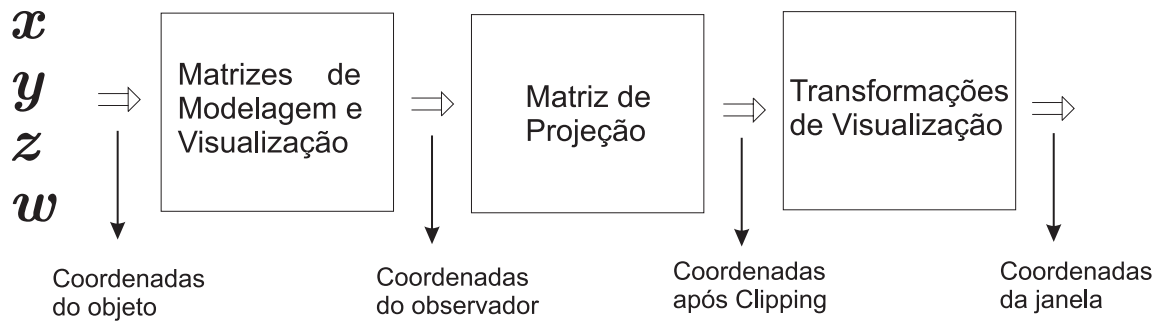
# CAPÍTULO III – VISUALIZAÇÃO E APLICAÇÕES GRÁFICAS 3D

## 1- TRANSFORMAÇÕES DE VISUALIZAÇÃO

Nosso objetivo nesta seção é descrever a geração de uma imagem bi-dimensional partindo de um objeto tri-dimensional. De forma geral podemos dividir as operações necessárias em três grupos:

- Transformações (representadas por multiplicação de matrizes) incluindo operações de projeção, visualização e modelagem. Estas operações incluem rotações, escalas, translações, reflexões, projeções ortográficas e perspectivas.
- Operações de Clipping são responsáveis pela eliminação de objetos que estão fora da janela de visualização.
- Transformações que estabeleçam a correspondência entre as coordenadas e a dimensão da tela (Viewport transformation).

De forma esquemática podemos estabelecer:




Para especificar uma transformação o OpenGL constrói uma matriz 4x4 que representa a transformação desejada. Esta matriz é então multiplicada pelas coordenadas de cada vértice na cena. As transformações de Modelagem e Visualização são combinadas em uma única matriz denominada `MODELVIEW matrix.` A transformação de projeção é armazenada na `PROJECTION matrix`.

### 1.1 - Comandos de Auxílio

Antes de iniciar a descrição do mecanismo das transformações acima, apresentaremos um conjunto de comandos de caráter geral. Os comandos a seguir serão úteis durante todas as etapas do processo de modelagem, visualização e Projeção.

```
GlMatrixMode(Glenum tipo);
```

Este comando especifica a matriz a ser alterada. Existem três argumentos conforme o *tipo* da matriz:
1) GL_MODELVIEW
2) GL_PROJECTION
3) GL_TEXTURE
As transformações subsequentes afetam a matriz especificada. Observe que somente uma matriz pode ser alterada por vez.

```
GlLoadIdentity(void);
```

Este comando carrega a matriz identidade na matriz corrente especificada anteriormente pelo `GlMatrixMode`. O objetivo é limpar qualquer alteração anterior realizada sobre a matriz.

```
GlLoadMatrix(const TYPE *M);
```

Quando você deseja especificar uma matriz M particular para ser a matriz corrente, utilize o `glLoadMatrix`(M).

```
GlMultMatrix(const TYPE *M);
```

Este comando multiplica a matriz definida por

$$M = \begin{bmatrix} m1 & m5 & m9 & m13 \\ m2 & m6 & m10 & m14 \\ m3 & m7 & m11 & m15 \\ m4 & m8 & m12 & m16 \end{bmatrix}$$

pela matriz corrente.

## 1.2- Exemplo: cubo unitário

Antes de detalhar as transformações principais, vamos apresentar um programa exemplo que visualiza um objeto tri-dimensional: o cubo unitário. As coordenadas do cubo são:

V0 = {0,0,0}
V1 = {1,0,0}
V2 = {1,1,0}
V3 = {0,1,0}
V4 = {0,0,1}
V5 = {1,0,1}
V6 = {1,1,1}
V7 = {0,1,1}

```
#include <gl\glut.h>

void draw_cubo()
{
      glBegin(GL_LINE_LOOP);
       glVertex3f(0.0,0.0,0.0);
       glVertex3f(1.0,0.0,0.0);
       glVertex3f(1.0,1.0,0.0);
       glVertex3f(0.0,1.0,0.0);
      glEnd();
      glBegin(GL_LINE_LOOP);
       glVertex3f(0.0,0.0,1.0);
       glVertex3f(1.0,0.0,1.0);
       glVertex3f(1.0,1.0,1.0);
       glVertex3f(0.0,1.0,1.0);
      glEnd();
      glBegin(GL_LINES);
       glVertex3f(0.0,0.0,0.0);
       glVertex3f(0.0,0.0,1.0);
       glVertex3f(1.0,0.0,0.0);
       glVertex3f(1.0,0.0,1.0);
       glVertex3f(1.0,1.0,0.0);
       glVertex3f(1.0,1.0,1.0);
       glVertex3f(0.0,1.0,0.0);
       glVertex3f(0.0,1.0,1.0);
      glEnd();
}

void display()
```

```
{
      glClearColor(0.0,0.0,0.0,0.0);
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT  );
      glColor3f(1.0,1.0,0.0);
      draw_cubo();
      glFlush();
}

void inicia_config()
{
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-2.0,2.0,-2.0,2.0,-10.0,10.0);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glRotatef(45,0.0,1.0,0.0);
      glRotatef(45.0,0.0,0.0,1.0);
}

void main(int argc, char **argv)
{
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);
      glutInitWindowSize(400,400);
      glutInitWindowPosition(50,50);
      glutCreateWindow("Cubo 3D");
      glutDisplayFunc(display);
      inicia_config();
      glutMainLoop();
}
```

## 1.3 - Transformações de Modelagem e Visualização

Existem três tipos de comandos para transformações de modelagem: `glTranslate()`, `glRotate()` e `glScale()`. Vamos descrever abaixo cada uma dessas funções.

### 1.3.1- Translação

```
void glTranslatef(float x, float y, float z);
```

Este comando multiplica a matriz corrente por uma matriz que translada o objeto conforme o vetor $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$.

### 1.3.2- Rotação

```
void glRotatef(float theta, float x, float y, float z);
```

Multiplica a matriz corrente por uma matriz que rotaciona o objeto no sentido anti-horário de `theta` graus, na direção do eixo dado pelo vetor $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$.

### 1.3.3- Escala

```
void glScalef(float x, float y, float z);
```

Este comando realiza transformações de escala e reflexão. Cada ponto x, y e z do objeto é multiplicado pelo correspondente argumento `x,y e z`.

### 1.3.4-Exemplo

Como exemplo vamos apresentar um programa que executa as três transformações citadas acima. Partindo de um triângulo, desenhamos este triângulo quatro vezes sendo que:

- O triângulo vermelho é desenhado sem nenhuma transformação.
- O triângulo verde sofreu uma translação.
- O triângulo azul sofreu uma rotação.
- O triângulo amarelo sofreu uma transformação de escala.

```
#include <gl\glut.h>
#include <stdio.h>

void draw_triangulo()
{

      glBegin(GL_LINE_LOOP);
        glVertex3f(0.0,0.0,0.0);
        glVertex3f(0.5,0.0,0.0);
        glVertex3f(0.25,0.43,0.0);
      glEnd();
}


void display()
{
      glClearColor(0.0,0.0,0.0,0.0);
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT  );
      glColor3f(1.0,0.0,0.0);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      draw_triangulo();
      glColor3f(0.0,1.0,0.0);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glTranslatef( 0.5, 0.5,0.0);
      draw_triangulo();
      glColor3f(0.0,0.0,1.0);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glRotatef(45,0.0,0.0,1.0);
      draw_triangulo();
      glColor3f(1.0,1.0,0.0);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glScalef(0.5, 0.5,0.5);
      draw_triangulo();
      glFlush();
}


void main(int argc, char **argv)
{
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);
      glutInitWindowSize(400,400);
      glutInitWindowPosition(50,50);
      glutCreateWindow("Triangulo");
      glutDisplayFunc(display);
      glutMainLoop();
}
```

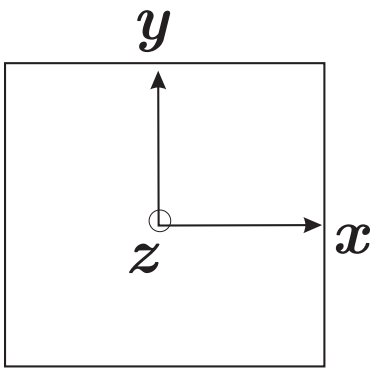
## 1.4- Projeção Ortográfica

Em uma projeção ortográfica, nós estabelecemos um volume de visualização (que corresponde a um paralelepípedo retângulo). O objeto só será visualizado se ele estiver contido neste volume. O comando utilizado é:

