Sistemas de Cores

Henrique Damasceno Vianna [1]

[1] Disciplina de Computação Gráfica – Centro Universitário La Salle, Av Victor Barreto 2288, Centro, Canoas/RS - Cep 92010-000 - Fone: (51) 476.8500 - unilasalle@unilasalle.edu.br

hdvianna@cpovo.net

Resumo. *O artigo a seguir descreve as etapas e métodos utilizados para criar um ambiente tridimensional que simula um passeio em um carro utilizando funções da biblioteca gráfica Open GL. O artigo apontará, também algumas das dificuldades encontradas, e também técnicas utilizadas para solucionar os eventuais problemas.*

1.1 Projeto

Para atingir o objetivo proposto do trabalho foi utilizado a linguagem C junto com a *api glut* que é uma interface de programação da biblioteca *OpenGL*. No inicio do trabalho ficou definido que os objetos do mundo (as edificações), exceto o carro, teriam suas propriedades dimensionais definidas em um arquivo. Este arquivo ficou definido da seguinte maneira. O arquivo é formado por *n* linhas cada linha caracterizando uma edificação no mundo. Esta linha é dividida em 6 elementos, cada elemento corrrespondendo ao ponto mínimo e máximo nos eixos X, Y e Z do objeto no mundo. As linhas iniciadas pelo carcater *'#'* não são processadas. Desta maneira o arquivo ficou com este formato:

```
#ponto X inicial; ini_Y; ini_Z; ponto X final; fim_Y; fim_Z
-52.5;0;-55;-22.5;60;-25
-52.5;0;5.0;-22.5;30;35
-52.5;0;35.0;-22.5;45;75
#30;0;-30;60;30;0
30;0;-70;60;30;-40
30;0;10;60;30;40
30;0;50;60;30;80
-121.5;0;-105;-21.5;21;-75
30;0;-30;90;30;0
```

1.1 objetos.txt

Com esta formatação é possível representar uma gama de instâncias de um objeto cubo no universo.

Para processar esse arquivo foram contruídas duas bibliotecas: *arquivo.cpp* e *estruturas.cpp.* A biblioteca *arquivo.cpp* contém as funções para leitura linha a linha do arquivo.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifndef ARQUIVO

 #define ARQUIVO FILE

 ARQUIVO *abreArquivo(char *nomeArquivo, char *tipo);
 char *leLinha(char *linha, FILE *f, int *fim);
 int escreveLinha(ARQUIVO *f,char *linha);
 int fechaArquivo(ARQUIVO *f);

#endif
```

1.2 arquivo.h

A biblioteca *estrutura.cpp* contém uma estrutura para armazenar os dados dos objetos definidos no arquivo, este armazenamento tem grande importância para o tratamento de colisão, além de diminuir a quantidade de código do programa principal.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "arquivo.h"

#ifndef ESTRUTURAS

    #define ESTRUTURAS 1

    typedef struct sObj {
        double pIniX ;
        double pIniY;
        double pIniZ;
        double pFimX;
        double pFimY;
        double pFimZ;
    } tObj;

    tObj carregaObjeto(char *strLinha, int *carregou);
    tObj *carregaObjetos(tObj *objetos, int *qtd);

#endif
```

1.3 estruturas.h

2 Implementação

Para implementar o trabalho, foram utilizados alguns dos exemplos vistos em aula para a parte de iluminação, as demais partes do programa foram elaboradas usando conceitos matemáticos e recursos recolhidos em pesquisas e materiais referentes a disciplina.

O programa principal contêm diversas funções mas as principais são estas:

- `void Inicializa (void)`
- `void desenha (tObj objeto)`
- `void desenhaUniverso()`
- `void moveVeiculo ()`
- `int  detectaColisao(tObj carro)`
- `void moveCamera()`
- `void Teclado (int key, int x, int y)`

