FURG
Jaler Samuel Zago - 108352
Rafael Escher - 110666

OpenGL - Parada de Ônibus da FURG

05 Novembro de 2019

INTRODUÇÃO

Foi proposto, em aula, o desenvolvimento de um objeto 3D que pudesse representar o uso das ferramentas de computação gráfica presentes na biblioteca OpenGL. O OpenGL é uma API (Application Programming Interface) livre, lançado em 1992, desenvolvido e mantido pelo Khronos Group.

Utilizando essa biblioteca, propõe-se a criação de um cenário contendo uma parada de ônibus da FURG, buscando a maior semelhança da realidade possível, apresentando as ferramentas e técnicas utilizadas no programa, e a forma em que foram utilizadas.

OBJETIVOS

Objetivo Geral

Utilizar a biblioteca OpenGL e criar geometrias referenciando algum objeto, cenário, figura, prédio, etc, relacionado à FURG. Para este trabalho, foi escolhido criar uma parada de ônibus da FURG. Para o desenvolvimento, a imagem abaixo foi utilizada como guia.

Figura 1: Fotografia da parada de ônibus da FURG, situada a frente do pórtico de entrada à universidade.

Objetivos Específicos

Utilizar o máximo de técnicas e tecnologias possíveis oferecidas pela biblioteca OpenGL, como:

- Transformações geométricas: escala, rotação e translação;
- Projeções geométricas;
- Visibilidade;
- Rasterização;
- Iluminação;
- Blending;
- Animação;
- Textura;
- Sombreamento.

TÉCNICAS E TECNOLOGIAS UTILIZADAS

Transformações geométricas

O usuário tem a opção de rotacionar o cenário no eixo X e Y, a fim de observar de todos os ângulos a parada de ônibus, ao apertar, respectivamente, as setas DIREITA ou ESQUERDA, e CIMA ou BAIXO.

Também a função de rotação foi utilizada para construir cilindros que funcionam como um ‘suporte físico’ dos painéis transparentes da parada de ônibus e do banco de madeira.

Projeção Geométrica

No programa, optou-se por utilizar a projeção do tipo perspectiva, com ângulo de visão de 45º, *zNear* de 0.4, e *zFar* de 500. Declarou-se a projeção da forma:

```
gluPerspective(angle, fAspect, 0.4, 500);
```

O *fAspect*, ou só *aspect*, é reajustado conforme tamanho da janela, mas idealmente é 1.

Visibilidade

A fim de se obter uma visibilidade mais realista, foi utilizado o *DEPTH BUFFER*. Com ele, o programa consegue identificar, em cada pixel do cenário, qual a cor do objeto que deve ser mostrada ao usuário, resultando em ‘objetos na frente de outros’, ou seja, as partes de objetos que estão atrás de outro, ou partes traseiras de objetos não serão renderizadas.

A utilização do *DEPTH BUFFER,* na *main()* do programa, foi declarada da forma:

```
glutInitDisplayMode ( GLUT_DEPTH );
```

E seu uso é inicializado da forma:

```
glEnable( GL_DEPTH_TEST );
```

Não julgou-se necessária a utilização da técnica *back-face culling*, pois o uso do *depth buffer* já atendeu todas as necessidades no quesito de visibilidade. Mas caso contrário, sua ativação não é nada complicada, apenas um par de linhas de código já é suficiente para isso. Inclusive, retas normais em cada face dos polígonos foram geradas em caso de utilização do *back-face culling*.

Rasterização

Foi utilizado, no programa, a técnica de rasterização também chamada de *Multi-Sampling Anti-Aliasing*. Para isso, primeiramente declara-se a utilização do buffer na função *main(),* da forma:

```
glutInitDisplayMode ( GLUT_MULTISAMPLE );
```

Por fim, habilita-se o multisampling na inicialização do programa, da forma:

```
glEnable( GL_MULTISAMPLE_ARB );
```

Após essas duas etapas, a rasterização será aplicada automaticamente pelo programa.

Iluminação

A iluminação do programa é configurada na função *initLights()*, chamada na função *init()*. Diversos valores de iluminação são então configurados, como luz difusa, luz especular, especularidade do material, entre outros.

A iluminação em geral é inicializada da forma:

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

As configurações citadas logo acima, foram definidas para o único ‘foco de luz’ utilizado, sendo ele o *GL_LIGHT0*.

Blending

Utilizou-se o *blending* para criar o efeito de transparência nos painéis da parada de ônibus. A ideia é misturar/juntar (blend) a cor do painel com os objetos que estão atrás do mesmo, de forma a dar a impressão ao observador de estar olhando através de um vidro.