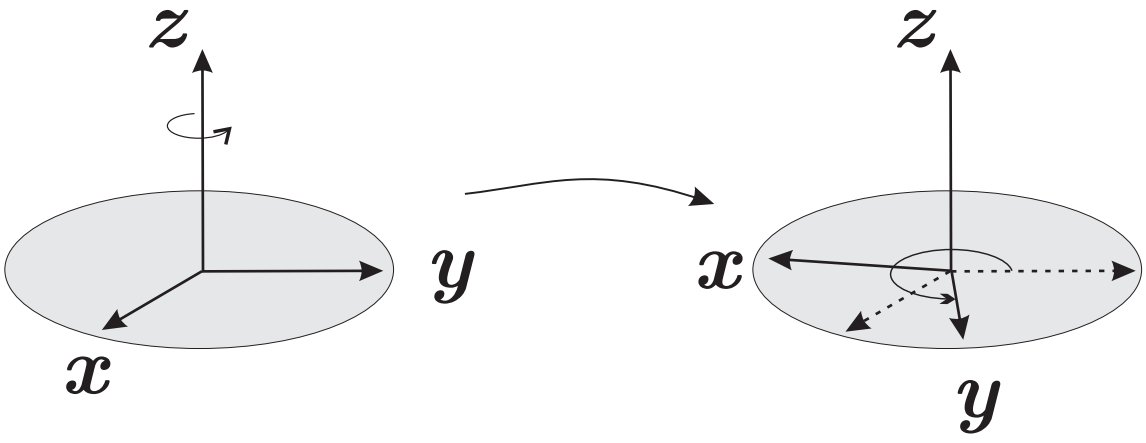
```
glOrtho(double left, double right, double bottom, double top,
                                   double near, double far);
```

## 1.5- Ângulos de Euler

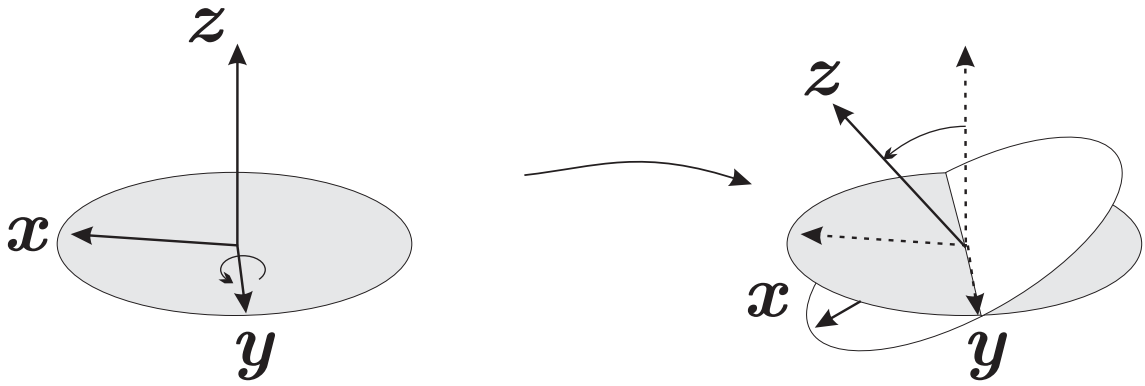
Um sistema de referência que irá nos auxiliar na montagem do ambiente 3D são ao ângulos de Euler. Os Ângulos de Euler são definidos como três sucessivos ângulos de rotação através do qual definiremos um novo sistema de coordenadas partindo das coordenadas cartesianas.
No Opengl o sistema de coordenadas está posicionado conforme o desenho abaixo, onde o eixo x está na posição horizontal, o eixo y na posição vertical e o eixo z está apontando para fora da tela do computador.



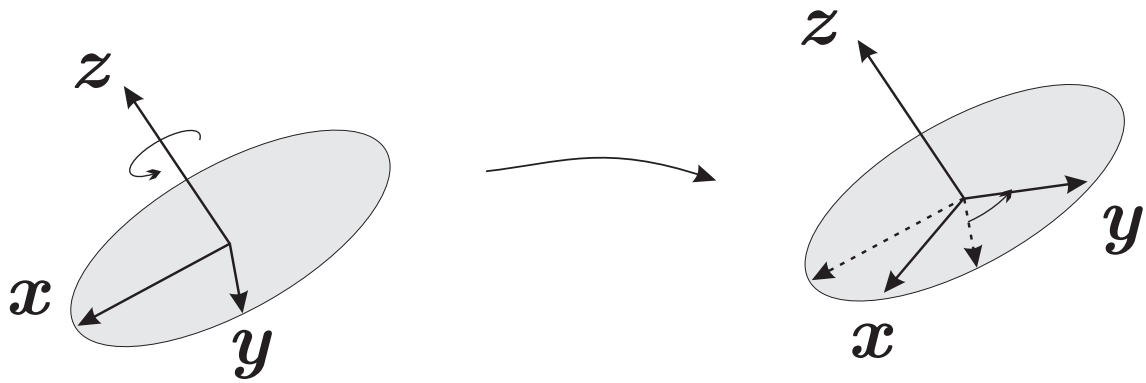

A seqüência inicia rotacionando o sistema de coordenadas xyz por um ângulo $\theta$ no sentido anti-horário em torno do eixo z.



Em seguida o sistema de coordenadas resultante é rotacionado em torno do eixo y de $\phi$ graus.




Por fim o sistema de coordenadas sofre uma nova rotação de $\psi$ em torno do eixo z.




Estes três ângulos definem os ângulos de Euler.

## 1.6 - Criando um Ambiente de Visualização 3D

Vamos montar um ambiente para visualização tri-dimensional, através do qual poderemos visualizar nossos objetos 3D.

### 1.6.1- Módulo Básico de Visualização

Como primeiro passo vamos desenhar os três eixos de referencia, mantendo a seguinte escala de cores:
- Eixo x: vermelho
- Eixo y: verde
- Eixo z: azul

Uma posição inicial para a visualização pode ser obtida utilizando-se os ângulos theta = 135, phi = 45, gamma=90.

```
#include <gl\glut.h>
#include <stdio.h>

float gamma=90.0,phi=45.0,theta=135.0;

void draw_eixos()
{

      glColor3f(1.0,0.0,0.0);
      glBegin(GL_LINES);
        glVertex3f(10.0,0.0,0.0);
        7glVertex3f(0.0,0.0,0.0);
```

```
      glEnd();
      glColor3f(0.0,1.0,0.0);
      glBegin(GL_LINES);
        glVertex3f(0.0,10.0,0.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
      glColor3f(0.0,0.0,1.0);
      glBegin(GL_LINES);
        glVertex3f(0.0,0.0,10.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
}

void display()
{
      glClearColor(0.0,0.0,0.0,0.0);
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT  );
      glColor3f(1.0,1.0,0.0);
      draw_eixos();
      glFlush();
}

void inicia_config()
{
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glRotatef(gamma,0.0,0.0,1.0);
      glRotatef(phi,0.0,1.0,0.0);
      glRotatef(theta,0.0,0.0,1.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-2.0,2.0,-2.0,2.0,-10.0,10.0);

}

void main(int argc, char **argv)
{
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);
      glutInitWindowSize(400,400);
      glutInitWindowPosition(50,50);
      glutCreateWindow("Cubo 3D");
      glutDisplayFunc(display);
      inicia_config();
      glutMainLoop();
}
```

### 1.6.2- Alterando os ângulos de Euler

Vamos agora acrescentar a possibilidade de alterar os ângulos com auxílio do mouse. Com o botão Esquerdo do mouse pressionado, quando o mouse anda na direção horizontal, alteramos theta, quando o muse anda na direção vertical alteramos phi.

```
#include <gl\glut.h>
#include <stdio.h>

int   xm,xb,ym,yb;
float gamma=90.0,phi=45.0,theta=135.0;

void draw_eixos()
{

      glColor3f(1.0,0.0,0.0);
      glBegin(GL_LINES);
        glVertex3f(10.0,0.0,0.0);
```

```
        glVertex3f(0.0,0.0,0.0);
      glEnd();
      glColor3f(0.0,1.0,0.0);
      glBegin(GL_LINES);
        glVertex3f(0.0,10.0,0.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
      glColor3f(0.0,0.0,1.0);
      glBegin(GL_LINES);
        glVertex3f(0.0,0.0,10.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
}

void display()
{
      glClearColor(0.0,0.0,0.0,0.0);
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT  );
      glColor3f(1.0,1.0,0.0);
      draw_eixos();
      glFlush();
}

void inicia_config()
{
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glRotatef(gamma,0.0,0.0,1.0);
      glRotatef(phi,0.0,1.0,0.0);
      glRotatef(theta,0.0,0.0,1.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-2.0,2.0,-2.0,2.0,-10.0,10.0);

}

void botao_mouse(int b,int state,int x, int y)
{

      switch(b) {
         case GLUT_LEFT_BUTTON:
               switch(state) {
                  case GLUT_DOWN:
                                       xb = x;
                                       yb = y;
                                       break;
                  case GLUT_UP:
                                       theta = theta + xm - xb;
                                       phi   = phi   - ym + yb ;
                                        break;
               }
               break;
      }
}

void mov_mouse(int x, int y)
{
      xm = x;
      ym = y;
```

```
      theta = theta + xm - xb;
      phi   = phi   - ym + yb ;
      inicia_config();
      theta = theta - xm + xb;
      phi   = phi   + ym - yb;
      display();
}

void main(int argc, char **argv)
{
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);
      glutInitWindowSize(400,400);
      glutInitWindowPosition(50,50);
      glutCreateWindow("Cubo 3D");
      glutDisplayFunc(display);
      inicia_config();
      glutMouseFunc(botao_mouse);
      glutMotionFunc(mov_mouse);
      glutMainLoop();
}
```

## 1.7 – Visualização de gráfico de funções $z = f(x, y)$ □

A seguir apresentamos um programa exemplo para visualização de gráficos de funções $f : \Re^2 \to \Re$ . O programa está dividido em dois módulos: `funcao3D01.cpp` e `funcao3D01a.cpp`.

