Relatório Técnico - OpenGL

Willian Lemos - 87708
Carlos Nascimento - 85489

1 - Escolha:

Como o objetivo da atividade é desenvolver um ambiente virtual em OpenGL contendo geometrias que façam referência à FURG, escolhemos o próprio logo da FURG já que há mais familiaridade.

2 - Textura

Para a construção da textura, precisamos utilizar 2 códigos auxiliares, que são os Loaders. O primeiro, **imageLoader.h**, funciona como uma biblioteca que auxilia na leitura do bitmap de imagem. O segundo código, **imageLoader.cpp**, acessa a biblioteca e performa uma série de operações como conversão e leitura de bytes e pixels. Também há uma função no arquivo principal (**main2.cpp**) chamada **LoadTexture** na qual são pré-carregados os parâmetros de conversão da imagem a ser mapeada.

```
GLuint loadTexture(Image* image) {
    GLuint textureId;
    glGenTextures(1, &textureId);
    glBindTexture(GL_TEXTURE_2D, textureId);

    glTexImage2D(GL_TEXTURE_2D,
                 0,
                 GL_RGB,
                 image->width, image->height,
                 0,
                 GL_RGB,
                 GL_UNSIGNED_BYTE,

                 image->pixels);
    return textureId;
}
```

Figura 1: Função de construção da textura.

Após isso, as texturas só precisam ser habilitadas e mapeadas para as coordenadas desejadas de acordo com o id da textura declarado anteriormente, de acordo com as funções abaixo:

```
// Fundo
glEnable(GL_TEXTURE_2D);
glPushMatrix();
    glBindTexture(GL_TEXTURE_2D, _texturafundo);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTranslatef(0,0,-10);
    glBegin(GL_QUADS);
        glTexCoord3f(0.0,10.0,0.1);  glVertex3f(-20,20,0);
        glTexCoord3f(10.0,10.0,0.1);  glVertex3f(20,20,0);
        glTexCoord3f(10.0,0.0,0.1);  glVertex3f(20,-20,0);
        glTexCoord3f(0.0,0.0,0.1);  glVertex3f(-20,-20,0);
    glEnd();
    glDisable(GL_TEXTURE_2D);
glPopMatrix();
```

Figura 2: Função de habilitação e mapeamento da textura.

3 - Iluminação:

Em OpenGL considera-se que a luz é dividida em quatro componentes independentes, que são: Ambiente, Difusa, Especular e Emissiva. Como o material de uma superfície é caracterizado pela porcentagem dos componentes RGB que chegam e são refletidos em várias direções, procuramos balancear estes parâmetros de forma que a imagem seja visível e mantenha as propriedades de cor e iluminação desejadas.

```
GLfloat luzAmbiente[4]={r,g,b,1.0};
GLfloat luzDifusa[4]={r,g,b,1.0};      // "cor"
GLfloat luzEspecular[4]={r, g, b, 1.0};// "brilho"
GLfloat posicaoLuz[4]={0.0, 0.0, 0.0, 1.0};

// Capacidade de brilho do material
GLfloat especularidade[4]={1.0,1.0,1.0,1.0};
GLint especMaterial = 1;
glMateriali(GL_FRONT_AND_BACK,GL_SHININESS,especMaterial);

// Ativa o uso da luz ambiente
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, luzAmbiente);
glLightfv(GL_LIGHT0, GL_SPECULAR, luzEspecular );
glLightfv(GL_LIGHT0, GL_POSITION, posicaoLuz );

// Habilita o modelo de colorização de Gouraud
glShadeModel(GL_SMOOTH);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

Figura 3: Funções de iluminação.

4 - Geometria

A geometria da figura consiste em um cilindro alaranjado e cubos com escalas,posições e cores diferentes, criando um só sólido formado por sólidos sobrepostos, formando o logo da Furg.

5 - Cilindro:

Para construção do cilindro, utilizamos a função **gluCylinder**, construindo assim as bordas do sólido. Porém para a construção de seu preenchimento (discos frontal e traseiro) tivemos que implementar a construção do círculo manualmente, pois as funções encontradas funcionam num espaço 2D, corrompendo o aspecto 3D do objeto ao rotacionar.

Para isso, utilizamos a função GL_TRIANGLE_FAN, orientado pelos eixos x e y localizados no centro do círculo e sua posição em z, agora dando o aspecto 3d a figura.

```
void circulo_solido(GLfloat x, GLfloat y, GLfloat z, GLfloat radius) {
    int i;
    int triangleAmount = 50;

    GLfloat twicePi = 2.0f * 3.1415926;

    glBegin(GL_TRIANGLE_FAN);
    for(i = 0; i <= triangleAmount; i++) {
        glVertex3f(
            x + (radius * cos(i *  twicePi / triangleAmount)),
            y + (radius * sin(i * twicePi / triangleAmount)),
            z
        );
    }
    glEnd();
}
```

Figura 4: Função para a um círculo sólido.

Figura 5: Imagem do cilindro em 3D.

As outras figuras geométricas tratam-se de cubos criados com o método **gluSolidCube**. Cada um desses cubos possuem escalas, posições e cores diferentes, criando um só sólido formado por cubos sobrepostos, terminando a composição da imagem almejada.

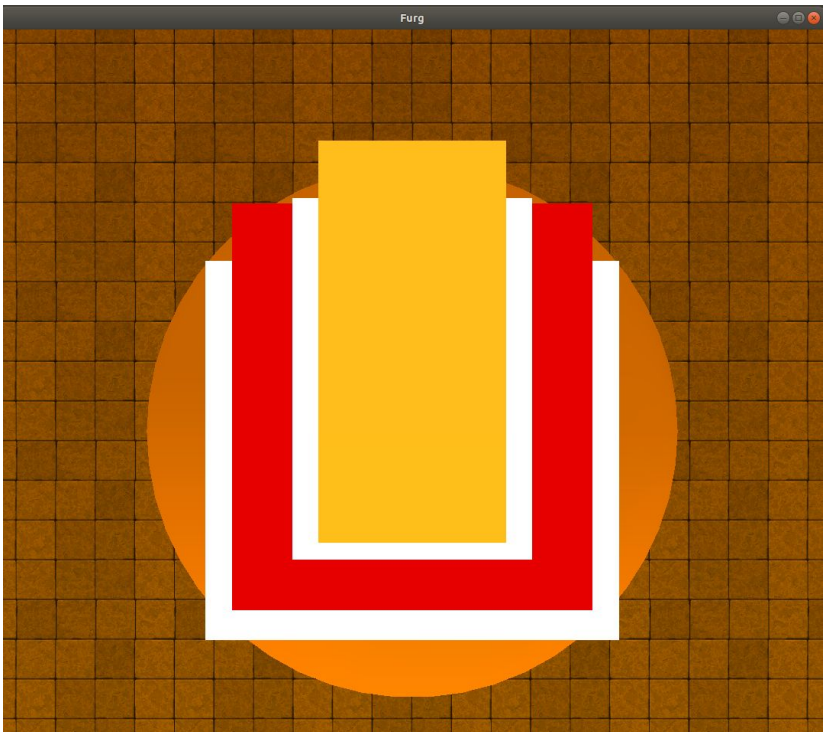

Figura 6: Imagem do produto final.

6 - Instruções de compilação:

-> g++ -o teste main2.cpp imageloader.o -lglut -lGLU -lGL -lm
-> ./teste