Não é preciso declarar sua utilização na inicialização do programa. Deve-se apenas ativá-la antes da construção do elemento em que se quer obter transparência, e desativá-la logo em seguida, da forma:

```
glEnable (GL_BLEND);
glBlendFunc (GL_ONE_MINUS_DST_COLOR, GL_ONE);
glDepthMask (GL_FALSE);
// cria objeto
glDisable (GL_BLEND);
glDepthMask (GL_TRUE);
```

Após ativa, deve-se declarar qual o tipo de blending do objeto, dividida em dois parâmetros: *source* e *destination*. Os parâmetros foram escolhidos da forma que melhor aparentou.

Animação

A animação no programa é dependente do usuário. Ele controla a rotação do cenário utilizando os botões direcionais do teclado. Os botões de *direita* e *esquerda* rotacionam o cenário no eixo Y, e os botões de *cima* e de *baixo* rotacionam no eixo X.

IMPLEMENTAÇÃO

Base

A base do cenário nada mais é do que um 'quadrado tridimensional', ou seja, 2 quadrados bidimensionais foram desenhados, e também 4 retângulos bidimensionais, utilizando o método *GL_QUADS*, da forma:

```
void chao(float compxz, float alty){
     glColor3f(0.1f, 0.4f, 0.1f);
     glBegin(GL_QUADS);                  // Face frontal
          glNormal3f(0.0, 0.0, 1.0);   // Normal da face
          glVertex3f(compxz/2, 0, compxz/2);
```

```
            glVertex3f(-compxz/2, 0, compxz/2);
            glVertex3f(-compxz/2, -alty, compxz/2);
            glVertex3f(compxz/2, -alty, compxz/2);
     glEnd();
     ...
 }
```

Faces Laterais Transparentes

As faces laterais são compostas de 3 painéis quadrados azuis transparentes, e cada quadrado foi criado da seguinte forma, mudando apenas as coordenadas de cada um:

```
void facesLaterais_parada(float distancia, float tamanho){
    glEnable (GL_BLEND);                // inicia blend
    glDepthMask (GL_FALSE);             // desliga uso do depthbuffer
    glBlendFunc (GL_ONE_MINUS_DST_COLOR, GL_ONE);
    glBegin(GL_QUADS);                  // Face lateral esquerda
        glNormal3f(-1.0, 0.0, 0.0);     // Normal da face
        glVertex3f(-distancia/2, 0, 0);
        glVertex3f(-distancia/2, 0, -tamanho);
        glVertex3f(-distancia/2, tamanho, -tamanho);
        glVertex3f(-distancia/2, tamanho, 0);
    glEnd();
    glDisable (GL_BLEND);               // finaliza blend
    glDepthMask (GL_TRUE);              // reativa depth buffer
    ...
}
```

Nesta face foi aplicada a técnica de *blending*, necessitando sua ativação e desativação antes de criar o objeto, e após criá-lo, respectivamente, como explicado mais acima.

Face Superior Curva

A face superior curva envolve toda a parada de ônibus, e é a parte mais complexa do cenário. Para sua criação, foi necessário a combinação das funções ***glMap2f()***, para definir a utilização de um 'mapeamento de vertex', ***glMapGrid2f()***, para definir um preenchimento da superfície gerada, ambas utilizadas na função de inicialização do programa, ***glEvalCoord2f()***, para criar um *evaluator* de *line_strip* ao longo dos pontos da superfície, e por fim ***glEvalMesh2()***, para ativar o preenchimento da superfície gerada, ambas as últimas utilizadas dentro da função *display()* do programa.

Um dos parâmetros da primeira função é um ponteiro para uma estrutura de coordenadas, que define como vai se comportar a curvatura e posição da face:

```
GLfloat curvedSurface[4][4][3] = {
   // curva 1
   {{-40, 0, -15}, {-40, 20, -40},
    {-40, 70, -15}, {-40, 45, 20}},
   // curva 2
   {{-40, 0, -15}, {-40, 20, -40},
    {-40, 70, -15}, {-40, 45, 20}},
   // curva 3
   {{40, 0, -15}, {40, 20, -40},
    {40, 70, -15}, {40, 45, 20}},
   // curva 4
   {{40, 0, -15}, {40, 20, -40},
    {40, 70, -15}, {40, 45, 20}}
};
```

Essa estrutura possui 4 conjuntos de coordenadas, ou 4 'curvas', e cada uma delas é composta por 4 pontos.