2.1 A função *inicializa()*

A função *inicializa* é encarregada de fazer a carga das propriedades do ambiente do universo, tais como intensidade de luz, cor da luz, tipos de iluminação (difusa, especular, ambiente). Nessa função também são definidos o compartamento do material dos objetos, como sua capacidade de refletir luz, brilho, etc.; o modelo de colorização, que nesse caso é o algoritmo de Gouraud.

```
void Inicializa (void)
{

      GLfloat luzAmbiente[4]={0.2,0.2,0.2,1.0};
      GLfloat luzDifusa[4]={0.7,0.7,0.7,1.0};
      GLfloat luzEspecular[4]={1.0, 1.0, 1.0, 1.0};
      GLfloat posicaoLuz[4]={0.0, 150.0, 0.0, 1.0};

      // Capacidade de brilho do material
      GLfloat especularidade[4]={1.0,1.0,1.0,1.0};
      GLint especMaterial = 60;
      // Especifica que a cor de fundo da janela será
      glClearColor(0.5f, 0.5f, 0.5f, 0.0f);
      // Habilita o modelo de colorização de Gouraud
      glShadeModel(GL_SMOOTH);
      // Define a refletância do material
      glMaterialfv(GL_FRONT,GL_SPECULAR, especularidade);
      // Define a concentração do brilho
      glMateriali(GL_FRONT, GL_SHININESS,especMaterial);
      // Ativa o uso da luz ambiente
      glLightModelfv(GL_LIGHT_MODEL_AMBIENT, luzAmbiente);
      // Define os parâmetros da luz de número 0
      glLightfv(GL_LIGHT0, GL_AMBIENT, luzAmbiente);
```

```
    glLightfv(GL_LIGHT0, GL_DIFFUSE, luzDifusa );
    glLightfv(GL_LIGHT0, GL_SPECULAR, luzEspecular );
    glLightfv(GL_LIGHT0, GL_POSITION, posicaoLuz );
    // Habilita a definição da cor do material a partir da
    cor corrente
    glEnable(GL_COLOR_MATERIAL);
    //Habilita o uso de iluminação
    glEnable(GL_LIGHTING);
    // Habilita a luz de número 0
    glEnable(GL_LIGHT0);
    // Habilita o depth-buffering
    glEnable(GL_DEPTH_TEST);

    angle=45;

}
```

1.4 Função Inicializa()

2.2 A função *desenha (tObj objeto)*

A função *desenha* cria um cubo no universo, este cubo é redimensionado utilizando as propriedades espaciais passadas pelo argumento, após isso ele é transposto para o seu lugar no plano, definido pelos valores encontrados no argumento. O argumento *objeto* é uma estrutura definida na biblioteca *estrutura.cpp*, e que já foi apresentada nesse documento.

```
#define TAMANHO  30 /*Tamanho default dos objetos*/

void desenha (tObj objeto) {

    glPushMatrix();
    glColor3f(0.0f, 0.0f, 1.0f);
    glTranslatef(valEscala(objeto.pIniX, objeto.pFimX),
                 valEscala(objeto.pIniY, objeto.pFimY),
                 valEscala(objeto.pIniZ, objeto.pFimZ));
    glScalef((objeto.pFimX - objeto.pIniX)/TAMANHO,
             (objeto.pFimY - objeto.pIniY)/TAMANHO,
             (objeto.pFimZ - objeto.pIniZ)/TAMANHO);
    glutSolidCube(TAMANHO);
    glPopMatrix();
}
```

1.5 Função desenha(tObj objeto)

2.3 A função *desenhaUniverso()*

A função *desenhaUniverso* faz a criação do nosso mundo virtual. É uma função de *callback* onde serão chamados os procedimentos para a criação da topografia do universo (ruas, solo), bem como os procedimentos que irão posicionar o veículo em seu local, e também, fará o desenho dos prédios carregados do arquivo, em suas respectivas posições no universo.

```
// Função callback chamada para fazer o desenho
void desenhaUniverso()
{
    int i;

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    desenhaVertices();
    desenhaTopografia();

    for (i=0; i < qtd; i++)
        desenha(objetos[i]);/*Desenha os objetos do mundo*/

    if (atualCamera != 2)
        moveVeiculo(); /*Desenha o veiculo no seu local*/

    glutSwapBuffers();// Executa os comandos OpenGL
 }
```

