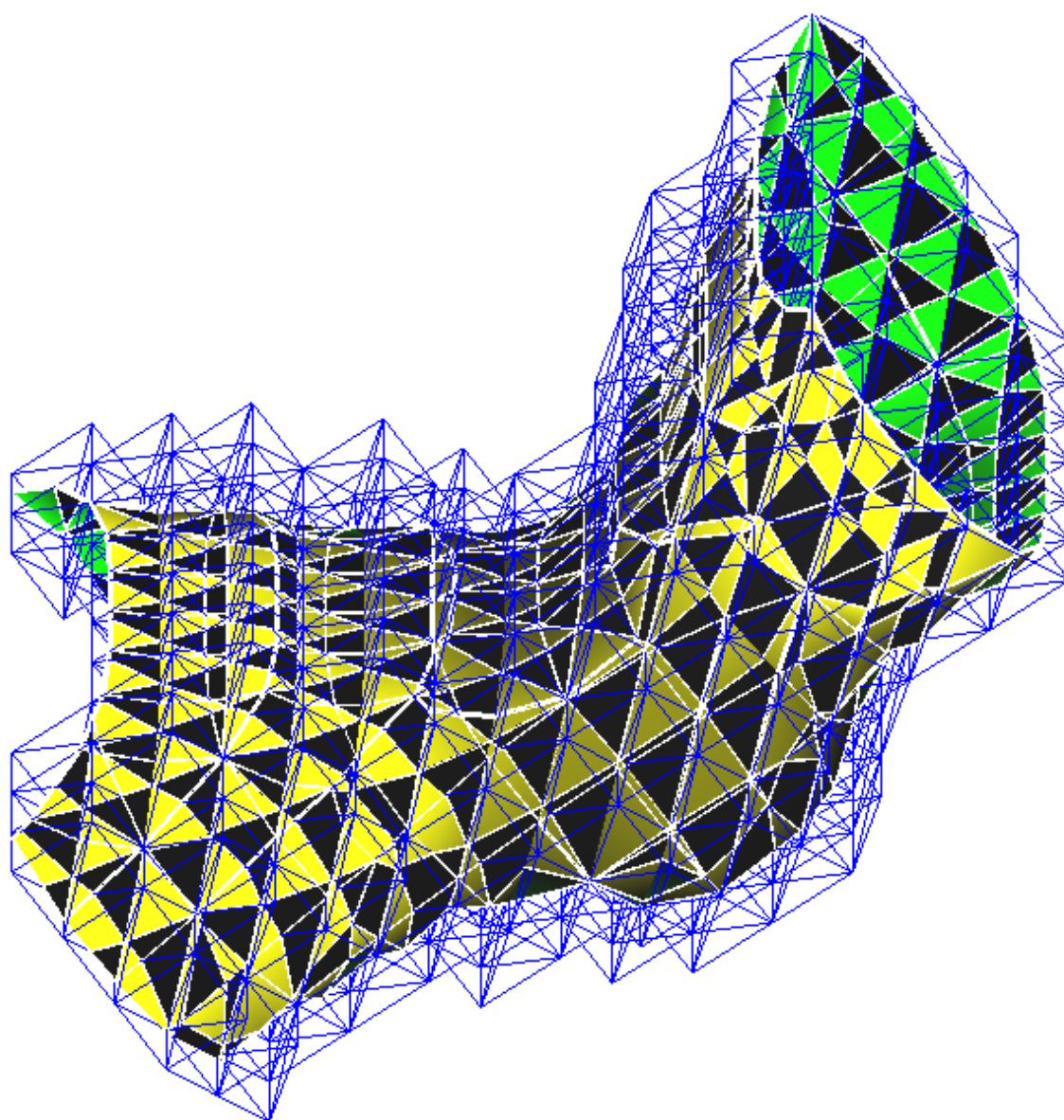


SINÉSIO PESCO
HÉLIO CORTES VIEIRA LOPES
DIRCE UESU
GEOVAN TAVARES DOS SANTOS

**FUNDAMENTOS DE MATEMÁTICA PARA
COMPUTAÇÃO GRÁFICA**



Elementos Matemáticos para Computação Gráfica

Referências:

- 1 – Fundamentos de Matemática para Computação Gráfica
Autor: Sinésio Pesco
Hélio Lopes
Dirce Uesu Pesco
Geovan Tavares
Apostila que será disponibilizada no xerox da Taisho.
- 2 - C – A linguagem de programação
Autor: Brian W. Kernighan
Dennis M. Ritchie
Editora: Campus LTDA.
- 3 – The OpenGL Utility Toolkit (GLUT) Programming Interface
Autor: Mark J. Kilgard
Silicon Graphics, Inc.
Novembro 13, 1996.
- 4 – Open GL Programming for the X Window System
Autor: Mark J. Kilgard
Editora: Addison-Wesley Developers Press.

ÍNDICE

<u>CAPÍTULO I – INTRODUÇÃO A LINGUAGEM C</u>	4
1- ESTRUTURA BÁSICA DE UM PROGRAMA C	4
1.1 - Introdução.....	4
1.2 - Variáveis.....	4
1.3 - A função <code>printf()</code>	5
1.4 - A função <code>scanf()</code>	6
1.5 - Operadores.....	7
1.6 - Comentários.....	8
2 - LAÇOS (LOOPS)	9
2.1 - Comando <code>for(;;) {}</code>	9
2.2 - Comando <code>do {} while();</code>	10
2.2 - Comando <code>while() {};</code>	11
2.4 - Comando <code>break</code> e <code>continue</code>	12
3 - DECISÕES	13
3.1 - Comando <code>if () {}</code>	13
4 - FUNÇÕES	15
4.1 - Introdução.....	15
5 - VETORES E MATRIZES	18
5.1 - Vetores.....	18
5.2 - Declaração.....	18
5.3 - Inicializando vetores	19
5.4 - Matrizes.....	20
6 - PONTEIROS	21
6.1 - Introdução.....	21
6.2 – O que é um ponteiro ?	22
6.3 – Principais Aplicações de Ponteiros.....	22
6.4 – Armazenando o Endereços das Variáveis	22
6.5 – Acessando o conteúdo de um endereço	23
6.6 – Ponteiros e Argumentos de Funções	23
6.7 – Ponteiros com Vetores.....	24
6.8 – Alocação de Vetores Dinamicamente.....	25
6.9 – Alocação de Matrizes Dinamicamente.....	26