A ideia geral do funcionamento desse conjunto de funções acima mencionado, é que em cada curva acima definida, uma linha será ligada entre o primeiro ponto e o último ponto, e os 2 pontos intermediários controlarão a curvatura dessa linha, puxando a linha em

direção à eles. Assim, como são 4 curvas, 4 linhas curvas serão geradas, e então, utilizando a mesma função ***glEvalCoord2f()***, mais 4 linhas serão geradas na ortogonal das anteriores, a fim de se criar uma 'grade', da forma:

```
void faceCurva(){
    ...
    for (int j = 0; j <= 3; j++) {
        glBegin(GL_LINE_STRIP);
            for (int i = 0; i <= 30; i++)
                glEvalCoord2f((GLfloat)i/30.0, (GLfloat)j/2.0);
        glEnd();
        glBegin(GL_LINE_STRIP);
            for (int i = 0; i <= 30; i++)
                glEvalCoord2f((GLfloat)j/3.0, (GLfloat)i/30.0);
        glEnd();
    }
    ...
}
```

Tendo sido criado a grade de forma a 'moldurar' a curvatura, precisa-se preenchê-la de cor. Para isso, ainda na função *faceCurva()*:

```
glEvalMesh2(GL_FILL, 0, 20, 0, 20);
```

Nesta face foi aplicada a técnica de *blending*, necessitando sua ativação e desativação antes de criar o objeto, e após criá-lo, respectivamente, como explicado mais acima.

Banco

O banco representado no programa, da mesma forma que a base, é um conjunto de figuras 2Ds. A partir de 6 retângulos bidimensionais, é criada a ilusão de uma imagem 3D, onde o interior é oco. O método *GL_QUADS* é utilizado para a representação da seguinte forma:

```
void banco(){
    ...
    glBegin(GL_QUADS);          // Face frontal
        glNormal3f(0.0, 0.0, 1.0);  // Normal da face
        glVertex3f(x, y, z);
        glVertex3f(-x, y, z);
        glVertex3f(-x, y2, z);
        glVertex3f(x, y2, z);
    glEnd();
    ...
}
```

Suportes Cilíndricos

Os suportes cilíndricos, na estrutura física, são barras metálicas brancas que suportam os painéis acrílicos e o banco de madeira da mesma. No programa optou-se pela criação de cilindros finos para este fim.

Para a criação de tais cilindros, primeiro é necessária a declaração de um objeto quádrico:

```
GLUquadric* qobj;
```

Então, na variável criada, atribui-se um objeto quádrico novo, e define-se uma suavização para o mesmo:

```
qobj = gluNewQuadric();

gluQuadricNormals(qobj, GLU_SMOOTH);
```

Por fim, para a criação de um cilindro, deve-se utilizar a seguinte função, passando como parâmetros o objeto quádrico criado, o raio superior e inferior, a altura do cilindro, e o número de ‘iterações’ que terá a renderização deste cilindro (poucas iterações não deixam a impressão de redondo):

```
gluCylinder(qobj, r, r, h, 100, 100);
```

Para a criação de tais estruturas cilíndricas utilizadas no cenário, criou-se uma função, que recebe os parâmetros **x**, **y** e **z**: para transladar o cilindro criado até essa posição; **r**: raio superior e inferior do cilindro; **h**: a altura do cilindro; **a:** ângulo desejado para o cilindro; e **da**: direção do ângulo (**a=90** e **da=1** deixam-o na vertical, ou seja, rotaciona o cilindro 90° no eixo x).

```
void cilindro(float x, float y, float z, float r, float h, float a, int
da){

    /* x,y,z: posições desejadas | r: raio | h: altura | a: angulo | da:
direção do angulo (x:1,y:2,z:3) */

    glColor3f(1.0f, 1.0f, 1.0f);

    glPushMatrix ();

        // define direção do cilindro (vertical/horizontal)

        switch (da) {

        case 1:

            glRotatef(a, 1.0, 0.0, 0.0);

            break;

        case 2:

            glRotatef(a, 0.0, 1.0, 0.0);

            break;

        case 3:

            glRotatef(a, 0.0, 0.0, 1.0);

            break;

        }

        glTranslatef(x, y, z);

        // cria cilindro

        gluCylinder(qobj, r, r, h, 100, 100);

    glPopMatrix ();

}
```

RESULTADOS

Após a implementação dos algoritmos elencados anteriormente, obteve-se a representação 3D de uma parada de ônibus localizada na FURG, cumprindo o objetivo deste trabalho. A representação estática da figura pode ser vista a seguir. Para uma melhor visualização, segue em anexo a codificação do projeto.