1.6 Função desenhaUniverso()

2.4 A função *moveVeiculo()*

A função *moveVeiculo* faz o posicionamento do veículo no universo virtual, de acordo com as coordenadas que estão nas variáveis *pX* e *pZ,* e de acordo com com as propriedades emcontradas na variável *carro*. É nesta função que é chamado a função para detecção de colisão do veículo com outros objetos. Caso seja detectada uma colisão o veículo voltará a sua posição anterior.

```
/*Desenha e move o veículo*/
void moveVeiculo () {

    tObj carro;

    /*monta o veículo pelos vértices*/
    carro.pIniX = 0  + pX;
    carro.pFimX = 10 + pX;
    carro.pIniY = 0  + pY;
    carro.pFimY = 5  + pY;
```

```
    carro.pIniZ = 0  + pZ;
    carro.pFimZ = 15 + pZ;

    if (detectaColisao(carro)) {
        printf("Colidiu!\n");
        pX = p_oldX;
        pZ = p_oldZ;
        carro.pIniX = 0  + pX;
        carro.pFimX = 10 + pX;
        carro.pIniY = 0  + pY;
        carro.pFimY = 5  + pY;
        carro.pIniZ = 0 + pZ;
        carro.pFimZ = 15 + pZ;
    }

    glPushMatrix();
    glColor3f(1.0f, 0.0f, 0.0f);
    glTranslatef(valEscala(carro.pIniX, carro.pFimX),
                    valEscala(carro.pIniY, carro.pFimY),
                    valEscala(carro.pIniZ, carro.pFimZ));
    glRotatef(rttAngl, 0, 1, 0);
    glScalef((carro.pFimX - carro.pIniX)/5,
                (carro.pFimY - carro.pIniY)/5,
                (carro.pFimZ - carro.pIniZ)/5);
    glutSolidCube(5);
    glPopMatrix();


}
```

1.7 Função moveVeiculo()

2.5 A função *detectaColisao(tObj carro)*

A função *detectaColisao* verifica a colisão entre o argumento *carro* e algum objeto que está dentro do universo virtual. De certo modo o algoritmo para esta função é bem primitivo, já que ele foi baseado em funções de recorte de imagem (*clipping*), sendo menos preciso que algoritmos como *Bounding Sphere Collision Detection* ou *Triangle-to-Triangle Collision Detection.* Mesmo assim este algoritmo se mostrou eficaz na realidade do problema proposto, já que o universo é formado por cubos.

A função trabalha da seguinte maneira, passado um argumento do tipo *tObj*, verifica-se se um dos vértices do argumento encontra-se dentro dos limites de algum objeto do universo. Caso um vértice do argumento esteja dentro do limite do objeto a função retornará o valor 1, acusando uma colisão.

```
/*Detecta  a colisao do veiculo com os demais objetos*/
int detectaColisao(tObj carro) {
    int colX = 0, colZ = 0;

    for (int i=0; i < qtd; i++) {
        /*Colidiu em X*/
        if ((carro.pIniX >= objetos[i].pIniX) &&
            (carro.pIniX <= objetos[i].pFimX))
            colX = 1;
        /*Colidiu em X*/
        if ((carro.pFimX >= objetos[i].pIniX) &&
            (carro.pFimX <= objetos[i].pFimX))
            colX = 1;
        /*Colidiu em Z*/
        if ((carro.pIniZ >= objetos[i].pIniZ) &&
            (carro.pIniZ <= objetos[i].pFimZ))
            colZ = 1;
        /*Colidiu em Z*/
        if ((carro.pFimZ >= objetos[i].pIniZ) &&
            (carro.pFimZ <= objetos[i].pFimZ))
            colZ = 1;

        if (colZ && colX)
            return 1;/*Haverá uma colisão se houverem
                       colisões simultâneas em X e Z*/
        colZ = colX = 0; /*Reseta as flags de colisão*/
    }
    return 0;
}
```

1.8 Função detectaColisao(tObj carro)

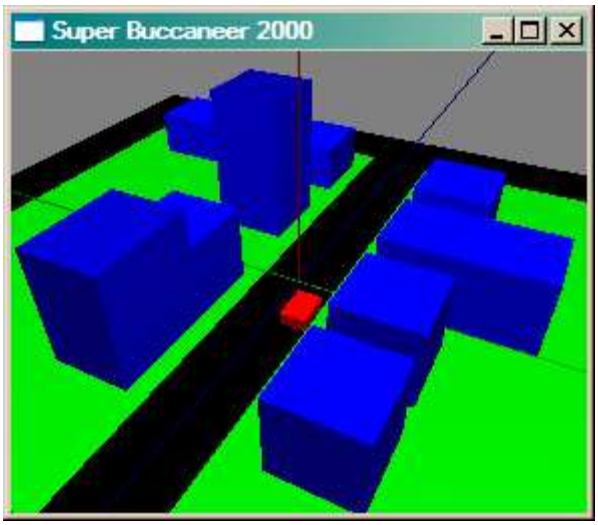
2.6 A função *moveCamera()*

A função *moveCamera* reposiciona a câmera em um ponto do universo. Existem três modos de câmera: perspectiva, visão do topo e 1º pessoa.

