Roteiro de Laboratório – OpenGL 3D

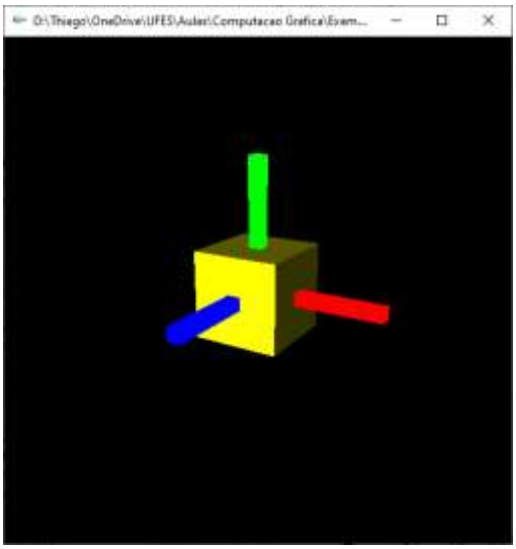
Objetivo: Trabalhar os conceitos transformações 3D, câmeras e luz.

Material: Sistema operacional Linux (recomendação Ubuntu 20.04) com g++, OpenGL e GLUT instalados (geralmente só são necessários dois comandos para a instalação, https://gist.github.com/AbdullahKady/f2782157991df652c2baee0bba05b788).

Antes de iniciar o roteiro, deve-se assistir o vídeo de OpenGL 3D!

1. Tarefa:

Criar uma aplicação para mostrar um cubo e eixos do sistema de coordenadas usando diferentes configurações de luz e câmera. Ver exemplo na figura abaixo!



1.1. Passo: criação da janela

- Compilar os arquivos fornecidos com este roteiro com o comando "g++ -o test opengl3d.cpp -lGL -lGLU -lglut". As bibliotecas OpenGL, GLU e GLUT são referenciadas respectivamente com os comandos -lGL -lGLU -lglut.
- Executar o arquivo compilado com o comando "./test"
- Verificar se uma janela foi criada com o fundo preto

1.2. Passo: estudo sobre transformações 3D

- A chamada de *DrawAxes* na função *display* desenha os três eixos do sistema de coordenadas (r, g e b para x, y e z respectivamente) no sistema de coordenadas do mundo. O tamanho dos eixos é passado como parâmetro.
    - Comentar essa função, compilar e rodar para ver os eixos sumirem
    - Veja o código utilizado para criar os eixos para ver se faz sentido
    - Retorne com a função
- A chamada de *DrawObj* na função *display* desenha um cubo amarelo na origem do sistema de coordenadas do mundo. O tamanho do cubo é passado como parâmetro.
    - Troque as cores do material do cubo e veja como ele é afetado.

- Faça uma chamada de *glRotatef(45, 0,1,0);* antes de desenhar o objeto e veja o resultado
    - O objeto deve girar em torno do eixo verde (eixo y)
    - Brinque com outras transformações 3D, desenhe outros objetos (por exemplo com *glutSolidSphere(size, 20, 10);*) e eixos de coordenadas para ver se seu pensamento faz sentido
    - Quando estiver testando a esfera, teste os diferentes tipos de shading (troque GL_FLAT por GL_SMOOTH)

1.3. Passo: estudo sobre a luz

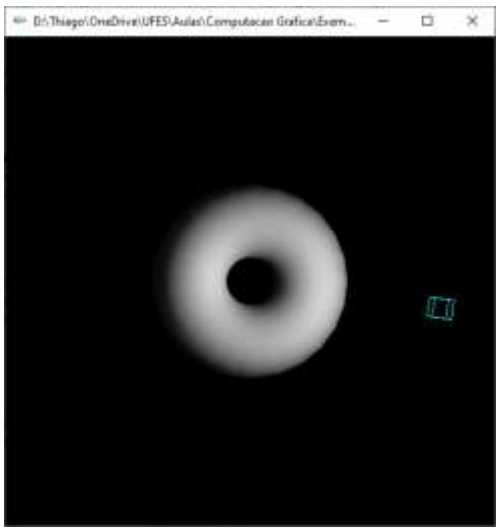
- A função *glLightfv(GL_LIGHT0, GL_POSITION, light_position);* na display é responsável por colocar a fonte de luz 0 na posição informada em *light_position*.
    - Perceba que ela está posicionada no plano formado pelo eixo z e y, porém mais próxima do eixo z. Com isso, a face perpendicular ao eixo z fica mais iluminada.
    - Faça alterações na posição da luz e veja se faz sentido.
    - Transformações (como rotação, translação e escala) também afetarão a posição da fonte de luz.
- Coloque a fonte de luz ao longo do eixo z e desenhe um outro cubo entre o primeiro e a fonte de luz
    - Perceba que os dois serão iluminados igualmente, pois não existe o conceito de sombras entre objetos. Somente o caminho fonte-objeto-olho é considerado.
- Comente a linha *glEnable(GL_LIGHTING);* na *init*
    - Perceba que o objeto mudou de cor. Por quê?
    - Perceba que ele perdeu as arestas internas. Por quê?