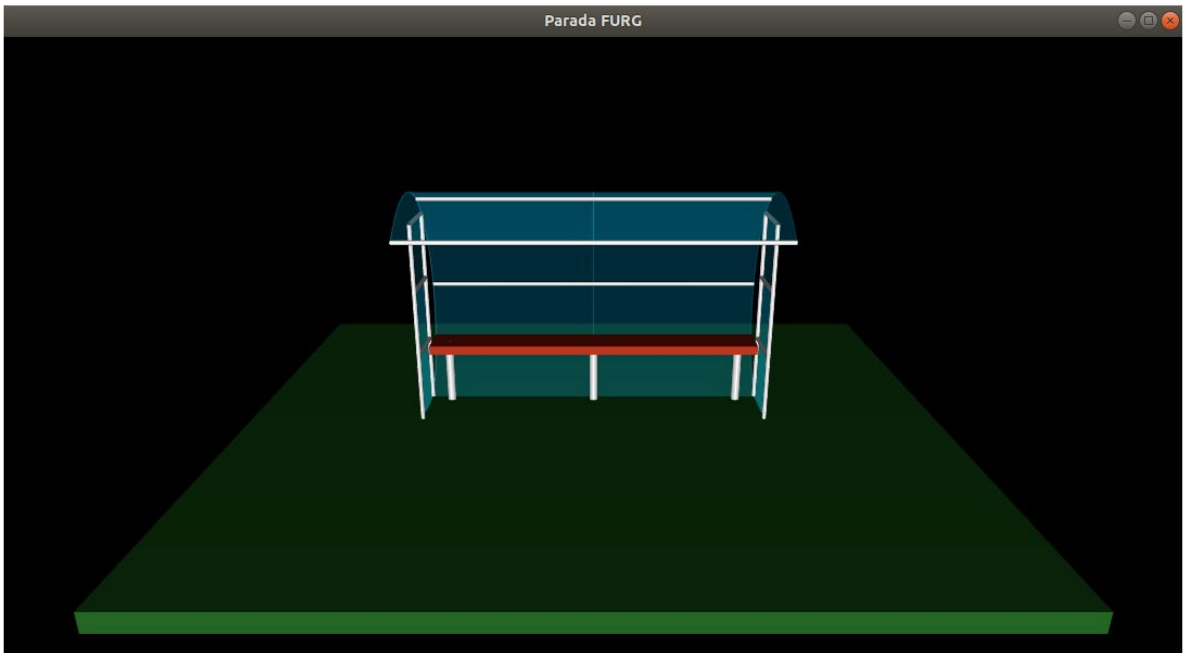

Figura 2: Print Screen capturado durante execução do algoritmo presente neste trabalho.

Interação com o programa

Durante execução do programa, o usuário consegue interagir com o mesmo, tanto com botões do teclado, ou com o mouse. São eles:

- **Mouse:**
  - **Botão direito:** Ao clique do botão direito do mouse, no cenário ocorre um '*zoom-out*';
  - **Botão esquerdo:** Ao clique do botão esquerdo do mouse, no cenário ocorre um '*zoom-in*'.
- **Teclado:**
  - **Escape:** Ao clique do botão do teclado 'Escape', o programa se encerra;

- **F:** Ao clique do botão do teclado ‘f’, as luzes do programa se desligam;
- **G:** Ao clique do botão do teclado ‘g’, as luzes do programa se ligam;
- **R:** Ao clique do botão do teclado ‘r’, o cenário rotacionado volta à posição original;
- **Tecla direcional superior:** Ao clique do botão do teclado ‘para cima’, o cenário se rotaciona no eixo x;
- **Tecla direcional inferior:** Ao clique do botão do teclado ‘para baixo, o cenário se rotaciona no eixo -x;
- **Tecla direcional direita:** Ao clique do botão do teclado ‘para direita, o cenário se rotaciona no eixo y;
- **Tecla direcional esquerda:** Ao clique do botão do teclado ‘para esquerda, o cenário se rotaciona no eixo -y;

REFERÊNCIAS

[1] Isabel Harb Manssour. Introdução à OpenGL [Online]. Editado em: [Agosto de 2005]. Disponível em: http://www.inf.pucrs.br/~manssour/OpenGL/Tutorial.html. Acesso em: 14 de out. 2019.

[2] The Official Guide to Learning OpenGL, Version 1.1. OpenGL Programming Guide. [s.d.]. Disponível em: https://www.glprogramming.com/red/index.html. Acesso em: 17 de out. 2019.