### 1.7.1 – Módulo `funcao3D010.cpp`

```
#include <gl\glut.h>
#include <stdio.h>
#include "funcao3D010.h"

extern float ptx,pty;
extern float xmin,xmax,ymin,ymax;

int   xb,yb,xm,ym;
float gamma=90.0,phi=45.0,theta=135.0;
float scale = 1.0;

void display()
{
      glClearColor(0.0,0.0,0.0,0.0);
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT  );
      glColor3f(1.0,1.0,0.0);
      draw_eixos();
      draw_funcao();
      glFlush();
}

void inicia_config()
{
      glEnable(GL_DEPTH_TEST);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
     glScalef(scale,scale,scale);
      glRotatef(gamma,0.0,0.0,1.0);
```

```
      glRotatef(phi,0.0,1.0,0.0);
      glRotatef(theta,0.0,0.0,1.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-2.0,2.0,-2.0,2.0,-10.0,10.0);
}

void botao_mouse(int b,int state,int x, int y)
{

      switch(b) {
         case GLUT_LEFT_BUTTON:
               switch(state) {
                  case GLUT_DOWN:
                                    xb = x;
                                    yb = y;
                                     break;
                  case GLUT_UP:
                              theta = theta + xm - xb;
                              phi   = phi   - ym + yb ;
                                     break;
               }
               break;
      }
}

void mov_mouse(int x, int y)
{
      xm = x;
      ym = y;
     theta = theta + xm - xb;
     phi   = phi   - ym + yb ;
      inicia_config();
      theta = theta - xm + xb;
      phi   = phi   + ym - yb;
      display();
}

void le_tecla(unsigned char key, int x, int y)
{
      switch(key)
      {
           case '+':
                        scale+=0.2;
                           inicia_config();
                           display();
                        break;
           case '-':
                        scale-=0.2;
                           inicia_config();
                           display();
                        break;
      }
}

void main(int argc, char **argv)
{
      printf("\nxmin = ");
      scanf("%f",&xmin);
      printf("\nxmax = ");
      scanf("%f",&xmax);
```

```
      printf("\nymin = ");
      scanf("%f",&ymin);
      printf("\nymax = ");
      scanf("%f",&ymax);
      printf("\nPontos em x = ");
      scanf("%f",&ptx);
      printf("\nPontos em y = ");
      scanf("%f",&pty);
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);
      glutInitWindowSize(400,400);
      glutInitWindowPosition(50,50);
      glutCreateWindow("Cubo 3D");
      glutDisplayFunc(display);
    inicia_config();
      glutMouseFunc(botao_mouse);
      glutMotionFunc(mov_mouse);
      glutKeyboardFunc(le_tecla);
      glutMainLoop();
}
```

### 1.7.2 - Módulo `funcao3D011.cpp`

```
#include <gl/glut.h>
#include <math.h>

float xmin,xmax,ymin,ymax;
float ptx,pty;

float funcao(float x,float y)
{
//    return(sqrt(x*x+y*y));
//    return(y*y-x*x);
//    return(x*x+y*y);
//    return(-x-y+1);
      return(sin(sqrt(x*x+y*y)));
}

void draw_eixos()
{
      glColor3f(1.0,0.0,0.0);
      glBegin(GL_LINES);
        glVertex3f(10.0,0.0,0.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
      glColor3f(0.0,1.0,0.0);
      glBegin(GL_LINES);
        glVertex3f(0.0,10.0,0.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
      glColor3f(0.0,0.0,1.0);
      glBegin(GL_LINES);
        glVertex3f(0.0,0.0,10.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
}

void draw_funcao()
{
   float x,y;
   float stepx,stepy;
   float z0,z1,z2,z3;
```

```
   stepx = (xmax-xmin)/ptx;
   stepy = (ymax-ymin)/pty;

   glColor3f(1.0,1.0,1.0);
   for(x=xmin;x<=xmax;x+=stepx)
   {
         for(y=ymin;y<=ymax;y+=stepy)
         {
               z0 = funcao(x,y);
               z1 = funcao(x+stepx,y);
               z2 = funcao(x+stepx,y+stepy);
               z3 = funcao(x,y+stepy);
               glBegin(GL_LINE_LOOP);
                  glVertex3f(x,y,z0);
                    glVertex3f(x+stepx,y,z1);
                    glVertex3f(x+stepx,y+stepy,z2);
                    glVertex3f(x,y+stepy,z3);
               glEnd();
         }
   }
}
```

**1.7.3 - Módulo funcao3D010.h**

```
void draw_eixos();
void draw_funcao();
void plota_curvas_nivel(float ci,float cf);
void triangulo(float x1,float y1,float x2,float y2,float x3,float y3);
void plota_curva_nivel();
```

# 2- ILUMINAÇÃO

Para definir o seu modelo de iluminação são necessárias três etapas básicas:
1) Definir as fontes de luz (posição, cor, direção, etc.);
2) Definir a iluminação.
3) Definir o tipo de material do objeto.

O modelo de iluminação do OPENGL considera que a iluminação pode ser dividida em quatro componentes independentes: emitida, ambiente, difusa e especular.
- Luz Emitida: é a componente que se origina de um objeto e é inalterada pelas fontes de luz.
- Luz Ambiente: é a luz proveniente de uma fonte dispersa tal que sua direção não pode ser determinada.
- Luz Difusa: é a luz proveniente de uma única direção.
- Luz Especular: é a luz proveniente de uma direção particular e tende a refletir em uma direção preferencial.

## 2.1– Criando Fontes de Luz

.
As fontes de luz têm certas propriedades que devem ser definidas (Cor, direção, posição, etc.). O comando para especificar essas propriedades é:

```
void glLightfv(GLenum luz, GLenum iluminação, GLenum param);
```

O parâmetro luz indica apenas qual fonte de luz estamos trabalhando. Existem no máximo oito fontes que são: `GL_LIGHT0, GL_LIGHT1, ... , GL_LIGHT7.` Por default a luz `GL_LIGHT0` inicia com a cor branca e as sete luzes restantes ficam apagadas (luz preta).

Os parâmetros da iluminação são:

| **Parâmetro** | **Valor default** | **Significado** |
|---|---|---|
| GL_AMBIENT | (0.0, 0.0, 0.0, 1.0) | Intensidade da luz ambiente |
| GL_DIFFUSE | (1.0, 1.0, 1.0, 1.0) | Intensidade da luz difusa |
| GL_SPECULAR | (1.0, 1.0, 1.0, 1.0) | Intensidade da luz especular |
| GL_POSITION | (0.0, 0.0, 1.0, 0.0) | posição da luz |
| GL_SPOT_DIRECTION | (0.0, 0.0, -1.0) | direção da luz |
| GL_SPOT_EXPONENT | 0.0 | Parâmetro que controla a distribuição da luz. |
| GL_SPOT_CUTOFF | 180.0 | Ângulo de abertura da luz |
| GL_CONSTANT_ATTENUATION | 1.0 | |
| GL_LINEAR_ATTENUATION | 0.0 | |
| GL_QUADRATIC_ATTENUATION | 0.0 | |

### 2.1.1– Cor

A característica da luz é definida pelo parâmetro iluminação. O modelo de iluminação do OPENGL considera que a iluminação pode ser dividida em quatro componentes independentes: emitida, ambiente, difusa e especular.
- Luz Emitida: é a componente que se origina de um objeto e é inalterada pelas fontes de luz.
- Luz Ambiente: é a luz proveniente de uma fonte dispersa tal que sua direção não pode ser determinada.
- Luz Difusa: é a luz proveniente de uma única direção. Define a luz que naturalmente definiríamos como a cor da luz.
- Luz Especular: é a luz proveniente de uma direção particular e tende a refletir em uma direção preferencial. Se você quer criar efeitos realísticos, mantenha a luz especular com os mesmos parâmetros da luz difusa.