CAPÍTULO I – INTRODUÇÃO A LINGUAGEM C

1- ESTRUTURA BÁSICA DE UM PROGRAMA C

1.1 - Introdução

Vamos acompanhar um exemplo de programa em C.

```
#include <stdio.h>

main()
{
    printf("Programa Inicial");
}
```

Os programas em C sempre iniciam pelo `main()`. De forma geral os programas em C consistem de funções (o termo função será explicado depois), e o `main()` consiste de uma função. Seguindo o nome da função seguem as chaves `{}` que delimitam o início e o fim da função.

Neste exemplo o `printf()` é o único comando da função `main()`. Este comando imprime mensagens na tela padrão e como será visto mais tarde, também imprime o conteúdo de variáveis. Observe que após todo comando dentro de uma função segue um ponto e vírgula `;`.

Os caracteres em branco são invisíveis para o compilador. Assim o programa acima também poderia ser:

```
#include <stdio.h>

main(){ printf("Programa Inicial"); }
```

1.2 - Variáveis

Uma variável é um espaço reservado na memória do computador para armazenar certos tipos de dados.

```
#include <stdio.h>

main()
{
    int num;

    num = 2;
    printf("Numero = %d", num);
}
```

A primeira linha do `main()` declara a variável e o tipo:

Int	num;
Tipo da variável	Nome da variável

Declarando uma variável você informa ao compilador o nome e o tipo da variável. Como regra geral, todas as variáveis em C devem ser declaradas nas primeiras linhas da função. O nome da variável deve iniciar com uma letra.

A segunda linha atribui o valor 2 para a variável `num`:

```
num = 2;
```

Existem sete tipos principais de variáveis:

char	Capaz de conter um caractere (-128 a 127)
short	Inteiro com capacidade reduzida (0 a 15384)
int	Inteiro (-32768 a 32767)
long int	Inteiro estendido (-2 147 483 648 a 2 147 483 647)
float	Ponto flutuante em precisão simples (10e-38 a 10e38)
double	Ponto flutuante em precisão dupla (10e-308 a 10e308)
char int long Unsigned	Retira o sinal e portanto troca a variação de valores, por exemplo: unsigned int varia de 0 a 65 535

Obs: No compilador Visual C++ 4.0, o tipo short tem capacidade: (-32768 a 32767) e o tipo int e long int tem a mesma capacidade: (-2 147 483 648 a 2 147 483 647)

1.3 - A função **printf()**

O printf() é de fato uma função assim como o main(). Os parênteses contêm a sentença a ser apresentada na tela. Esta sentença deve ser limitada por aspas (“ ”), e o conteúdo de variáveis também podem ser exibidas pelo printf conforme o exemplo abaixo:

```
#include <stdio.h>

main()
{
    int a = 1;
    char tipo = 'c';
    float valor = 22.53;

    printf("inteiro= %d   char= %c   float= %f \n",a,tipo,valor);
}
```

O resultado da execução deste programa será:

```
Inteiro= 1   char= c   float= 22.530001
```

O símbolo % dentro de uma sentença do printf indica que desejamos imprimir o conteúdo de alguma variável. Abaixo uma descrição dos principais tipos que o printf reconhece:

%c	Caracter
%s	String
%d	Inteiro
%f	Float (notação decimal)
%e	Float (notação exponencial)
%g	Float
%u	Inteiro sem sinal
%x	Hexadecimal inteiro
%o	Octal inteiro
%ld %lu %lx %lo	Long

Ainda no último exemplo, observe que o último caractere do printf era “\n”. Este caractere de controle indica que queremos inserir uma nova linha. Existem outros caracteres de controle, e todos iniciam com um \ “, conforme a tabela abaixo:

\n	Nova linha
\t	Tabulação
\b	Backspace
\'	Aspas simples
\"	Dupla aspas
\\	Barra

No exemplo abaixo ilustraremos alguns resultados obtidos com o comando printf (Os retângulos abaixo de cada comando printf ilustra o resultado obtido na tela do computador):

```
#include <stdio.h>

main()
{
    int i = 52;
    float x = 28.534;

    printf("i = %2d\n",i);
        i      =      5 2
    printf("i = %4d\n",i);
        i      =          5 2
    printf("x = %8.3f\n",x);
        x      =          2 8 . 5 3 4
    printf("x = %-8.3f\n",x);
        x      =      2 8 . 5 3 4
}
```

1.4 - A função scanf()

Esta função é responsável pela entrada de dados via teclado. Existem várias funções de entrada e saída em C, porém o printf e o scanf são as mais comuns. Examine o exemplo abaixo:

```
#include <stdio.h>

main()
{
    int    i,j;
    float  x,y;
    double a,b;

    scanf("%d",&i);
    scanf("%f",&x);
    scanf("%lf",&a);

    j = i % 2;
    y = x * 2;
    b = a / 2;

    printf("i= %3d  j= %3d \n",i,j);
    printf("x= %.3f  y= %.3f \n",x,y);
    printf("a= %g   b= %g \n",a,b);
}
```

O comando `scanf` interrompe a execução do programa e aguarda o usuário entrar com um valor para a variável `x`. Existem dois campos a serem informados neste comando:

<code>scanf (</code>	<code>"%f"</code>	<code>, &x) ;</code>
	Tipo da variável	Endereço da variável (você deve colocar um "&" na frente da variável)

Para o tipo `double` o prefixo do `scanf` deve ser `lf`, como no exemplo.