1.4. Passo: estudo sobre câmera

- A função *gluLookAt(3,2,5, 0,0,0, 0,1,0);* na display é responsável por colocar a câmera na posição x=3, y=2 e z=5, olhando para a origem do sistema de coordenadas (0, 0 e 0) e com o up na direção de y (0, 1 e 0).
    - Mude a posição da câmera para ver como isso afeta a visualização
    - Mude a direção para onde ela olha
    - Mude o up
- Coloque a câmera no plano z e y, por exemplo, na posição 0, 2 e 5 e desabilite a luz.
    - Perceba que o efeito 3D interno sumiu, mas a silhueta ainda é afetada pela câmera perspectiva.
    - Troque a câmera perspectiva pela câmera paralela. Comente as linhas referentes a *gluPerspective* e descomente a *glOrtho*. Qual deveria ser a aparência do cubo? Pense antes de fazer e veja se fez sentido.

2. Tarefa:

Criar uma aplicação para mostrar a luz mudando em uma toroide. Ver exemplo na figura abaixo!



2.1. Passo: criação da janela

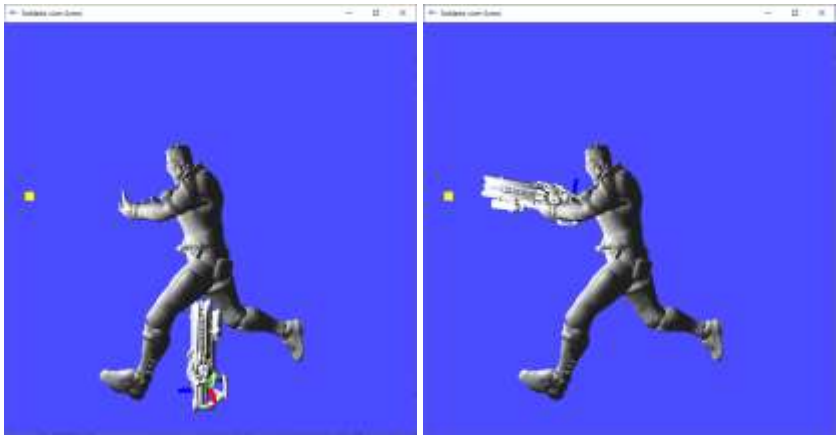
- Compilar os arquivos fornecidos com este roteiro com o comando “g++ -o test lightToroide.cpp -lGL -lGLU -lglut”. As bibliotecas OpenGL, GLU e GLUT são referenciadas respectivamente com os comandos -lGL -lGLU -lglut.
- Executar o arquivo compilado com o comando “./test”
- Verificar se uma janela foi criada com o fundo preto
- Estude o que o código faz.

2.2. Passo: efeitos da iluminação no wirecube

- Perceba o efeito de comentar as *glDisable (GL_LIGHTING);* e *glEnable (GL_LIGHTING);* na *display*.
    - Isso vai fazer com que o material seja aplicado também no wirecube o que geralmente não é desejável, portanto desabilitamos a iluminação e aplicamos uma cor básica.

3. Tarefa:

Criar uma aplicação para mostrar um soldado e uma arma. Inicialmente, o soldado e a arma estarão nos seus próprios sistemas de coordenadas e o nosso objetivo é calcular a transformação que coloca a arma na mão do soldado. Ver exemplo nas figuras abaixo, sendo a da esquerda sem a transformação ser aplicada e a da direita com a transformação aplicada! Observe que essa transformação deverá ser calculada de forma genérica em função dos vértices da malha do soldado. Ou seja, você deverá montar um sistema de coordenadas na mão do soldado e montar a matriz de transformação conforme a aula de transformações genéricas. Não deve usar funções trigonométricas (seno, coseno, ...) e nem deve fazer a transformação manualmente em um programa externo.