Por exemplo, para alterar a luz ambiente utiliza-se o seguinte código:

```
GLfloat luz_ambiente[4] = { 0.0, 0.0, 0.0, 1.0 };

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambiente);
```

### 2.1.2– Posição

A posição da luz pode ser de dois tipos básicos:

- *Direcional*: é quando a fonte de luz é considerada no infinito. Neste caso os raios de luz incidem paralelos ao objeto. Para obter, por exemplo, uma fonte de luz branca você deve utilizar o seguinte código:

```
GLfloat luz_posicao[4] = { 1.0, 1.0, 1.0, 0.0 };

glLightfv(GL_LIGHT0, GL_POSITION, luz_posicao);
```

- *Posicional* : Se o último valor do vetor `luz_posicao[]` for diferente de zero, a luz é posicional e sua localização é definida pelo vetor `luz_posicao[4]={x , y, z, 1.0}.`

## 2.2– Selecionando o Modelo de Iluminação

### 2.2.1- Luz Ambiente Global

Cada fonte de luz pode contribuir com uma parcela da luz ambiente. Além disso é possível adicionar uma outra parcela de luz ambiente que não dependa das fontes de iluminação. Para isso utiliza-se o comando:

```
GLfloat luz_ambiente_modelo[4] = { 0.2, 0.2, 0.2, 1.0 };

glLightModelfv(GL_LIGHT_MODEL_AMBIENT, luz_ambiente_modelo);
```

Observe que neste caso, mesmo que todas as fontes de luz estejam desligadas ainda assim será possível ver os objetos na cena.

### 2.2.2 – Posição do observador local ou no infinito

A localização do observador pode ou não influenciar na iluminação. O default é o observador no infinito. Para mudar a configuração, considerando-se a iluminação conforme o observador utilize:

```
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
```

### 2.2.3 – Iluminação nos dois lados das faces

O cálculo da iluminação é feito para todos os polígonos. É possível considerar diferentes iluminações nos dois lados de um polígono. Para isso utilize:

```
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
```

### 2.2.4 – Habilitando a iluminação

No OpenGL você precisa explicitamente habilitar a iluminação. Para isso utilize o comando:

```
glEnable(GL_LIGHTING);
```

Para desabilitar basta utilizar o comando:

```
glDisable(GL_LIGHTING);
```

## 2.3– Selecionando as Propriedades do Material

Para definir as propriedades do material do objeto em cena utilizamos os seguinte comando:

```
void glMaterialfv(GLenum face, GLenum iluminacao, TYPE param);
```

O parâmetro face pode ser: GL_FRONT, GL_BACK ou GL_FRONT_AND_BACK.

Os parâmetros da iluminação são:

| **Parâmetro** | **Valor default** | **Significado** |
|---|---|---|
| GL_AMBIENT | (0.2, 0.2, 0.2, 1.0) | Cor da luz ambiente do material |
| GL_DIFFUSE | (0.8, 0.8, 0.8, 1.0) | Cor da luz difusa do material |
| GL_SPECULAR | (0.0, 0.0, 0.0, 1.0) | Especular cor do material |
| GL_SHININESS | 0.0 | Índice especular |
| GL_EMISSION | (0.0, 0.0, 0.0, 1.0) | Cor de emissão do material |

## 2.4– Exemplo

O exemplo abaixo altera o programa funcao3D, e permite que o usuário altere algumas propriedades da iluminação. O programa esta dividido em três módulos: `Ilumina_funcao.cpp`, `Ilumina_main.cpp`, `ilumina_set.cpp`.

### 2.4.1- Módulo `Ilumina_main.cpp`

```
#include <gl\glut.h>
#include <stdio.h>
#include "ilumina_funcao.h"
#include "ilumina_set.h"

extern float ptx,pty;
extern float xmin,xmax,ymin,ymax;
extern float light[8][4];
extern GLfloat    spot_direction[3] ;
extern GLfloat    spot_cutoff       ;
extern GLfloat    spot_exponent     ;
extern GLfloat    c_attenuation     ;
extern GLfloat    l_attenuation     ;
extern GLfloat    q_attenuation     ;
extern GLfloat    material_shininess[1];

int   type_light = 0;
int   xb,yb,xm,ym;
float gamma=90.0,phi=45.0,theta=135.0;
float scale = 1.0;

void display()
{

  glClearColor(0.0,0.0,0.0,0.0);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT  );
  glEnable(GL_NORMALIZE);
  glColor3f(1.0,1.0,0.0);
  draw_eixos();
```

```
  draw_funcao();
  glFlush();
  glutSwapBuffers();
}

void inicia_config()
{

  glEnable(GL_DEPTH_TEST);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glLightfv(GL_LIGHT0, GL_POSITION, light[2]);
  glScalef(scale,scale,scale);
  glRotatef(gamma,0.0,0.0,1.0);
  glRotatef(phi,0.0,1.0,0.0);
  glRotatef(theta,0.0,0.0,1.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(-2.0,2.0,-2.0,2.0,-10.0,10.0);
}

void botao_mouse(int b,int state,int x, int y)
{
  switch(b) {
     case GLUT_LEFT_BUTTON:
       switch(state) {
                        case GLUT_DOWN:
                                       xb = x;
                                       yb = y;
                                       break;
                        case GLUT_UP:
                                       theta = theta + xm - xb;
                                       phi   = phi   - ym + yb ;
                                  break;
       }
       break;
  }
}

void mov_mouse(int x, int y)
{
      xm = x;
      ym = y;
      theta = theta + xm - xb;
      phi   = phi   - ym + yb ;
      inicia_config();
      theta = theta - xm + xb;
      phi   = phi   + ym - yb;
      display();
}


void le_tecla(unsigned char key, int x, int y)
{
      switch(key)
      {
           case '+':
                       scale+=0.2;
                           inicia_config();
                           display();
                       break;
           case '-':
                       scale-=0.2;
                           inicia_config();
                           display();
                       break;
             case ' ':
```

```
                        type_light = (type_light + 1 ) % 8 ;
                          switch(type_light)
                          {
                              case 0:
                                        printf("Luz Ambiente \n");
                                  printf("%f %f %f \n
",light[0][0],light[0][1],light[0][2]);
                                  break;
                              case 1:
                                        printf("Luz Difusa \n");
                                  printf("%f %f %f \n
",light[1][0],light[1][1],light[1][2]);
                                  break;
                              case 2:
                                        printf("Posicao Luz \n");
                                  printf("%f %f %f \n
",light[2][0],light[2][1],light[2][2]);
                                  break;
                              case 3:
                                        printf("Luz Especular \n");
                                  printf("%f %f %f \n
",light[3][0],light[3][1],light[3][2]);
                                  break;
                              case 4:
                                        printf("Material Ambiente \n");
                                  printf("%f %f %f \n
",light[4][0],light[4][1],light[4][2]);
                                  break;
                              case 5:
                                        printf("Material Difusa \n");
                                  printf("%f %f %f \n
",light[5][0],light[5][1],light[5][2]);
                                  break;
                              case 6:
                                        printf("Material Especular \n");
                                  printf("%f %f %f \n
",light[6][0],light[6][1],light[6][2]);
                                  break;
                              case 7:
                                        printf("Ambiente \n");
                                  printf("%f %f %f \n
",light[7][0],light[7][1],light[7][2]);
                                  break;
                          }
                          break;
            case 'R':
                        light[type_light][0] += 0.1;
                          light[type_light][0] = (light[type_light][0] >
1.0) ? 1.0 : light[type_light][0];
                          light[type_light][0] = (light[type_light][0] <
0.0) ? 0.0 : light[type_light][0];
                          printf("%f \n",light[type_light][0]);
                          display();
                          break;
            case 'r':
                        light[type_light][0] -= 0.1;
                          light[type_light][0] = (light[type_light][0] >
1.0) ? 1.0 : light[type_light][0];
                          light[type_light][0] = (light[type_light][0] <
0.0) ? 0.0 : light[type_light][0];
                          printf("%f \n",light[type_light][0]);
                          display();
                          break;
            case 'G':
                        light[type_light][1] += 0.1;
                          light[type_light][1] = (light[type_light][1] >
1.0) ? 1.0 : light[type_light][1];
```

```
                            light[type_light][1] = (light[type_light][1] <
0.0) ? 0.0 : light[type_light][1];
                            printf("%f \n",light[type_light][1]);
                            display();
                            break;
            case 'g':
                          light[type_light][1] -= 0.1;
                            light[type_light][1] = (light[type_light][1] >
1.0) ? 1.0 : light[type_light][1];
                            light[type_light][1] = (light[type_light][1] <
0.0) ? 0.0 : light[type_light][1];
                            printf("%f \n",light[type_light][1]);
                            display();
                            break;
            case 'B':
                          light[type_light][2] += 0.1;
                            light[type_light][2] = (light[type_light][2] >
1.0) ? 1.0 : light[type_light][2];
                            light[type_light][2] = (light[type_light][2] <
0.0) ? 0.0 : light[type_light][2];
                            printf("%f \n",light[type_light][2]);
                            display();
                            break;
            case 'b':
                          light[type_light][2] -= 0.1;
                            light[type_light][2] = (light[type_light][2] >
1.0) ? 1.0 : light[type_light][2];
                            light[type_light][2] = (light[type_light][2] <
0.0) ? 0.0 : light[type_light][2];
                            printf("%f \n",light[type_light][2]);
                            display();
                            break;
            case 's':
                         material_shininess[0] -= 1;
                            printf("%f \n",material_shininess[0]);
                            display();
                            break;
            case 'S':
                         material_shininess[0] += 1;
                            printf("%f \n",material_shininess[0]);
                            display();
                            break;
            case 'c':
                         spot_cutoff -= 1;
                            printf("%f \n",spot_cutoff);
                            display();
                            break;
            case 'C':
                         spot_cutoff += 1;
                            printf("%f \n",spot_cutoff);
                            display();
                            break;
            case 'e':
                         spot_exponent -= 0.1;
                            printf("%f \n",spot_exponent);
                            display();
                            break;
            case 'E':
                         spot_exponent += 0.1;
                            printf("%f \n",spot_exponent);
                            display();
                            break;
            case 'a':
                         c_attenuation -= 0.1;
                            printf("%f \n",c_attenuation);
                            display();
                            break;
```

```
            case 'A':
                        c_attenuation += 0.1;
                          printf("%f \n",c_attenuation);
                          display();
                          break;
            case 'l':
                        l_attenuation -= 0.1;
                          printf("%f \n",l_attenuation);
                          display();
                          break;
            case 'L':
                        l_attenuation += 0.1;
                          printf("%f \n",l_attenuation);
                          display();
                          break;
            case 'q':
                        q_attenuation -= 0.1;
                          printf("%f \n",q_attenuation);
                          display();
                          break;
            case 'Q':
                        q_attenuation += 0.1;
                          printf("%f \n",q_attenuation);
                          display();
                          break;

      }
}

void main(int argc, char **argv)
{
   xmin = ymin = -4;
   xmax = ymax =  4;
   ptx = pty = 10;
   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
   glutInitWindowSize(400,400);
   glutInitWindowPosition(50,50);
   glutCreateWindow("Cubo 3D");
   glutDisplayFunc(display);
   inicia_config();
   glutMouseFunc(botao_mouse);
   glutMotionFunc(mov_mouse);
   glutKeyboardFunc(le_tecla);
   glutMainLoop();
}
```