1.5 - Operadores

Os operadores aritméticos sobre as variáveis são os seguintes:

<code>+</code>	Soma
<code>-</code>	Subtração
<code>*</code>	Multiplicação
<code>/</code>	Divisão
<code>%</code>	Resto da divisão

A operação de incremento de uma unidade tem também um formato reduzido, ou seja, o comando:

```
i = i + 1;
```

é freqüentemente representado por:

```
i++;
```

Da mesma forma `i = i-1;` pode ser escrito como `i--;`

Como exemplo dos operadores, o programa abaixo calcula as raízes reais de um polinômio de segundo grau:

```
#include <stdio.h>
#include <math.h>

main()
{
    float a,b,c;
    float x1,x2;

    printf("Entre a,b,c:");
    scanf("%f %f %f",&a,&b,&c);

    x1=(-b+sqrt(b*b-4*a*c))/(2*a);
    x2=(-b-sqrt(b*b-4*a*c))/(2*a);

    printf("\n x1 = %f x2 = %f\n",x1,x2);
}
```

1.6 - Comentários

É possível introduzir comentários dentro de um programa C, bastando para isso colocá-los no seguinte formato:

```
/* Comentario */
```

Observe o exemplo abaixo:

```
/* * * * * * * * * * * * * * * * * * * * * * * * * */
/* Soluções reais da equação ax*x + b*x + c = 0 */

#include <stdio.h>
#include <math.h>

main()
{
    /* Definicao das variaveis */

    float a,b,c; /* Coeficientes do polinomio */
    float x1,x2; /* Solucoes desejadas */

    /* Entrada de Dados */

    printf("Entre a,b,c:");
    scanf("%f %f %f",&a,&b,&c);

    /* Calculo das raizes */

    x1=(-b+sqrt(b*b-4*a*c))/(2*a);
    x2=(-b-sqrt(b*b-4*a*c))/(2*a);

    /* Impressao dos dados finais */

    printf("\n x1 = %f x2 = %f\n",x1,x2);
}
```


2 - LAÇOS (LOOPS)

Existem três formatos básicos de laços:

```
1 - do {} while();
2 - while() {}
3 - for( ; ; ) { }
```

2.1 - Comando **for(;;){}**

O for é um comando apropriado quando queremos executar um conjunto de operações um número fixo de vezes. O exemplo abaixo imprime os números de 0 até 9:

```
#include <stdio.h>

main()
{
    int n;

    for(n=0; n<10; n++) {
        printf("n = %d\n",n);
    }
}
```

O resultado será:

```
n = 0
n = 1
n = 2
n = 3
n = 4
n = 5
n = 6
n = 7
n = 8
n = 9
```

O comando for é composto de três argumentos:

for(n=0	;	n<10	;	n++)
	Expressão de inicialização		Expressão de teste		Incremento	

Expressão de inicialização

Inicializa a variável do laço. A inicialização é feita uma única vez quando o laço inicia.

Expressão de teste

Esta expressão testa (a cada vez que o conjunto de comandos no interior do for finaliza), se o laço deve ser encerrado. Enquanto a expressão for verdadeira o laço é repetido. Para realizar teste utilizamos os operadores relacionais. Os principais operadores são:

<	Menor
>	Maior
<=	Menor ou igual
>=	Maior ou igual
==	Igual
!=	Diferente

&&	e
!	negação
	ou

Expressão de incremento

A cada repetição do laço, o terceiro argumento (n++) incrementa a variável n.

Exemplo: Métodos numéricos de integração (ponto a esquerda)

Como aplicação do comando `for` o exemplo abaixo ilustra a implementação do método do ponto a esquerda para avaliar numericamente :

$$\int_0^1 x^2 dx$$

```
/* * * * * * * * * * * * * * * * * * * * * * * * * */
/* Integracao Numerica: Ponto a Esquerda */
/* * * * * * * * * * * * * * * * * * * * * * * * */

#include <stdio.h>

main()
{
    int    i;
    int    n;
    float  w;
    float  dx ;
    float  a,b ;
    float  soma;

    printf("\nEntre o extremo inferior do intervalo: ");
    scanf("%f",&a);
    printf("\nEntre o extremo superior do intervalo: ");
    scanf("%f",&b);
    printf("\nEntre com o numero de particoes: ");
    scanf("%d",&n);

    soma = 0.0;
    dx   = (b-a)/n;
    for(i=0; i< n; i++) {
        w = a + i * dx;
        soma += (w * w) * dx;
        printf("\nSoma parcial = %f ",soma);
    }
    printf("\nIntegral = %15.9f ",soma);
}
```

2.2 - Comando `do {} while();`

Este segundo tipo de laço é adequado para situações onde não sabemos ao certo quantas vezes o laço deve ser repetido. O exemplo abaixo ilustra a utilização do comando:

```
#include <stdio.h>

main()
{
    int n;

    n = 0;
    do {
        printf("n = %d\n",n);
        n++;
    } while (n < 10);
}
```

No exemplo acima o programa entra dentro do laço faz o primeiro printf, incrementa n de uma unidade e só então verifica a condição while (n < 10). Caso a condição seja verdadeira, a execução do programa retorna a primeira linha do laço (printf("n = %d\n", n);) e prossegue até que a condição seja falsa. *O programa sempre entra executando os comando no interior do laço, para somente no final realizar o teste.*

Exemplo: Método de Newton

$$y = x^2 - 2$$

O programa abaixo determina as raízes da função: utilizando o método de Newton, ou seja, dada uma condição inicial x0, e um erro máximo E, a seqüência abaixo pode convergir para uma das raízes:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

```
/* * * * * * * * * * * */
/* Metodo de Newton */

#include <stdio.h>
#include <math.h>

main()
{
    int i;
    double xn, x0, xn_1;
    double erro;

    printf(" Entre com a condicao inicial: ");
    scanf("%lf", &x0);
    printf("\n Erro maximo: ");
    scanf("%lf", &erro);

    xn = x0;
    i = 0;
    do {
        xn_1 = xn;
        xn_1 = xn_1 - (xn_1 * xn_1 - 2) / (2 * xn_1);
        i++;
        printf("\nx[%2d] = %20.17f", i, xn);
    } while (fabs(xn - xn_1) > erro);
    printf ("\n A raiz obtida foi: %20.17f\n", xn);
}
```

2.2 - Comando while() {};

Este último tipo de laço é muito semelhante ao anterior. Partindo do exemplo abaixo vamos observar a única diferença para o laço da seção anterior:

```
#include <stdio.h>

main()
{
    int n;

    n = 0;
    while (n < 10){
        printf("n = %d\n", n);
        n++;
    }
}
```

Neste exemplo, primeiro é feito o teste (`while (n < 10)`) e somente em caso verdadeiro os comando dentro do `while` são executados. Assim se a condição for falsa a primeira vez, em nenhum momento os comandos dentro do laço serão executados. Já no `do{ } while()`; pelo menos a primeira vez os comandos dentro do laço serão executados.

