A definição e implementação da classe de janela gráfica da aplicação são codificadas, respectivamente, em P3.h e P3.cpp. Além desses dois, você precisará alterar e completar Light.h, conforme descrito em sala.

No Visual Studio 2019, abra o arquivo `build/vs2019/p3.sln` e construa o arquivo executável. Você notará, entre outros gerados pelo Visual Studio, o arquivo `p3.exe` no diretório `p3`. A execução do programa exibe em sua tela uma janela gráfica cuja interface com o usuário é quase a mesma do exercício anterior. Como mostrado na Figura 1, a janela intitulada `Editor View Settings` apresenta novos controles destinados à seleção do tipo de tonalização de triângulos e cor e visibilidade de arestas em OpenGL. O botão `Create` da janela `Hierarchy`, por sua vez, apresenta opções para adição de objetos de cena contendo uma luz pontual, direcional ou *spot*, e a janela `Inspector` sugere controles para edição do material de um primitivo e algumas (não todas) propriedades de uma luz.

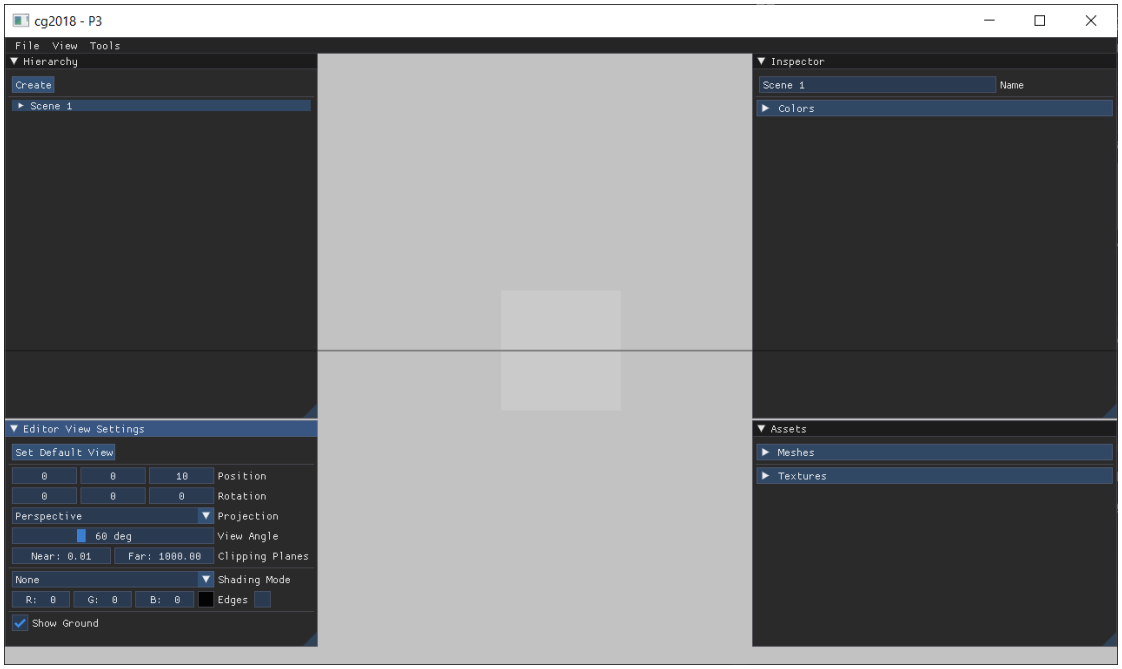

Figura 1: Programa `p3.exe` (no Windows).

2 Passo 2: Tarefas

Antes de começar as atividades específicas da **Parte 3** do trabalho, você deve primeiro integrar o código que você desenvolveu no exercício anterior com o código fornecido para este exercício.

Em seguida, você deveria completar a classe `Light`, em `Light.h`, com a declaração de atributos (e respectivos métodos de acesso) específicos de cada tipo de fonte luz (pontual, direcional e *spot*). Note que a classe `Light` deriva de `Component`. Assim, a posição de uma luz (quando esta for pontual ou *spot*) e sua direção (quando esta for direcional ou *spot*) já são dadas pela posição e rotação, respectivamente, do componente `Transform` do objeto de cena ao qual a luz pertence.

Feito isso, você já poderá criar luzes em suas cenas. Como uma luz é um componente, qualquer objeto de cena poderá conter uma luz. Há duas alternativas. A primeira é usar as novas opções do botão `Create`, na janela `Hierachy`, para criar um novo objeto de cena contendo, além de um `Transform`, uma luz pontual, direcional ou *spot*. A segunda alternativa é usar as opções do botão `Add Component`, na janela `Inspector`, para adicionar um novo componente — que pode ser um primitivo, luz ou câmera — ao objeto de cena, A, sendo inspecionado. Assim como feito com a câmera, se A contiver um componente que é

uma luz, este deverá ser mostrado na janela de vista. Para tal, você deverá implementar o método `P3::drawLight(l)`, em `P3.cpp`, em que `l` é uma referência para uma luz. Adote, em sua implementação, um desenho diferente para cada tipo de luz.

