FACOM - CIÊNCIA DA COMPUTAÇÃO
COMPUTAÇÃO GRÁFICA - 2020.2
Relatório do Trabalho prático (CG)

- Integrantes:
  - Amanda Vieira - RGA 2018.1904.042-9
  - Julio Huang - RGA 2018.1904.056-9
- Linguagem escolhida:
  - C++
- OpenGL Version:
  - 3.1 Mesa 20.1.9 (Julio Huang)
  - 3.0 Mesa 20.0.8 (Amanda Vieira)
- OpenGLSL Version:
  - 1.40 (Julio Huang)
  - 1.30 (Amanda Vieira)

- Informações sobre os arquivos no zip:
  - Diretório `shader130`: contém todos os shaders;
  - Diretório `objs` : contém todos os arquivos .obj exportados do Blender;
  - Arquivos *headers* (.h) contendo as todas classes criadas;
  - Arquivo `app.cpp`: programa principal contendo a implementação completa da aplicação;

- Classes criadas para o armazenamento de estruturas, salvas em arquivos *headers*:
  - `axis.h` : armazena dados dos eixos de coordenadas;
  - `camera.h` : armazena dados da câmera;
  - `light.h` : armazena dados dos pontos de luz;
  - `objeto.h` : armazena dados para construção das formas geométricas;
  - `lerComando.h` : efetua a leitura da linha de comando.

- Parâmetros fixos (constantes) usados na implementação:
  - `#define MAX 50`: define o número máximo de uma *string* auxiliar usada para ler cada linha do arquivo.obj, na função `loadObj`.

- Estruturas globais criadas para manipulação de dados de cena:
  - Vetor global de classes para guardar os objetos adicionados à cena (`vector<objeto*> objetoVetor`);
  - Vetor global de classes para guardar as luzes adicionadas à cena (`vector<light*> lightVetor`);
  - Um ponteiro global para criação de um único objeto da classe “axis”, referente aos eixos de coordenadas, usado em toda a execução (`axis *axisScene`);
  - Um ponteiro global para criação de um único objeto da classe “camera”, referente à câmera, usado em toda a execução (`camera *cam`);
  - Uma Matriz View global 4x4 para guardar as mudanças nos dados do objeto cam;

- Um objeto global da classe “lerComando” para efetuar todas as leituras da linha de comando;
- Variáveis globais para os coeficientes de reflexão, inicializadas em 0.2f - `Ka, Kd, Ks`;
- Identificadores globais de VAOs e VBOs para cada estrutura a ser desenhada na cena (objetos, eixos e pontos de luz);
- Um inteiro global “`wire`”, inicializado com 0, usado para verificar se o desenho deve ser feito no modo wireframe (`wire` = 1), ou normal (`wire` = 0);
- Inteiros globais “`flat_on`”, “`smooth_on`”, “`phong_on`”, todos inicializados com 0, para verificar qual o modo de *shading* deve ser utilizado para renderizar a cena. Isso ocorrerá quando o valor da variável correspondente for igual a 1. Quando as 3 variáveis estão zeradas, o programa utiliza o *shading* NONE como padrão;
- Um inteiro global “`lights_on`” para verificar quando os pontos de luz devem ser desenhados na cena (`lights_on` = 1 desenha, `lights_on` = 0 não desenha);
- Um inteiro global “`axis`” para verificar quando os eixos de coordenadas devem ser desenhados na cena (`axis_on` = 1 desenha, `axis_on` = 0 não desenha);

- Funções auxiliares criadas:
  - **`void loadObj(`**`const char *path, vector<glm::vec3> &vbuffer`**`);`**
    Pertencente à classe OpenGLContext, realiza a leitura e processamento dos arquivos *.obj,* construindo o *buffer* que será passado aos *shaders* como VBOs. O *buffer* construído é o mesmo que tem o endereço passado como parâmetro.
    - **`const char *path`:** parâmetro com o caminho do arquivo.obj a ser lido;
    - **`vector<glm::vec3> &vbuffer`:** endereço do atributo `vector<glm::vec3> vertexBuffer` da classe “objeto”.

  - **`void initialize();`**
    Pertencente à classe OpenGLContext, realiza, de acordo com a leitura da linha de comando, a preparação de todos os dados necessários para passagem aos *shaders* e renderização da cena: construção de VAOs/VBOs para adição/remoção de estruturas à cena (objetos, eixos, ou pontos de luz), mudanças na câmera padrão, construção das matrizes de transformação, adição/remoção de reflexões, e escolha de *shadings.*

  - **`void rendering();`**
    Pertencente à classe OpenGLContext, renderiza a cena com os dados carregados na função `initialize()`. No caso dos objetos, os shaders usados são escolhidos com base nos *shadings* correspondentes, informados pela linha de comando.

- Sobre os *shaders*:
  - Foram criados *vertex* e *fragment shaders* para objetos, eixos e pontos de luz, separadamente. Para os eixos e pontos de luz:
    - axis.vp e axis.fp: implementam as coordenadas X, Y e Z;
    - light.vp e light.fp: implementam as luzes;

- ➢ Para os objetos, foram criados 4 tipos de *shaders*, de acordo com os *shadings*:
  - none.vp e none.fp: implementam os objetos sem iluminação;
  - flat.vp e flat.fp: implementam os objetos com a iluminação no modo *flat*;
  - smooth.vp e smooth.fp: implementam os objetos com a iluminação no modo *smooth*;
  - phong.vp e phong.fp: implementam os objetos com a iluminação no modo *phong*;
- ➢ Foram criadas algumas funções auxiliares para criação e carregamento de cada tipo de *shader*, de acordo com o que se quer desenhar na cena a cada redesenho:
  - `void createShadersNone();`
  - `void createShadersFlat();`
  - `void createShadersSmooth();`
  - `void createShadersPhong();`
  - `void createShadersLight();`
  - `void createShadersAxis();`

- ❖ Informações para compilação e execução:
  - ➢ Linux:
    - compição: g++ --std=c++11 *.cpp -o main -lGLEW -lGL -lGLU -lglut
    - execução: ./main

  - ➢ Windows:
    - g++ --std=c++11 *.cpp -o main -lglew32 -lfreeglut -lglu32 -lopengl32
    - execução: ./main

- ❖ Outras informações:
  - ➢ *Shaders* usam versão 130