2.4 - Comando `break` e `continue`

Para auxiliar nos laços contamos com dois comandos de interrupção do laço, cada um com objetivos diferentes.

O comando `break`; interrompe o laço (Qualquer dos três formatos apresentados) e o programa continua no primeiro comando após o laço. Exemplo:

```
#include <stdio.h>

main()
{
    int n;

    n = 0;
    while (n < 10){
        printf("n = %d\n",n);
        if (n > 3)
            break;
        n++;
    }
    printf ("Fim do programa \n");
}
```

O resultado deste programa será:

```
n = 0
n = 1
n = 2
n = 3
n = 4
Fim do programa
```

O comando `continue`; transfere a execução do programa para o teste do laço, que pode ou não prosseguir conforme a condição seja verdadeira ou falsa.

3 - DECISÕES

3.1 - Comando `if () {}`

O comando principal de decisão é o `if ()`. Através deste comando o fluxo do programa pode ser desviado para executar ou não um conjunto de comandos. Considere o exemplo abaixo:

```
/* * * * * * * * * * * * * * * * * */
/* Testa se um numero e par ou impar */
/* * * * * * * * * * * * * * * * * */

#include <stdio.h>

main()
{
    int i,n;

    printf("Entre com n = ");
    scanf("%d", &n);

    i = n % 2;

    if (i == 0) {
        printf("\n n e um numero par\n");
    }
    else {
        printf("\n n e um numero impar\n");
    }
}
```

Neste exemplo a variável `i` armazena o resto da divisão de `n` por 2. Caso seja zero, então o programa passa a execução do comando `printf("\n n e um numero par\n")`. Se a condição falha, o comando `else` indica que o programa deve executar o comando `printf("\n n e um numero impar\n")`. Observe que o `else` é um comando opcional. Caso você não o inclua, o programa segue para o próximo comando após o `if`.

Exemplo: Método da Bissecção

O programa abaixo determina as raízes da função: $y = x^2 - 2$ utilizando o método da bissecção. No método da bissecção procuramos uma raiz contida em certo intervalo $[a,b]$ dado. A raiz existe desde que a função seja contínua e o sinal da função troque de um extremo para outro (ou seja $f(a) * f(b) < 0$).

```
/* * * * * * * * * * * * * * * * * */
/*           Metodo da Bissecacao           */
/* * * * * * * * * * * * * * * * * */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

main()
{
    float a,b,c;
    float erro;
    float fa,fb,fc;

    printf("\nEntre com a = ");
```

```

scanf("%f",&a);
printf("\nEntre com b = ");
scanf("%f",&b);

printf("\nEntre com o erro = ");
scanf("%f",&erro);

fa = a*a - 2;
fb = b*b - 2;

if ((fa * fb) > 0) {
    printf("\nCondicao inicial nao contem raiz !\n");
    exit(0);
}

while(fabs(a-b) > erro) {
    c = (a+b)/2.0;
    fc = c*c - 2.0;

    if (fa * fc < 0) {
        b = c;
    }
    else {
        if (fb * fc < 0)
            a = c;
        else
            break;
    }
    printf("\n Raiz parcial = %f ",c);
}
printf("\n\n Raiz obtida = %f \n",c);
}

```

4 - FUNÇÕES

4.1 - Introdução

As funções cumprem como primeiro papel, evitar repetições desnecessárias de código. No exemplo anterior precisamos calcular o valor $y=x*x-2$ em diversas partes do programa. Se desejássemos trocar a função, seria necessário alterar várias partes do código. Para evitar isso, utilizaremos uma função como no exemplo abaixo:

```
/* * * * * * * * * * * * * * * * * * * * */
/*      Metodo da Bisssecao      */
/* * * * * * * * * * * * * * * * * * * * */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

float f(float x)
{
    return(x*x-2);
}

void main()
{
    int    i = 0;
    float  a,b,c;
    float  erro;
    float  fa,fb,fc;

    printf("Entre com os extremos do intervalo [a, b]: ");
    scanf("%f,%f",&a,&b);

    printf("\nErro: ");
    scanf("%f",&erro);

    fa = f(a);
    fb = f(b);

    if ((fa * fb) > 0) {
        printf("Condicao inicial nao contem raiz !\n");
        exit(0);
    }

    while(fabs(a-b) > erro) {
        i++;
        c = (a+b)/2.0;
        fc = f(c);

        if (fa * fc < 0)
            b = c;
        else
            if (fb * fc < 0)
                a = c;
            else
                break;
        printf("Raiz parcial no passo %d = %f \n",i,c);
    }
    printf("\nRaiz obtida = %f \n",c);
}
```

Vamos examinar alguns detalhes da função introduzida:

float	f	(float x)
Define o tipo que será retornado	Nome da função	Argumentos de entrada

```
{
    float y;

    y = x * x - 2;
```

return(y);
Valor a ser retornado

```
}
```

Uma observação importante é que as variáveis dentro da função não são conhecidas fora da função e vice-versa.