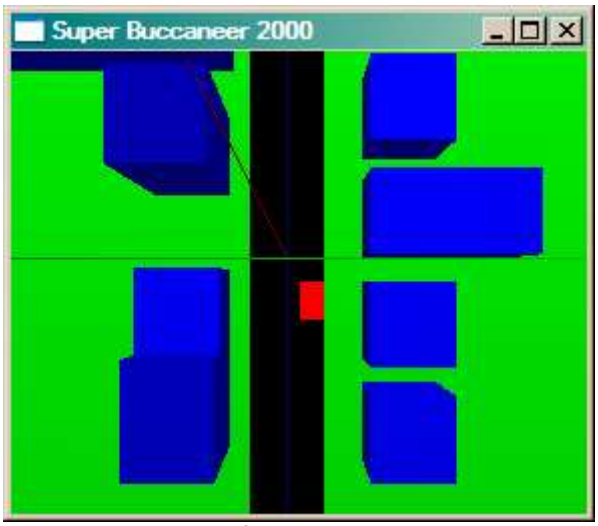
```
void moveCamera() {
    tObj carro; /*Objeto carro*/

       switch (atualCamera) {
        case 2: /*Primeira pessoa*/
            carro.pIniX = 0  + pX;
            carro.pFimX = 10 + pX;
            carro.pIniY = 0  + pY;
            carro.pFimY = 5  + pY;
            carro.pIniZ = 0  + pZ;
```

```
            carro.pFimZ = 15 + pZ;

            if (detectaColisao(carro)) {
                pX = p_oldX;
                pZ = p_oldZ;
                printf("Colidiu!\n");
            }
            /*Posiciona a câmera em primeira pessoa*/

          gluLookAt((carro.pIniX+carro.pFimX)/2,
                (carro.pIniY+carro.pFimY)/2
                ,(carro.pIniZ+carro.pFimZ)/2,
                cos(((rttAngl+90)/180)*PI) +
                (carro.pIniX+carro.pFimX)/2
                ,(carro.pIniY+carro.pFimY)/2,
                sin(((rttAngl-90)/180)*PI) +
                carro.pIniZ+carro.pFimZ)/2
                , 0,1,0);
            break;
        case 3: /*Perspectiva*/

           gluLookAt(100*((pX)/(fabs(pX))),
                    135,200*((pZ)/(fabs(pZ))),
                     0,0,0, 0,1,0);
            break;
        case 4: /*camera topo*/
            gluLookAt(pX,250,pZ, pX,pY,pZ, 0,0,-1);
            break;
        default:
            break;
    }
}
```

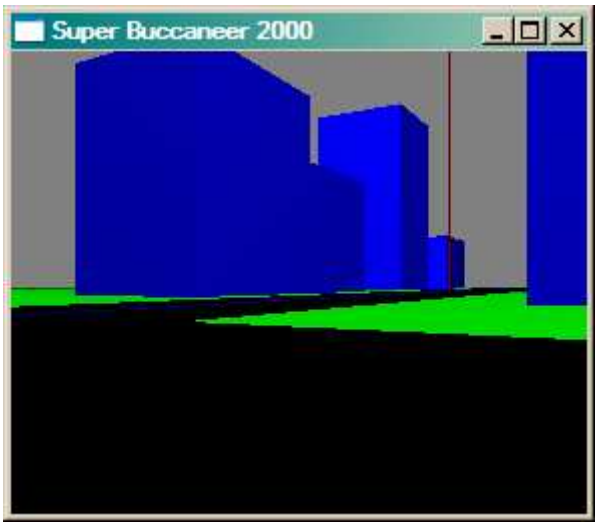
1.9 Função moveCamera()



2.1 Câmera em perspectiva



2.2 Câmera no topo



2.3 Câmera em primeira pessoa

2.7 A função *Teclado (int key, int x, int y)*

Teclado é a principal função do sistema. Esta é a função de *callback* que retorna a tecla pressionada. Toda interação do usuário com o programa é feita através o teclado, sendo que é por esse dispositivo que o usuário movimenta o 'veículo', usando as setas de direção. Muda a câmera, utilizando as teclas F2, F3 e F4. Ou aumenta ou diminui a velocidade do veículo, utilizando as teclas PAGE_UP e PAGE_DOWN respectivamente.