### 2.4.2 – Módulo `ilumina_funcao.cpp`

```
#include <gl/glut.h>
#include <math.h>
#include "ilumina_set.h"

float xmin,xmax,ymin,ymax;
float ptx,pty;

float funcao(float x,float y)
{
//    return(sqrt(x*x+y*y));
//    return(y*y-x*x);
//    return(x*x+y*y);
//    return(-x-y+1);
      return(sin(sqrt(x*x+y*y)));
}
```

```
void draw_eixos()
{
      glColor3f(1.0,0.0,0.0);
      glBegin(GL_LINES);
        glVertex3f(10.0,0.0,0.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
      glColor3f(0.0,1.0,0.0);
      glBegin(GL_LINES);
        glVertex3f(0.0,10.0,0.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
      glColor3f(0.0,0.0,1.0);
      glBegin(GL_LINES);
        glVertex3f(0.0,0.0,10.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
}

void normalv(float *v,float x,float y,float dx, float dy)
{
      float v1[3],v2[3],norma;

      v1[0] = dx ;
      v1[1] = 0.0;
      v1[2] = funcao(x+dx,y)-funcao(x,y);
      v2[0] = 0.0;
      v2[1] = dy;
      v2[2] = funcao(x,y+dy)-funcao(x,y);

      v[0] = v1[1] * v2[2] - v1[2] * v2[1];
      v[1] = v1[0] * v2[2] - v1[2] * v2[0];
      v[2] = v1[0] * v2[1] - v1[1] * v2[0];

      norma = sqrt(v[0] * v[0] + v[1] *v[1] + v[2] * v[2]);
      v[0] = v[0] / norma;
      v[1] = v[1] / norma;
      v[2] = v[2] / norma;
}

float dfx(float x,float y)
{
      return(-2*x);
}

float dfy(float x,float y)
{
      return(2*y);
}

void draw_normal(float x,float y)
{
   float z,z1,z2;

   desabilita_lighting();
   glColor3f(1.0,1.0,0.0);
   z1 = dfx(x,y)/10.0;  /* Derivada parcial com relacao a x */
   z2 = dfy(x,y)/10.0;  /* Derivada parcial com relacao a y */
   z  = funcao(x,y);
   glBegin(GL_LINES);
      glVertex3f(x,y,z);
      glVertex3f(x-z1,y-z2,z+0.1);
   glEnd();
   habilita_lighting();
}
```

```
void draw_funcao()
{
   float x,y;
   float stepx,stepy;
   float z0,z1,z2,z3;
   float v[3];

   stepx = (xmax-xmin)/ptx;
   stepy = (ymax-ymin)/pty;

   habilita_lighting();
   glColorMaterial(GL_FRONT, GL_DIFFUSE);
   glColor3f(1.0,1.0,0.0);
   glColorMaterial(GL_BACK , GL_DIFFUSE);
   glColor3f(1.0,0.0,0.2);
   for(x=xmin;x<=xmax;x+=stepx)
   {
         for(y=ymin;y<=ymax;y+=stepy)
         {
               z0 = funcao(x,y);
           z1 = funcao(x+stepx,y);
               z2 = funcao(x+stepx,y+stepy);
               z3 = funcao(x,y+stepy);
               glBegin(GL_QUADS);
                    normalv(v,x,y,stepx,stepy);
                    glNormal3f(v[0],v[1],v[2]);
                  glVertex3f(x,y,z0);
                    normalv(v,x+stepx,y,stepx,stepy);
                    glNormal3f(v[0],v[1],v[2]);
                    glVertex3f(x+stepx,y,z1);
                    normalv(v,x+stepx,y+stepy,stepx,stepy);
                    glNormal3f(v[0],v[1],v[2]);
                    glVertex3f(x+stepx,y+stepy,z2);
                    normalv(v,x,y+stepy,stepx,stepy);
                    glNormal3f(v[0],v[1],v[2]);
                    glVertex3f(x,y+stepy,z3);
               glEnd();
               draw_normal(x,y);
         }
   }
   desabilita_lighting();
}
```

### 2.4.3- Módulo `ilumina_set.cpp`

```
#include <gl\glut.h>
#include <stdio.h>




GLfloat   light[8][4] = {{ 0.3, 0.3, 0.3, 1.0 },  /* Luz Ambiente
*/
                         { 1.0, 1.0, 1.0, 1.0 },  /* Luz difusa
*/
                         { 1.0, 1.0, 1.0, 0.0 },  /* Posicao da luz
*/
                         { 1.0, 1.0, 1.0, 1.0 },  /* Luz Especular
*/
                         { 0.4, 0.4, 0.4, 1.0 },  /* Material Ambiente
*/
                         { 0.8, 0.8, 0.8, 1.0 },  /* Material Difusa
*/
                         { 1.0, 1.0, 1.0, 1.0 },  /* Material Especular
```

```
*/
                        { 0.0, 0.0, 0.0, 1.0 }}; /* Iluminacao do
Ambiente */


GLfloat    spot_direction[3]     = {0.0,0.0,-10.0};
GLfloat    spot_cutoff           = 180.0;
GLfloat    spot_exponent         =   0.0;
GLfloat    c_attenuation         =   1.0;
GLfloat    l_attenuation         =   0.0;
GLfloat    q_attenuation         =   0.0;
GLfloat    material_shininess[1] = { 60.0 } ;

/*--------------------------------------------------------------------*/
void desabilita_lighting()
/*--------------------------------------------------------------------*/
/*--------------------------------------------------------------------*/
{
  glDisable(GL_LIGHTING);
  glDisable(GL_COLOR_MATERIAL);
  glDisable(GL_LIGHT0);
}

/*--------------------------------------------------------------------*/
void habilita_lighting()
/*--------------------------------------------------------------------*/
/*--------------------------------------------------------------------*/
{

  glShadeModel(GL_FLAT);

  glEnable(GL_NORMALIZE);
  glLightfv(GL_LIGHT0, GL_AMBIENT               ,  light[0]);
  glLightfv(GL_LIGHT0, GL_DIFFUSE               ,  light[1]);
  glLightfv(GL_LIGHT0, GL_POSITION              , light[2]);
  glLightfv(GL_LIGHT0, GL_SPECULAR              , light[3]);
  glLightf (GL_LIGHT0, GL_SPOT_CUTOFF           , spot_cutoff);
  glLightf (GL_LIGHT0, GL_SPOT_EXPONENT         , spot_exponent);
  glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION        , spot_direction);
  glLightf (GL_LIGHT0, GL_CONSTANT_ATTENUATION  , c_attenuation);
  glLightf (GL_LIGHT0, GL_LINEAR_ATTENUATION    , l_attenuation);
  glLightf (GL_LIGHT0, GL_QUADRATIC_ATTENUATION , q_attenuation);
  glEnable(GL_LIGHT0);

  glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
  glLightModeli(GL_LIGHT_MODEL_TWO_SIDE    , GL_TRUE);
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT,  light[7]);
  glEnable(GL_LIGHTING);

  glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT,   light[4]);
  glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR,  light[5]);
  glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE,   light[6]);
  glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, material_shininess );
  glEnable(GL_COLOR_MATERIAL);
  glColorMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE);
}
```