Exemplo: Métodos de Integração Numérica (Ponto a esquerda, Trapézio e Simpson)

Para ilustrar o conceito de funções, o exemplo seguinte calcula a integral numérica de uma função definida na função `float f(float x)`.

```
/* * * * * * * * * * * * * * * * * */
/*  Metodos de Integracao Numerica  */
/* * * * * * * * * * * * * * * * * */

#include <stdio.h>
#include <math.h>
#include <conio.h>

float f(float x)
{
    float y;

    y = 1/(1+x*x);
    return(y);
}

float Integra_Ponto_a_esquerda(float a,float b,int n)
{
    int i;
    float soma;
    float dx;

    soma = 0.0;
    dx = (b-a)/n;
    for(i=1; i<= n; i++) {
        soma = soma + f(a + (i-1)*dx) * dx;
        printf("\nSoma parcial = %f ",soma);
    }
    return(soma);
}

float Integral_Trapezio(float a,float b,int n)
{
    int i;
    float soma;
    float dx;
```



```

    dx  = (b-a)/n;
    soma = f(a);
    for(i=1; i < n; i++) {
        soma = soma + 2*f(a + i*dx);
        printf("\nSoma parcial = %f ",soma);
    }
    soma = dx/2 * (soma + f(b));
    return(soma);
}

float Integral_Simpson(float a,float b,int n)
{
    int i;
    float soma;
    float dx;

    dx  = (b-a)/n;
    soma = f(a);
    for(i=1; i < n; i++) {
        if ((i%2) == 1)
            soma = soma + 4*f(a + i*dx);
        else
            soma = soma + 2*f(a + i*dx);
        printf("\nSoma parcial = %f ",soma);
    }
    soma = dx/3 * (soma + f(b));
    return(soma);
}

main()
{
    int i;
    int n;
    char c;
    float a,b;
    float soma;

    do {
        printf("Selecione um do metodos de integracao \n");
        printf("1- Ponto a esquerda \n");
        printf("2- Trapezio \n");
        printf("3- Simpson \n");
        printf("Opcao: ");
        scanf("%d",&i);
        printf("\nEntre o extremo inferior do intervalo: ");
        scanf("%f",&a);
        printf("\nEntre o extremo superior do intervalo: ");
        scanf("%f",&b);
        printf("\nEntre com o numero de particoes: ");
        scanf("%d",&n);
        switch(i)
        {
            case 1:
                soma = Integra_Ponto_a_esquerda(a,b,n);
                break;
            case 2:
                soma = Integral_Trapezio(a,b,n);
                break;
            case 3:
                soma = Integral_Simpson(a,b,n);
                break;
        }
        printf("\nIntegral = %15.9f ",soma);
        printf("\n Continua (s/n) ");
        c = getche();
    } while((c != 'n') && (c != 'N'));
}

```

5 - VETORES E MATRIZES

5.1 - Vetores

Quando você deseja representar uma coleção de dados semelhantes, pode ser muito inconveniente utilizar um nome de variável diferente para cada dado.

Para ilustrar vamos considerar o seguinte exemplo: Montar um programa que armazena as notas de 15 alunos e calcula a média obtida pela turma. As notas serão armazenadas em uma variável do tipo `float`, porém ao invés de criarmos 15 variáveis, utilizamos uma variável do tipo vetor, definida como abaixo:

```
float notas[15];
```

o programa completo seria:

```
#include <stdio.h>

main()
{
    int    i        ;
    float  media     ;
    float  soma      ;
    float  notas[15];

    for(i=0;i < 15;i++) {
        printf("\n Aluno %2d ",i+1);
        scanf("%f",&notas[i]);
    }

    for(i=0;i<15;i++)
        soma = soma + notas[i];

    media = soma / 15;

    printf("\n A media final foi: %f",media);
}
```

5.2 - Declaração

Um vetor é uma coleção de variáveis de certo tipo, alocadas sequencialmente na memória. Para o compilador C um declaração de variável vetor do tipo:

```
int n[5];
```

reserva o espaço de 5 variáveis do tipo inteira, onde cada variável pode ser referenciada conforme abaixo:

n[0]	n[1]	n[2]	n[3]	n[4]

IMPORTANTE: Observe que a declaração anterior cria cinco variáveis, porém o primeiro elemento é `n[0]`. A declaração de vetor inicia com o índice 0 e finaliza no índice 4.

Se você quer atribuir um valor a um dos componentes do vetor basta referencia-lo por exemplo:

```
n[3] = 29;
```

resulta em:

			29	
n[0]	n[1]	n[2]	n[3]	n[4]

5.3 - Inicializando vetores

Assim como é possível inicializar uma variável simples na mesma linha da declaração, o mesmo pode ser feito para vetores. Observe o exemplo abaixo:

```
int n[5] = {23, 3, -7, 288, 14};
```

Exemplo: Método de Ordenação

Como exemplo vamos apresentar um programa que ordena uma sequência de 10 números reais.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Metodo da Bolha (ordenacao de um vetor) */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */

#define MAX 10
#include <stdio.h>

main()
{
    int    i    ;
    int    flag;
    float  swap;
    float  n[MAX];

    /* Entrada de Dados */

    printf("Entre com os numeros para ordenacao \n");
    for(i=0; i<MAX; i++)
    {
        printf("\n numero %2d: ", i);
        scanf("%f", &(n[i]));
    }

    /* Ordena a sequencia de numeros */

    flag = 1;
    while(flag == 1) {
        flag = 0;
        for(i=0; i<(MAX-1); i++)
        {
            if (n[i] > n[i+1])
            {
                swap  = n[i]    ;
                n[i]   = n[i+1];
                n[i+1] = swap    ;
                flag   = 1      ;
            }
        }
    }

    /* Imprime a sequencia de numeros ordenada */

    printf("\nSequencia ordenada : ");
    for(i=0; i<MAX; i++)
        printf("\n %10.5f ", n[i]);
    printf("\n");
}
```

5.4 - Matrizes

Para representar uma matriz 3x4 (3 linha e 4 colunas) de números reais utilizamos a seguinte declaração:

```
float A[3][4];
```

Assim fica reservado um espaço de memória conforme a figura abaixo:

A[0][0]	A[0][1]	A[0][2]	A[0][3]
A[1][0]	A[1][1]	A[1][2]	A[1][3]
A[2][0]	A[2][1]	A[2][2]	A[2][3]

Exemplo: Produto de uma matriz por um vetor

Vamos montar um programa que multiplica um vetor por uma matriz.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Multiplicacao de um vetor por uma matriz */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <stdio.h>

main()
{
    int    i,j;
    float A[3][3] = {{1.0, 1.5, 2.1}, {3.4, 2.2, 9.1}, {-1.2, -3.4, 0.9}};
    float v[3]    = {2.0, 1.0, 0.5};
    float p[3];

    for(i=0;i<3;i++) {
        p[i] = 0;
        for(j=0;j<3;j++) {
            p[i] += A[i][j] * v[j];
        }
    }

    for(i=0;i<3;i++) {
        printf("\n[");
        for(j=0;j<3;j++) {
            printf("%5.3f ",A[i][j]);
        }
        printf(" ] ");
        printf(" [ %5.3f ]",v[i]);
    }

    for(i=0;i<3;i++)
        printf("\n p[%1d] = %10.4f",i,p[i]);
}
```

6 - PONTEIROS

6.1 - Introdução

Considere o seguinte programa:

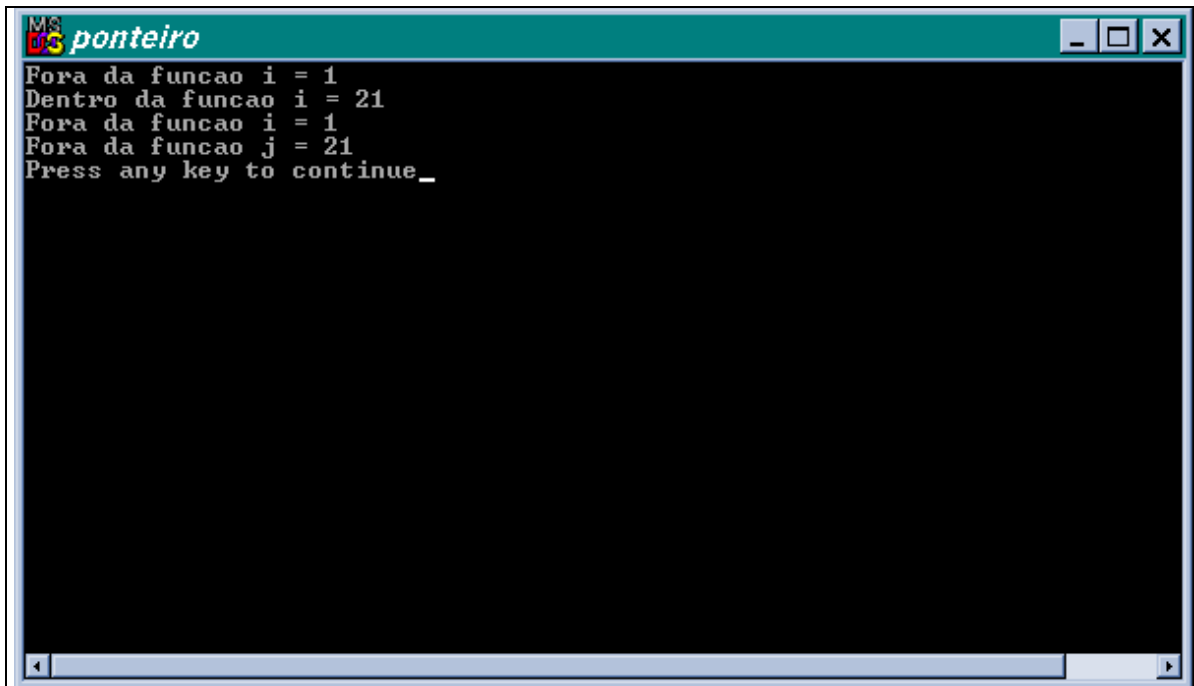
```
#include <stdio.h>

int teste(int i)
{
    i = i + 20;
    printf("Dentro da funcao i = %d \n",i);
    return(i);
}

void main()
{
    int i,j;

    i = 1;
    printf("Fora da funcao i = %d \n",i);
    j = teste(i);
    printf("Fora da funcao i = %d \n",i);
    printf("Fora da funcao j = %d \n",j);
}
```

O resultado será:



Observe que o valor da variável `i` não tem seu conteúdo alterado pela função. Isto ocorre porque quando uma função é chamada durante o programa, todas as variáveis presentes na função são criadas neste. Quando a função finaliza, todas as variáveis da função são apagadas da memória. Muitas vezes gostaríamos que os argumentos de entrada da função pudessem ter seus valores alterados quando a função finalizasse. Esta será uma das aplicações do conceito de ponteiros.

6.2 – O que é um ponteiro ?

Um ponteiro é uma variável que contém o endereço de memória de outra variável. Todas as variáveis são alocadas em algum espaço de memória do computador. O ponteiro fornece um mecanismo para obter e armazenar este endereço de memória. Considere o exemplo:

```
#include <stdio.h>

void main()
{
    int i;

    printf("Endereco de i = %x\n",&i);
}
```

O endereço da variável é obtido utilizando-se o operador unário & na frente da variável. Assim &i fornece o endereço da variável i.

6.3 – Principais Aplicações de Ponteiros

Algumas situações em que os ponteiros são úteis:

1. Quando em uma função desejamos retornar mais de um valor.
2. Para passar vetores e matrizes de forma mais conveniente como argumentos de funções.
3. Para manipular vetores e matrizes de forma mais eficiente.
4. Para manipular estruturas de dados mais complexas, tais como listas e árvores.
5. Na utilização das funções `calloc()` e `malloc()` responsáveis pela alocação de memória dinâmica.