Com luzes na cena, passe à implementação dos modos de tonalização de triângulos. Para a tonalização de Gouraud, use o *shader* de vértices em `assets/shaders` como ponto de partida para codificar os termos restantes da Equação (1). Você vai precisar de mais variáveis uniformes, incluindo vetores e, possivelmente, estruturas, como discutido em sala. Para simplificar, adote NL ≤ 10. Para a tonalização de Phong, o modelo de iluminação deve ser codificado em um *shader* de fragmento. Assim, distintamente dos outros exercícios, você terá neste **dois** programas GLSL: um formado por extensões dos *shaders* `p3.vs` e `p3.fs`, e outro formado por outros dois *shaders*, digamos, `p3-smooth.vs` e `p3-smooth.fs`, os quais implementarão a tonalização de Phong. O programa GLSL responsável pela tonalização de Gouraud deverá ser usado na renderização no modo de edição de cena, enquanto que o programa GLSL responsável pela tonalização de Phong deverá ser empregado na renderização com OpenGL em `GLRenderer.cpp`.

As atividades de programação dessa parte do trabalho consistem em:

A1 Completar a implementação da classe `Light` com os atributos e métodos específicos de luzes pontuais, direcionais e *spot* (por exemplo, uma luz pontual ou *spot* tem um fator de decaimento, uma luz *spot* tem um ângulo de abertura, etc.). Deve ser possível a edição das propriedades de uma luz em `Inspector`,

A2 Implementar as funções da interface gráfica de criação de objetos de cena contendo uma luz, em `Hierarchy`, bem como de adição de componentes do tipo `Light` no objeto de cena sendo inspecionado, em `Inspector`.

A3 Implementar, em `P3.cpp`, o método `P3::drawLight(l)`, responsável pelo desenho de uma luz `l`. O método deverá ser invocado durante a renderização da cena no modo de edição (selecionado no menu `View`), sempre que o objeto de cena sendo inspecionado contiver um componente do tipo `Light`. Para tal, use as funções de ajuste de cor de linha e de desenho de linha do editor. Cada tipo de luz deverá ter um desenho distinto. Se preferir, você pode mover a definição e implementação deste método de `P3` para a classe `SceneEditor`.

A4 Implementar um programa GLSL de tonalização de Gouraud com o modelo de iluminação de Phong, para até dez fontes de luz. Este deverá ser o programa usado para renderização da cena no modo de edição, tal como vista pela câmera do editor de cena.

A5 Implementar um programa GLSL de tonalização de Phong com o modelo de iluminação de Phong, para até dez fontes de luz. Este deverá ser o programa usado por `GLRenderer` para renderização da cena.

A6 Criar uma cena inicial com luzes que demonstrem que a implementação do modelo de iluminação de Phong está funcionando.

Bônus: você notará que há mais dois modos de tonalização que podem ser selecionados na janela `Editor View Settings`: `None` (os triângulos não são tonalizados) e `Flat` (cada triângulo é tonalizado com a cor de seu centroide, calculada de acordo com o modelo de iluminação de Phong). A implementação de ambos ou de um desses modos, o que poderá requerer um terceiro programa GLSL, bem como o desenho das arestas dos triângulos, como explicado em sala, valerá pontuação extra no exercício.

3 Passo 3: README

Como no exercício anterior, o arquivo `README` deve conter o nome(s) do(s) autor(es) e uma descrição de como gerar (caso o Visual Studio 2019 não tenha sido utilizado) e executar o programa. Em seguida, você deve descrever quais as atividades você conseguiu ou não implementar, parcial ou totalmente, além de um breve manual do usuário (por exemplo, se é preciso usar alguma tecla para alguma funcionalidade para a qual não há ajuda textual na interface gráfica com o usuário). Descreva também quaisquer outras extensões que você implementou por iniciativa própria, as quais podem valer pontuação extra em sua nota do exercício.

4 Passo 4: Entrega do programa

O exercício deverá ser entregue via AVA em arquivo único compactado (somente um arquivo por grupo), chamado `p3.zip` (nome(s) do(s) autor(es) vão no arquivo `README`), contendo:

- o diretório `p3` com o código-fonte completo e com o arquivo executável `p3.exe`, mas **sem** quaisquer arquivos intermediários de backup ou resultantes da compilação, tanto em `p3` como em seus subdiretórios; e
- o arquivo `README`.

Não serão considerados e terão nota zero programas plagiados de qualquer que seja a fonte, mesmo que parcialmente, exceção feita ao código fornecido pelo professor.

5 Lista de objetivos

A verificação dos objetivos da **Parte 3** levará em conta se:

☐ O arquivo `p3.zip` foi submetido com os arquivos corretos, e o arquivo `README` contém as informações solicitadas.

☐ A classe `Light` foi implementada e a implementação está correta (as propriedades de cada tipo de luz são exibidas e podem ser editadas na janela `Inspector`).

☐ As funções em **A2** foram implementados e a implementação está correta (objetos de cena contendo uma luz, nomeados `Light 1`, `Light 2`, etc., podem ser adicionados à hierarquia de objetos de cena da cena inicial, e novos componentes do tipo `Light` podem ser adicionados a um objeto de cena).

☐ O desenho da luz pertencente ao objeto de cena sendo inspecionado foi implementado e está correto para cada diferente tipo de luz da cena inicial (por exemplo, para uma luz *spot* pode-se observar sua posição, direção, abertura, etc.).

☐ A tonalização de Gouraud foi implementada e a implementação está correta (o programa GLSL correspondente funciona no modo de edição de cena).

☐ A tonalização de Phong foi implementada e a implementação está correta (o programa GLSL correspondente funciona em *previews* da cena e no modo vista de renderização).