### 2.4.4 – Módulo `ilumina_funcao.h`

```
void draw_eixos();
void draw_funcao();
```

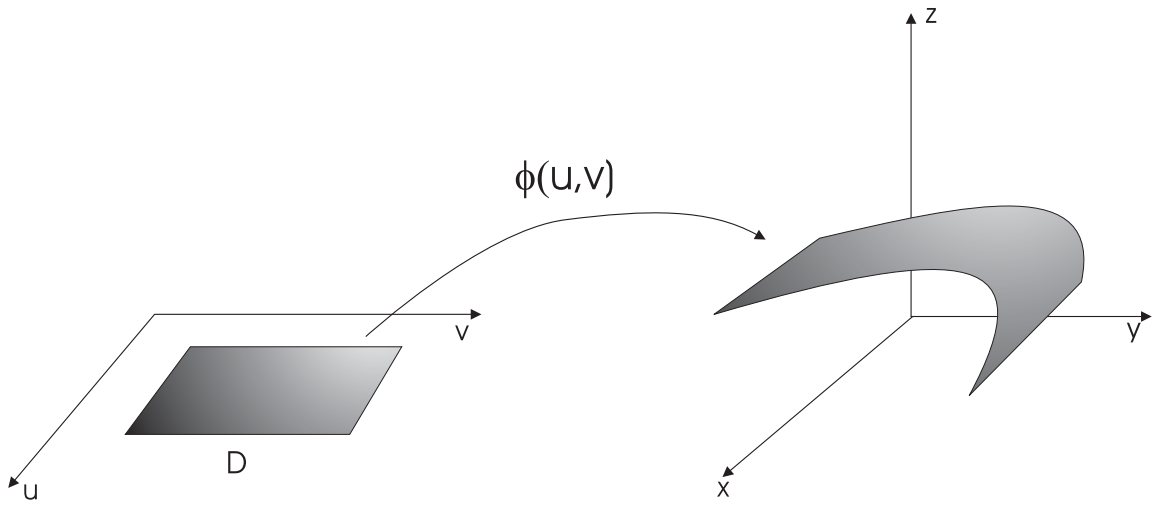
### 2.4.5 – Módulo `ilumina_set.h`

```
void desabilita_lighting();
void habilita_lighting();
void le_tecla(unsigned char key, int x, int y);
```

# 3- SUPERFÍCIES PARAMETRIZADAS

Para compreender a idéia de uma superfície parametrizada, considere D uma região do plano, cujas variáveis são denotadas por $(u,v)$. A cada par $(u,v)$ de D vamos associar um ponto $\phi(u,v)$ no espaço tri-dimensional, o qual pode ser escrito em termos de suas funções coordenadas por:

$$\phi(u,v) = (x(u,v), y(u,v), z(u,v))$$




Uma superfície parametrizada é uma aplicação $\phi : D \subset R^2 \rightarrow R^3$ onde D é algum domínio em $R^2$. A superfície S correspondente a função $\phi$ é a imagem $S = \phi(D)$. A superfície parametrizada depende de dois parâmetros (u, v).

## 3.1– Visualização de superfícies na forma paramétrica

.

Para visualizar superfícies dadas na forma paramétrica, utilizaremos o programa abaixo. O programa é composto de dois módulos: `sup_main.cpp` e `sup_param.cpp`. O módulo `sup_main.cpp` é responsável pelas tarefas de iluminação e definição do ambiente OpenGL. O segundo módulo `sup_param.cpp` define a parametrização e através da função draw_superficie(), visualiza a superfície desejada.

```
/* * * * * * * * * * * * * * * * * * * * * * * * */
/*                                               */
/*        Superficies Parametrizadas             */
/*                                               */
/* Modulo: Sup_main.cpp                          */
/*                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * */

#include <gl\glut.h>
#include <stdio.h>
#include "Sup_param.h"

extern float ptx,pty;
extern float umin,umax,vmin,vmax;

int   type_light = 0;
```

```
int   xb,yb,xm,ym;
float gamma=90.0,phi=45.0,theta=135.0;
float scale = 1.0;

GLfloat   light[8][4] = {{ 0.1, 0.0, 0.0, 1.0 },
                         { 0.3, 0.3, 0.0, 1.0 },
                         { 1.0, 1.0, 1.0, 0.0 },
                         { 0.0, 0.0, 1.0, 1.0 },
                         { 0.2, 0.4, 0.0, 1.0 },
                         { 1.0, 1.0, 0.0, 1.0 },
                         { 0.3, 0.8, 1.0, 1.0 },
                         { 0.5, 0.0, 0.1, 1.0 }};

GLfloat  light1[8][4] = {{ 0.3, 0.3, 0.3, 1.0 },
                         { 1.0, 1.0, 1.0, 1.0 },
                         { 0.0, 0.0, -1.0, 0.0 },
                         { 1.0, 1.0, 1.0, 1.0 },
                         { 0.0, 0.0, 1.0, 1.0 },
                         { -1.0, -1.0, -1.0, 1.0 },
                         { 0.0, 0.1, 1.0, 1.0 },
                         { 0.5, 0.5, 0.5, 1.0 }};
GLfloat    material_shininess[1] = { 60.0 } ;
GLfloat    lmodel_ambient[4]     = {0.5,0.5,0.5,1.0};
GLfloat    spot_cutoff           = 90.0;
GLfloat    spot_exponent         = 0.0;
GLfloat    c_attenuation         = 1.0;
GLfloat    l_attenuation         = 0.0;
GLfloat    q_attenuation         = 0.0;

/*---------------------------------------------------------------------*/
void desabilita_lighting()
/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------*/
{
  glDisable(GL_LIGHTING);
  glDisable(GL_COLOR_MATERIAL);
  glDisable(GL_LIGHT0);
  glDisable(GL_LIGHT1);
  glDisable(GL_LIGHT2);
}

/*---------------------------------------------------------------------*/
void habilita_lighting()
/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------*/
{

  glShadeModel(GL_SMOOTH);

  glLightfv(GL_LIGHT0, GL_AMBIENT,  light[0]);
  glLightfv(GL_LIGHT0, GL_DIFFUSE,  light[1]);
//  glLightfv(GL_LIGHT0, GL_POSITION, light[2]);
  glLightfv(GL_LIGHT0, GL_SPECULAR, light[3]);
  glLightf (GL_LIGHT0, GL_SPOT_CUTOFF   , spot_cutoff);
  glLightf (GL_LIGHT0, GL_SPOT_EXPONENT , spot_exponent);
  glLightf (GL_LIGHT0, GL_CONSTANT_ATTENUATION  , c_attenuation);
  glLightf (GL_LIGHT0, GL_LINEAR_ATTENUATION    , l_attenuation);
  glLightf (GL_LIGHT0, GL_QUADRATIC_ATTENUATION , q_attenuation);
  glEnable(GL_LIGHT0);
```

```
  glLightfv(GL_LIGHT1, GL_AMBIENT,  light1[0]);
  glLightfv(GL_LIGHT1, GL_DIFFUSE,  light1[1]);
  glLightfv(GL_LIGHT1, GL_POSITION, light1[2]);
  glLightfv(GL_LIGHT1, GL_SPECULAR, light1[3]);
  glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, light1[4]);
  glLightf (GL_LIGHT1, GL_SPOT_CUTOFF   , spot_cutoff);
  glLightf (GL_LIGHT1, GL_SPOT_EXPONENT , spot_exponent);
  glLightf (GL_LIGHT1, GL_CONSTANT_ATTENUATION  , c_attenuation);
  glLightf (GL_LIGHT1, GL_LINEAR_ATTENUATION    , l_attenuation);
  glLightf (GL_LIGHT1, GL_QUADRATIC_ATTENUATION , q_attenuation);
  glEnable(GL_LIGHT1);

  glLightfv(GL_LIGHT2, GL_AMBIENT,  light[0]);
  glLightfv(GL_LIGHT2, GL_DIFFUSE,  light[1]);
  glLightfv(GL_LIGHT2, GL_POSITION, light1[5]);
  glLightfv(GL_LIGHT2, GL_SPECULAR, light[3]);
  glLightf (GL_LIGHT2, GL_SPOT_CUTOFF   , spot_cutoff);
  glLightf (GL_LIGHT2, GL_SPOT_EXPONENT , spot_exponent);
  glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, light1[6]);
  glLightf (GL_LIGHT2, GL_CONSTANT_ATTENUATION  , c_attenuation);
  glLightf (GL_LIGHT2, GL_LINEAR_ATTENUATION    , l_attenuation);
  glLightf (GL_LIGHT2, GL_QUADRATIC_ATTENUATION , q_attenuation);
  glEnable(GL_LIGHT2);

  glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
  glLightModeli(GL_LIGHT_MODEL_TWO_SIDE    , GL_TRUE);
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT,  light[7]);
  glEnable(GL_LIGHTING);

  glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT,   light[4]);
  glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR,  light[5]);
  glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE,   light[6]);
  glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, material_shininess );
  glEnable(GL_COLOR_MATERIAL);
  glColorMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE);

}

void display()
{
      glClearColor(0.0,0.0,0.0,0.0);
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT  );
      glEnable(GL_NORMALIZE);
      glColor3f(1.0,1.0,0.0);
      draw_eixos();
      draw_superficie();
      glFlush();
      glutSwapBuffers();
}

void inicia_config()
{

      glEnable(GL_DEPTH_TEST);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
    glLightfv(GL_LIGHT0, GL_POSITION, light[2]);
    glScalef(scale,scale,scale);
```

```
      glRotatef(gamma,0.0,0.0,1.0);
      glRotatef(phi,0.0,1.0,0.0);
      glRotatef(theta,0.0,0.0,1.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-2.0,2.0,-2.0,2.0,-10.0,10.0);
}

void botao_mouse(int b,int state,int x, int y)
{

      switch(b) {
         case GLUT_LEFT_BUTTON:
               switch(state) {
                  case GLUT_DOWN:
                                    xb = x;
                                      yb = y;
                                      break;
                  case GLUT_UP:
                              theta = theta + xm - xb;
                              phi   = phi   - ym + yb ;
                                      break;
               }
               break;
      }
}

void mov_mouse(int x, int y)
{
      xm = x;
      ym = y;
    theta = theta + xm - xb;
    phi   = phi   - ym + yb ;
      inicia_config();
      theta = theta - xm + xb;
      phi   = phi   + ym - yb;
      display();
}

void le_tecla(unsigned char key, int x, int y)
{
      switch(key)
      {
           case '+':
                        scale+=0.2;
                            inicia_config();
                            display();
                        break;
           case '-':
                        scale-=0.2;
                            inicia_config();
                            display();
                        break;
      }
}

void main(int argc, char **argv)
{
      glutInit(&argc,argv);
```

```
      glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
      glutInitWindowSize(400,400);
      glutInitWindowPosition(50,50);
      glutCreateWindow("Cubo 3D");
      glutDisplayFunc(display);
    inicia_config();
      glutMouseFunc(botao_mouse);
      glutMotionFunc(mov_mouse);
      glutKeyboardFunc(le_tecla);
      glutMainLoop();
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                     */
/*        Superficies Parametrizadas                   */
/*                                                     */
/* Modulo: Sup_param.cpp                               */
/*                                                     */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <gl/glut.h>
#include <math.h>
#include "Sup_param.h"

float umin = 0.0 ;
float umax = 6.283;
float vmin = 1.5 ;
float vmax = 4.28;
float ptx  = 20  ;
float pty  = 20  ;

void parametrizacao(float u,float v,float *x,float *y,float *z)
{
    *x = -sin(u)*(2+cos(v));
      *y =  cos(u)*(2+cos(v));
      *z = 2+sin(v);
}

void draw_eixos()
{
      glColor3f(1.0,0.0,0.0);
      glBegin(GL_LINES);
        glVertex3f(10.0,0.0,0.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
      glColor3f(0.0,1.0,0.0);
      glBegin(GL_LINES);
        glVertex3f(0.0,10.0,0.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
      glColor3f(0.0,0.0,1.0);
      glBegin(GL_LINES);
        glVertex3f(0.0,0.0,10.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
}

void normalv(float *n,float u,float v,float dx, float dy)
```

```
{
      float v1[3],v2[3],norma;
    float x1,y1,z1;
    float x2,y2,z2;
    float x3,y3,z3;

      parametrizacao(u,v,&x1,&y1,&z1);
      parametrizacao(u,v+dy,&x2,&y2,&z2);
      parametrizacao(u+dx,v,&x3,&y3,&z3);
      v1[0] = x2-x1 ;
      v1[1] = y2-y1;
      v1[2] = z2-z1;
      v2[0] = x3-x1;
      v2[1] = y3-y1;
      v2[2] = z3-z1;

      n[0] = v1[1] * v2[2] - v1[2] * v2[1];
      n[1] = v1[0] * v2[2] - v1[2] * v2[0];
      n[2] = v1[0] * v2[1] - v1[1] * v2[0];

      norma = sqrt(n[0] * n[0] + n[1] *n[1] + n[2] * n[2]);
      n[0] = n[0] / norma;
      n[1] = n[1] / norma;
      n[2] = n[2] / norma;
}

void draw_superficie()
{
   float x,y,z;
   float u,v;
   float stepu,stepv;
   float z0,z1,z2,z3;
   float n[3];

   stepu = (umax-umin)/ptx;
   stepv = (vmax-vmin)/pty;

   habilita_lighting();
   glColorMaterial(GL_FRONT, GL_DIFFUSE);
   glColor3f(1.0,1.0,0.0);
   glColorMaterial(GL_BACK , GL_DIFFUSE);
   glColor3f(1.0,0.0,0.2);
   for(u=umin;u<=umax;u+=stepu)
   {
         for(v=vmin;v<=vmax;v+=stepv)
         {

               glBegin(GL_QUADS);
                  parametrizacao(u,v,&x,&y,&z);
                    normalv(n,u,v,stepu,stepv);
               glNormal3f(n[0],n[1],n[2]);
                  glVertex3f(x,y,z);
                  parametrizacao(u+stepu,v,&x,&y,&z);
                    normalv(n,u+stepu,v,stepu,stepv);
               glNormal3f(n[0],n[1],n[2]);
                  glVertex3f(x,y,z);
                  parametrizacao(u+stepu,v+stepv,&x,&y,&z);
                    normalv(n,u+stepu,v+stepv,stepu,stepv);
               glNormal3f(n[0],n[1],n[2]);
```

```
                    glVertex3f(x,y,z);
                    parametrizacao(u,v+stepv,&x,&y,&z);
                      normalv(n,u,v+stepv,stepu,stepv);
                glNormal3f(n[0],n[1],n[2]);
                    glVertex3f(x,y,z);
                 glEnd();
            }
    }
    desabilita_lighting();
}
```