Nota-se, no algoritmo da função, que utilizamos o mesmo cálculo trigonométrico que o openGL utiliza para fazer a rotação dos objetos. Esse cálculo é feito para que a direção do movimento do objeto em relação ao eixo X seja corrigida quando o ângulo de rotação do objeto seja diferente de 0º, 45º ou 90º. Assim dando uma impressão correta de movimento do objeto sobre o plano. Este mesmo cálculo é utilizado para alterar o posicionamento da câmera e do alvo da câmera, na visualização em primeira pessoa (ver *2.6 A função moveCamera()*).

```
//Callback para gerenciar eventos do teclado para teclas especiais
void Teclado (int key, int x, int y)       {

        p_oldX = pX;
        p_oldZ = pZ;

        /*Troca a direção norte, sul*/
        if (fabs(rttAngl) <=  90)
            lat = 'n';
        else
            lat = 's';
        /*TROCA A DIREÇÃO LESTE - OESTE*/
        if (rttAngl > 0)
            lng='o';
        else
            lng='l';

            switch (key) {
                  /*Rotação para esquerda do veículo*/
            case GLUT_KEY_LEFT :

                if (fabs(rttAngl+5)>180) { /*Caso o ângulo fique maior
                                           que 180 graus, mudo o
                                           sentido*/
                    rttAngl = rttAngl*(-1);
                    if (rttAngl > 0)
                        lng='o';
                    else
                        lng='l';
                }
                op = 'r';
                rttAngl+=5;
                break;
            /*Rotação para direita do veículo*/
            case GLUT_KEY_RIGHT :
                if (fabs(rttAngl-5)>180) {  /*Caso o ângulo fique maior
                                            que 180 graus, mudo o
```

```
                                        sentido*/
                    rttAngl = rttAngl*(-1);
                    if (rttAngl > 0)
                        lng='o';
                    else
                        lng='l';
                }
                op = 'r';
                rttAngl-=5;
                break;
            /*Movimento para frente do veículo*/
            case GLUT_KEY_UP :
                pX = ((cos(((rttAngl+90)/180)*PI) * accl) + pX);
                pZ = (((sin(((rttAngl-90)/180)*PI)) * accl) + pZ);
                break;
            /*Movimento de Ré do veículo*/
            case GLUT_KEY_DOWN :
                pX = (pX - (cos(((rttAngl+90)/180)*PI)) * accl);
                pZ = (pZ - ((sin(((rttAngl-90)/180)*PI)) * accl));
                break;
            case GLUT_KEY_PAGE_UP :
                if (accl < 5)/*Aumenta a marcha =P*/
                    accl+=0.5;
                break;
            case GLUT_KEY_PAGE_DOWN :
                if (accl < 5)/*Diminui a marcha =(*/
                    accl-=0.5;
                break;
            case GLUT_KEY_F2 : /*Primeira pessoa*/
                atualCamera = 2;
                break;
            case GLUT_KEY_F3 : /*Visão perspectiva*/
                atualCamera = 3;
                break;
            case GLUT_KEY_F4 :/*Visão do topo*/
                atualCamera = 4;
                break;
        }
        glLoadIdentity();
        moveCamera();
        glutPostRedisplay();
}
```

1.10 Teclado (int key, int x, int y)

3.1 Conclusão

Ao desenvolver o trabalho pude observar que o openGL é uma API gráfica com numerosos recursos. A API consegue abstrair de maneira bem simples os conceitos de computação gráfica tridimensional, tornando o desenvolvimento de aplicativos com essa ferramenta mais fácil e de certa maneira intuitivos.

Apenas uma dificuldade foi assistida em relação a parte de profundidade e iluminação da cena. Esta dificuldade não foi presenciada em todos computadores em que

foi rodado o programa, sendo que ficou díficil concluir a causa do problema. Posteriormente foi detectado uma pequena dificuldade de certos dispositivos de vídeo tratarem valores baixos do *zNear* da câmera, o mesmo foi corrigido aumentanto o valor do *zNear* sem que diferenças fossem no objetivo final do programa. No restante a implementação dos conceitos vistos no decorrer do semestre com a API openGL foi bem tranquila, tanto na modelagem de objetos quanto no uso de câmera.

Referências

[Pinho 2000] PINHO, Márcio S. **Biblioteca Gráfica OpenGL**. Disponível em http://www.inf.pucrs.br/~pinho/CG/Aulas/OpenGL/OpenGL.html (Junho, 2004).

[Manssour 2001] MANSSOUR, Isabel Harb. **Introdução à OpenGL**. Disponível em http://www.inf.pucrs.br/~manssour/OpenGL/index.html (Junho, 2004).

[Longhi 2004] LONGHI, Magalí Teresinha. **Material sobre OpenGL**. Disponível em http://www.inf.unilasalle.edu.br/~magali/MaterialCG.htm (Junho, 2004).

DELOURA, Mark – Game Programming Gems. Massachussets: Charles River Media, 2000