Funcionamento geral da aplicação (já implementado):

- Apertar as teclas *1* e *2* alterna respectivamente entre não aplicar e aplicar a transformação que coloca a arma na mão do soldado. Observar que a transformação ainda deve ser calculada por vocês, portanto clicar o *2* não terá efeito inicialmente
- Clicar com o botão direito do mouse e arrastar muda a posição da câmera
- Clicar a tecla *a* alterna entre desenhar a arma ou não
- Clicar a tecla *c* alterna entre desenhas os eixos que representam o sistema de coordenadas ou não.
- Clicar as teclas *+* e – alteram o nível de zoom da aplicação.
- Clicar a tecla *m* alterna entre usar a função *gluLookAt* da biblioteca GLU (o fundo ficará azul) e a função implementada a ser implementada *MygluLookAt* (o fundo ficará vermelho). Após implementada, a alternância entre as duas funções não deverá causar diferenças, exceto a cor do fundo que foi implementada para diferenciar o estados.
- Clicar a tecla *s* carrega alterna entre duas meshes do soldado, sendo cada uma em um sistema de coordenadas diferente.

3.1. Passo: criação da janela

- Compilar os arquivos fornecidos com este roteiro com o comando “g++ -o test soldado.cpp objloader.cpp -lGL -lGLU -lglut”. As bibliotecas OpenGL, GLU e GLUT são referenciadas respectivamente com os comandos -lGL -lGLU -lglut.
- Executar o arquivo compilado com o comando “./test”

- Verificar se uma janela foi criada corretamente com o soldado e a arma

3.2. Passo: cálculo da transformação da arma

- Implementar a função de troca de sistema de coordenadas *ChangeCoordSys* que aplica a transformação para colocar a arma na mão do soldado. O sistema de coordenadas posicionado na mão do soldado será descrito pelos parâmetros da função, isto é, dois pontos (A e B) e um vetor (UP). O ponto A indica para onde a arma aponta, o ponto B onde a arma está e o vetor UP como a mira da arma está orientada.
    - Usar o programa MeshLab para identificar a orientação dos objetos (soldado e arma) em relação aos seus próprios sistemas de coordenadas e para obter o ID dos pontos da mesh que definirão a posição da arma e para onde ela aponta. O ícone *i* (Get Info) permite clicar na mesh e ver ID e posição dos vértices da mesh. Colocar os ID desejados nas variáveis *pontoArmaAponta* e *pontoPosicaoArma*.
    - Definir o vetor UP da arma, na variável *up[3]*, conforme desejado.
    - Fazer o código da *ChangeCoordSys* que aplica a transformação necessária.
        - Vai precisar montar a matriz a ser multiplicada
        - A matriz final calculada será aplicada na pilha de transformações (GL_MODELVIEW) com a função *glMultMatrixf*.
        - Usar funções auxiliares *normalize* e *cross* para fazer respectivamente a normalização de um vetor e o produto vetorial de dois vetores.
    - Se a transformação funcionar, a arma deverá permanecer na mão do soldado ao clicar a tecla *s*. Ou seja, a transformação da arma será relativa aos pontos da mesh do soldado e, ao fazer para uma das meshes, deve automaticamente funcionar para a outra.
    - Após fazer funcionar, teste com outros pontos.

3.3. Passo: cálculo da transformação da câmera

- Perceba que a câmera foi implementada com a função *gluLookAt*. Para se obter o efeito de giro com o botão direito, considerou-se que a posição da câmera estava na superfície de uma esfera de raio *zoom* (indicado para identificar a distância da câmera para o ponto de interesse no centro da esfera). Uma posição na superfície da esfera é identificada por dois ângulos, um para girar para cima e para baixo e outro para girar horizontalmente, ambos em torno do centro de atenção. O cálculo da posição é feito utilizando-se senos e cossenos dos ângulos.
- Implementar a função de troca de sistema de coordenadas *MygluLookAt* que aplica a transformação para visualizar o mudo de uma posição (definida pelo ponto *eye*), um local para onde está olhando (definida pelo ponto *center*) e uma orientação em torno de seu vetor direcional (definido pelo vetor *up*). A função deve produzir um efeito idêntico ao da função *gluLookAt*.
    - Calcular e aplicar a transformação que leva do sistema de coordenadas da câmera padrão do OpenGL para o definido pela câmera do usuário da função.
        - Vai precisar montar a matriz a ser multiplicada
        - A matriz final calculada será aplicada na pilha de transformações (GL_MODELVIEW) com a função *glMultMatrixf*.

- Perceba que as transformações básicas (de rotação, escala e translação) continuam valendo e podem compor transformações mais complexas juntamente com a função *glMultMatrixf.*