## 3.2– Exercícios

1) Utilizando o programa anterior, visualize as seguintes superfícies e identifique-as:

| | | | |
|---|---|---|---|
| a) $\begin{cases} x = \text{sen}(\varphi)\cos(\theta) \\ y = \text{sen}(\varphi)\text{sen}(\theta) \\ z = \cos(\varphi) \end{cases}$ $0 \le \varphi \le \pi$ $0 \le \theta \le 2\pi$ | b) $\begin{cases} x = 2u\cos(\theta) \\ y = 2u\,\text{sen}(\theta) \\ z = 2u \end{cases}$ $0 \le u \le 4$ $0 \le \theta \le 2\pi$ | c) $\begin{cases} x = 2u\cos(\theta) \\ y = 2u\,\text{sen}(\theta) \\ z = 4u^2 \end{cases}$ $0 \le u \le 1$ $0 \le \theta \le 2\pi$ | d) $\begin{cases} x = \cos(\theta) \\ y = \text{sen}(\theta) \\ z = u \end{cases}$ $0 \le u \le 4$ $0 \le \theta \le 2\pi$ |
| e) $\begin{cases} x = 2u\cos(\theta) \\ y = 2u\,\text{sen}(\theta) \\ z = 2u \end{cases}$ $0 \le u \le 4$ $0 \le \theta \le 2\pi$ | f) $\begin{cases} x = \cos(u) + v\cos(\frac{u}{2})\cos(u) \\ y = \text{sen}(u) + v\cos(\frac{u}{2})\text{sen}(u) \\ z = v\,\text{sen}(\frac{u}{2}) \end{cases}$ $-0.2 \le v \le 0.2$ $0 \le u \le 2\pi$ | g) $\begin{cases} x = u \\ y = v \\ z = \sqrt{u^2 + v^2} \end{cases}$ $0 \le u \le 1$ $0 \le v \le 1$ | |

2) Visualize os vetores normais sobre a superfície. Observe o exemplo da letra f).

# 4- INTERPOLAÇÃO

Considere o seguinte problema:

- Dados n pontos distintos no plano $(x_i, y_i)$ com os respectivos valores $z_i$ associados, determine uma função f tal que:

$$f(x_i, y_i) = z_i \qquad i = 0, \ldots, n-1$$

Entre as várias soluções possíveis, apresentaremos o método de Shepard.

## 4 .1– Interpolação Utilizando o Método de Shepard

O método de Shepard define a função por:

$$f(x,y) = \sum_{i=0}^{n-1} z_i v_i(x,y)$$

onde

$$v_i(x,y) = \frac{w_i(x,y)}{\sum_{k=0}^{n-1} w_k(x,y)} \text{ e } \begin{cases} w_i = \dfrac{1}{r_i^q} \\ r_i = \sqrt{(x-x_i)^2 + (y-y_i)^2} \end{cases}$$

Uma forma numericamente mais estável para se calcular os $v_i$ é dada por:

$$v_i(x,y) = \frac{\prod_{j=0, j\neq i}^{n-1} r_j^q}{\sum_{k=0}^{n-1} \left( \prod_{j=0, j\neq k}^{n-1} r_j^q \right)}$$

## 4 .2– Propriedades do Método de Shepard

Destacamos duas propriedades importantes do método de Shepard:

1) Desde que $v_i(x,y) \geq 0$ e $\sum_{k=0}^{n-1} v_k(x,y) = 1$ então

$$\min_i z_i \leq f(x,y) \leq \max_i z_i$$

2) Se $z_i \geq 0$, $i = 0, \ldots, n-1 \Rightarrow f(x,y) \geq 0$

3) Se $z_i = c$, $i = 0, \ldots, n-1 \Rightarrow f(x,y) = c$

## 4 .3– Programa Interpolação Método de Shepard

```
/* * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                   */
/*        Interpolação                               */
/*                                                   */
/* Modulo: interp.cpp                                */
/*                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * */

#include <gl\glut.h>
#include <stdio.h>
#include "interp.h"

int   xb,yb,xm,ym;
float gamma=90.0,phi=45.0,theta=135.0;
float scale = 1.0;

int   type_light = 0;

int    N;
float *xp;
float *yp;
float *zp;

GLfloat    visgl_lmodel_ambient[]     = { 0.1, 0.1, 0.1, 1.0 }    ;
GLfloat    material_shininess[1]= { 60.0 } ;
GLfloat   light[8][4] = {{ 0.3, 0.3, 0.3, 1.0 },
                         { 1.0, 1.0, 1.0, 1.0 },
                         { 0.0, 0.0, 1.0, 0.0 },
                         { 1.0, 1.0, 1.0, 1.0 },
                         { 0.4, 0.4, 0.4, 1.0 },
                         { 0.9, 0.9, 0.9, 1.0 },
                         { 1.0, 1.0, 1.0, 1.0 },
                         { 0.0, 0.0,-1.0, 0.0 }};


/*---------------------------------------------------------------------*/
void desabilita_lighting()
/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------*/
{
  glDisable(GL_LIGHTING);
  glDisable(GL_COLOR_MATERIAL);
  glDisable(GL_LIGHT0);
  glDisable(GL_LIGHT1);
}

/*---------------------------------------------------------------------*/
void habilita_lighting()
/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------*/
{

//  glShadeModel(GL_FLAT);

  glLightfv(GL_LIGHT0, GL_AMBIENT,  light[0]);
```

```
  glLightfv(GL_LIGHT0, GL_DIFFUSE,  light[1]);
  glLightfv(GL_LIGHT0, GL_POSITION, light[2]);
  glLightfv(GL_LIGHT0, GL_SPECULAR, light[3]);
  glEnable(GL_LIGHT0);

  glLightfv(GL_LIGHT1, GL_AMBIENT,  light[0]);
  glLightfv(GL_LIGHT1, GL_DIFFUSE,  light[1]);
  glLightfv(GL_LIGHT1, GL_POSITION, light[7]);
  glLightfv(GL_LIGHT1, GL_SPECULAR, light[3]);
  glEnable(GL_LIGHT1);

  glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
  glLightModeli(GL_LIGHT_MODEL_TWO_SIDE    , GL_TRUE);
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT,  visgl_lmodel_ambient);
  glEnable(GL_LIGHTING);

  glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT,   light[4]);
  glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR,  light[5]);
  glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE,   light[6]);
  glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, material_shininess );
  glEnable(GL_COLOR_MATERIAL);
  glColorMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE);

}

void display()
{
      glClearColor(0.0,0.0,0.0,0.0);
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT  );
      glEnable(GL_NORMALIZE);
      glColor3f(1.0,1.0,0.0);
      draw_eixos();
      draw_funcao();
      draw_points();
      glFlush();
      glutSwapBuffers();
}

void inicia_config()
{
      glEnable(GL_DEPTH_TEST);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glScalef(scale,scale,scale);
      glRotatef(gamma,0.0,0.0,1.0);
      glRotatef(phi,0.0,1.0,0.0);
      glRotatef(theta,0.0,0.0,1.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-2.0,2.0,-2.0,2.0,-10.0,10.0);
}

void botao_mouse(int b,int state,int x, int y)
{

      switch(b) {
         case GLUT_LEFT_BUTTON:
               switch(state) {
                  case GLUT_DOWN:
```

```
                                    xb = x;
                                      yb = y;
                                      break;
                  case GLUT_UP:
                              theta = theta + xm - xb;
                              phi   = phi   - ym + yb ;
                                      break;
               }
               break;
      }
}

void mov_mouse(int x, int y)
{
      xm = x;
      ym = y;
    theta = theta + xm - xb;
    phi   = phi   - ym + yb ;
      inicia_config();
      theta = theta - xm + xb;
      phi   = phi   + ym - yb;
      display();
}

void le_tecla(unsigned char key, int x, int y)
{
      switch(key)
      {
           case '+':
                      scale+=0.2;
                          inicia_config();
                          display();
                      break;
           case '-':
                      scale-=0.2;
                          inicia_config();
                          display();
                      break;


      }
}

void main(int argc, char **argv)
{
      int i;

      printf("\n Numero de pontos = ");
      scanf("%d",&N);
      xp = (float *) malloc(N*sizeof(float));
      yp = (float *) malloc(N*sizeof(float));
      zp = (float *) malloc(N*sizeof(float));

      for(i=0;i<N;i++) {
      printf("\n x,y,z = ");
            scanf("%f,%f,%f",&xp[i],&yp[i],&zp[i]);
      }

      glutInit(&argc,argv);
```

```
      glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
      glutInitWindowSize(400,400);
      glutInitWindowPosition(50,50);
      glutCreateWindow("Cubo 3D");
      glutDisplayFunc(display);
      inicia_config();
      glutMouseFunc(botao_mouse);
      glutMotionFunc(mov_mouse);
      glutKeyboardFunc(le_tecla);
      glutMainLoop();
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * */
/*                                               */
/*        Interpolação                           */
/*                                               */
/* Modulo: Interp_f.cpp                          */
/*                                               */
/* * * * * * * * * * * * * * * * * * * * * * * */

#include <gl\glut.h>
#include <stdio.h>
#include <math.h>
#include "interp.h"

extern int   xb,yb,xm,ym;

extern int    N;
extern float *xp;
extern float *yp;
extern float *zp;

int q = 1.0;

void draw_eixos()
{

      glColor3f(1.0,0.0,0.0);
      glBegin(GL_LINES);
        glVertex3f(10.0,0.0,0.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
      glColor3f(0.0,1.0,0.0);
      glBegin(GL_LINES);
        glVertex3f(0.0,10.0,0.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
      glColor3f(0.0,0.0,1.0);
      glBegin(GL_LINES);
        glVertex3f(0.0,0.0,10.0);
        glVertex3f(0.0,0.0,0.0);
      glEnd();
}

float interp(float xx,float yy)
{
      int    i,j;
```

```
      float  s = 0.0;
      float sv = 0.0;
    float *p;

      p = (float *)malloc(N*sizeof(float));

      for(j=0;j<N;j++) {
            p[j] = 1.0;
          for(i=0;i<N;i++) {
                if  (i != j)
                 p[j] *= pow(sqrt((xx-xp[i])*(xx-xp[i])+(yy-yp[i])*(yy-
yp[i])),q);
            }
    }

      for(j=0;j<N;j++)
            sv += p[j];

      for(i=0;i<N;i++)
            s+=zp[i]*p[i]/sv;

      free(p);
      return(s);
}

void normalv(float *v,float x,float y,float dx, float dy)
{
      float v1[3],v2[3];

      v1[0] = dx ;
      v1[1] = 0.0;
      v1[2] = interp(x+dx,y)-interp(x,y);
      v2[0] = 0.0;
      v2[1] = dy;
      v2[2] = interp(x,y+dy)-interp(x,y);

      v[0] = v1[1] * v2[2] - v1[2] * v2[1];
      v[1] = v1[0] * v2[2] - v1[2] * v2[0];
      v[2] = v1[0] * v2[1] - v1[1] * v2[0];
}

void draw_normal(float x,float y,float dx,float dy)
{
   float z,v[3];

   desabilita_lighting();
   glColor3f(1.0,1.0,0.0);
   normalv(v,x,y,dx,dy);
   z  = interp(x,y);
   glBegin(GL_LINES);
    glVertex3f(x,y,z);
      glVertex3f(x+v[0],y+v[1],z+v[2]);
   glEnd();
   habilita_lighting();
}


void draw_funcao()
{
```

```
      int i;
      float v[3];
    float dx,dy;
      float px   = 20;
      float py   = 20;
      float x,y;
      float xmin = -2.0;
      float ymin = -2.0;
      float xmax =  2.0;
      float ymax =  2.0;

      dx = (xmax - xmin)/px;
      dy = (ymax - ymin)/py;

      habilita_lighting();

    glColorMaterial(GL_FRONT, GL_DIFFUSE);
      glColor3f(1.0,0.0,0.0);
      glColorMaterial(GL_BACK, GL_DIFFUSE);
      glColor3f(0.0,0.0,1.0);
      for(x=xmin;x<xmax;x+=dx)
      {
            for(y=ymin;y<ymax;y+=dy)
            {
            glBegin(GL_QUADS);
                      normalv(v,x,y,dx,dy);
                        glNormal3f(v[0],v[1],v[2]);
                      glVertex3f(x   ,y   ,interp(x   ,y   ));
                      normalv(v,x+dx,y,dx,dy);
                        glNormal3f(v[0],v[1],v[2]);
                        glVertex3f(x+dx,y   ,interp(x+dx,y   ));
                      normalv(v,x+dx,y+dy,dx,dy);
                        glNormal3f(v[0],v[1],v[2]);
                        glVertex3f(x+dx,y+dy,interp(x+dx,y+dy));
                      normalv(v,x,y+dy,dx,dy);
                        glNormal3f(v[0],v[1],v[2]);
                        glVertex3f(x   ,y+dy,interp(x   ,y+dy));
                  glEnd();
                  draw_normal(x,y,dx,dy);
                  glColor3f(1.0,0.0,0.0);
            }
      }
      desabilita_lighting();
      for(i=0;i<N;i++)
            printf("z[%d]=%f, zcalc=%f\n",i,zp[i],interp(xp[i],yp[i]));
}

void draw_points()
{
      int i;

      glColor3f(0.0,0.0,1.0);
      glPointSize(3.0);
      glBegin(GL_POINTS);
         for(i=0;i<N;i++)
             glVertex3f(xp[i],yp[i],zp[i]);
      glEnd();
}
```