6.4 – Armazenando o Endereços das Variáveis

Para armazenar o endereço de uma variável (por exemplo &i) em outra variável, é necessário criar um tipo especial de variável denominada apontador. Exemplo:

```
#include <stdio.h>

void main()
{
    int i ;
    int *pi;

    pi = &i;
    printf("Endereco de i = %x ou %x\n",&i,pi);
}
```

A variável `pi` é uma variável do tipo ponteiro para inteiro (ou seja, ela recebe o endereço de uma variável do tipo inteiro). Para informar que esta variável é do tipo apontador colocamos um asterisco (*) na frente da variável no momento da sua declaração:

int	*	pi;
Tipo de ponteiro	Indica ponteiro	Nome da variável

6.5 – Acessando o conteúdo de um endereço

Considere o exemplo anterior em que `pi = &i`. Além do endereço de `i`, (já armazenado em `pi`) podemos também acessar o conteúdo armazenado no endereço de memória. Isto equivale a obter o valor da variável `i`. Observe o exemplo:

```
#include <stdio.h>

void main()
{
    int i ;
    int j ;
    int *pi;

    pi = &i;

    i = 25;
    j = *pi + 8; /* equivalente a j = i + 8 */

    printf("Endereco de i = %x \n",pi);
    printf("j = %d \n",j);
}
```

O operador unário `*` trata seu operando como um endereço e acessa este endereço para buscar o conteúdo da variável. Observe que o `*` tem dupla função:

1. Em uma declaração de variável, indica que a variável é do tipo ponteiro.
2. Durante uma atribuição, acessa o conteúdo do endereço armazenado pela variável ponteiro.

No nosso exemplo, para alterar o valor da variável `i`, temos duas alternativas (totalmente equivalentes) :

```
i = 5;
*pi = 5;
```

6.6 – Ponteiros e Argumentos de Funções

Como sabemos uma função não consegue alterar diretamente uma variável utilizada como argumento. Com o auxílio dos ponteiros podemos encontrar uma alternativa para esta situação. No exemplo do programa de ordenação, foi preciso permutar dois elementos do vetor que estavam fora de ordem. Não seria possível permutar dois valores através de uma função como abaixo:

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Este programa NÃO consegue permutar os valores */

#include <stdio.h>

void troca(float x, float y)
{
    float auxiliar;

    auxiliar = x;
    x = y;
    y = auxiliar;
}

main()
{
    float x = 1.2;
    float y = 56.89;

    troca(x,y);
    printf("x = %6.2f e y = %6.2f\n",x,y);
}
```

O resultado deste programa seria:

```
x = 1.2 e y = 56.89
```

Utilizando ponteiros existe uma maneira de obter o efeito desejado:

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Este programa permutar os valores das variaveis */

#include <stdio.h>

void troca(float *x, float *y)
{
    float auxiliar;

    auxiliar = *x;
    *x = *y;
    *y = auxiliar;
}

main()
{
    float x = 1.2;
    float y = 56.89;

    troca(&x,&y);

    printf("x = %6.2f e y = %6.2f\n",x,y);
}
```

O uso comum de argumentos do tipo ponteiro ocorre em funções que devem retornar mais de um valor. No exemplo acima, a função retornou dois valores.

6.7 – Ponteiros com Vetores

Na linguagem C, o relacionamento de ponteiros com vetores e matrizes é tão direto que daqui para frente sempre trataremos vetores e matrizes utilizando ponteiros. Qualquer operação que possa ser feita com índices de um vetor pode ser feita através de ponteiros. Vamos acompanhar através do exemplo de ordenação:

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Ordenacao pelo metodo da bolha */

#define MAX 10
#include <stdio.h>

void troca(float *x, float *y)
{
    float auxiliar;

    auxiliar = *x;
    *x = *y;
    *y = auxiliar;
}

void ordena(float *px)
{
    int i;
    int flag;

    /* Ordena a sequencia de numeros */

    flag = 1;
    while(flag == 1) {
```



```

        flag = 0;
        for(i=0;i<(MAX-1);i++)
        {
            if (px[i] > px[i+1])
            {
                troca(&(px[i]),&(px[i+1]));
                flag = 1;
            }
        }
    }
}

main()
{
    int    i    ;
    float  n[MAX];

    /* Entrada de Dados */

    printf("Entre com os numeros para ordenacao \n");
    for(i=0;i<MAX;i++)
    {
        printf("\n numero %2d: ",i);
        scanf("%f",&(n[i]));
    }

    ordena(n);

    /* Imprime a sequencia de numeros ordenada */

    printf("\nSequencia ordenada : ");
    for(i=0; i<MAX; i++)
        printf("\n %10.5f ",n[i]);
    printf("\n");
}

```

Observe que quando chamamos a função `ordena(n,MAX)`, utilizamos como argumento `n` referência a nenhum índice. A função `ordena` é declarada como:

```
void ordena(float *px, int limite)
```

Como o C sabe que estamos nos referenciando a um vetor e não a uma simples variável? Na verdade `px` é uma variável ponteiro para um float. Ela recebe o endereço do primeiro elemento do vetor. Porém, os vetores são alocados sequencialmente na memória do computador, de forma que a referência `px[5]`, acessa o quinto conteúdo em sequência na memória.

Quando o nome de um vetor é passado para uma função, o que é passado é a posição do início do vetor.

6.8 – Alocação de Vetores Dinamicamente

Uma das restrições do nosso programa de ordenação é que o usuário não pode definir a princípio quantos elementos ele pretende ordenar. Isto ocorre porque precisamos informar na declaração do vetor qual será a sua dimensão. Vamos retirar essa restrição. Para isso utilizaremos um novo comando: `malloc()`. Vamos utilizar um exemplo:

```

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

void main()
{
    int    i;
    int    n;
    float  *v;

```

```

printf("Entre com a dimensao do vetor desejada = ");
scanf("%d",&n);

v = (float *) malloc(n * sizeof(float));

for(i=0;i<n;i++)
    scanf("%f",&(v[i]));

for(i=0;i<n;i++)
    printf("v[%2d]=%f\n",i,v[i]);

free(v);
}

```

Para efetuar a alocação dinâmica observe as etapas do programa:

1. O vetor que será alocado foi declarado como um ponteiro: `float *v;`
2. O comando `malloc(n * sizeof(float))` reserva n espaços de memória, cada um do tamanho de um float.
3. A função `malloc` retorna o endereço do primeiro elemento do vetor: `(float *) malloc()`
4. A função `free`, libera o espaço de memória reservado para o vetor.

6.9 – Alocação de Matrizes Dinamicamente

Para alocar matrizes observe o exemplo:

```

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

float **matalloc(int n,int m)
{
    int    j;
    float  **A;

    A = (float **)malloc(n * sizeof(float *));
    for(j=0;j<n;j++)
        A[j] = (float *) malloc(m * sizeof(float));
    return(A);
}

void matfree(float **A,int n,int m)
{
    int i;
    for(i=0;i<n;i++)
        free(A[i]);
    free(A);
}

void main()
{
    int    i;
    int    j;
    int    n;
    int    m;
    float  **A;

    printf("Entre com a dimensão da matriz desejada = ");
    scanf("%d, %d",&n,&m);

    A = matalloc(n,m);
}

```

```

    for (i=0; i<n; i++)
        for (j=0; j<m; j++)
            scanf ("%f", &(A[i] [j]));

    for (i=0; i<n; i++) {
        printf ("\n");
        for (j=0; j<m; j++)
            printf ("A[%2d] [%2d] =%f\n", i, j, A[i] [j]);
    }
    matfree (A, n, m);
}

```

Exemplo: Método LU para resolução de Sistemas Lineares

Um dos métodos clássicos para se resolver um sistema linear da forma $Ax=b$, é a decomposição LU. De forma reduzida temos três etapas:

1. Decompor $A = LU$
2. Resolver $Lc = b$
3. Resolver $Ux = c$

Vamos descrever este método partindo de um exemplo. Considere o sistema linear abaixo:

$$\begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix}$$

Determinando a matriz L e U

Nosso objetivo inicial é aplicar um conjunto de operações elementares sobre a matriz dos coeficientes de tal forma que ao final do processo obtemos $A = LU$, ou seja, a matriz A pode ser escrita como o produto de duas matrizes onde:

L : Matriz triangular inferior, cuja diagonal contém apenas o valor 1.

U : Matriz triangular superior.

Para obter estas matrizes existem três tipos de operações elementares:

- 1- Permutação de linhas ($L_i \leftrightarrow L_j$) (*Não é necessário implementar, para o primeiro trabalho*).
- 2- Multiplicação de uma linha por um escalar K ($L_i = K L_i$). (*Não é necessário implementar, para o primeiro trabalho*).
- 3- Substituição da i -ésima linha pela i -ésima linha acrescida de K vezes a j -ésima linha ($L_i = L_i + K L_j$)

Vamos iniciar determinando a matriz U . Para isso nosso objetivo é transformar a matriz dada em uma matriz triangular superior. Começamos na primeira coluna, eliminando todos os elementos abaixo do 2:

$$\begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix} \xrightarrow{L_2 \rightarrow L_2 - 2L_1} \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ -2 & 7 & 2 \end{bmatrix}$$

E para a ultima linha obtemos:

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ -2 & 7 & 2 \end{bmatrix} L_3 \rightarrow L_3 + L_1 \Rightarrow \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 8 & 3 \end{bmatrix}$$

Passamos agora para a segunda coluna e eliminaremos os elementos abaixo do -8 (neste exemplo o 8):

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 8 & 3 \end{bmatrix} L_3 \rightarrow L_3 + L_2 \Rightarrow \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

Obtemos então uma matriz triangular superior. Esta matriz é matriz U desejada.

Para obter a matriz L, observamos que cada uma das operações elementares aplicadas pode ser representada por uma matriz. Vamos acompanhar pelo exemplo.

A primeira operação elementar foi :

$$L_2 \rightarrow L_2 - 2L_1$$

Esta operação elementar é representada pela matriz:

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Tal que:

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ -2 & 7 & 2 \end{bmatrix}$$

De forma geral todas as operações elementares são representadas por matrizes:

1. *Permutação de linhas ($L_i \leftrightarrow L_j$): Partindo da matriz identidade (de mesma dimensão que a matriz A) troque a i-ésima linha pela j-ésima linha.*

Ex: $L_2 \leftrightarrow L_3$, basta trocar as linhas 2 e 3 da matriz identidade para obter a matriz:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

2. *Multiplicação de uma linha por um escalar K ($L_i = K L_i$): Partindo da matriz identidade troque o i-ésimo elemento da diagonal pelo valor K.*

Ex: $L_2 = 5 L_2$, basta trocar na segunda linha da identidade a diagonal para 5:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3. Substituição da i -ésima linha pela i -ésima linha acrescida de K vezes a j -ésima linha ($L_i = L_i + K L_j$):
 Partindo da matriz identidade, coloque o valor K na i -ésima linha e j -ésima coluna.
 Ex: $L_2 = L_2 + 4 L_3$. Partindo da matriz identidade, coloque na segunda linha e na terceira coluna o valor 4:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix}$$

Voltando ao nosso exemplo, temos mais duas matrizes elementares:

$$L_3 \rightarrow L_3 + L_1 \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

$$L_3 \rightarrow L_3 + L_2 \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Podemos então representar as operações elementares sobre a matriz A por:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}}_{L^{-1}} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix}}_A = \underbrace{\begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{bmatrix}}_U$$

Se multiplicarmos pela inversa das matrizes elementares obtemos:

$$\begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

Multiplicando as matrizes elementares obtemos a decomposição LU:

$$\underbrace{\begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix}}_A = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{bmatrix}}_L \underbrace{\begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{bmatrix}}_U$$

Não é necessário fazer as contas para obter a matriz L . Observe que a matriz L só possui elementos abaixo da diagonal. Estes elementos são obtidos simplesmente invertendo o sinal da constante K obtida em cada operação elementar, e posicionando na matriz conforme a linha e coluna adequada.

Resolvendo $Lc = b$

Uma vez obtida a matriz L, resolvemos $Lc = b$:

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \\ 9 \end{bmatrix}$$

Assim:

$$\begin{aligned} c_0 &= 5 \\ c_1 &= -2 - 10 = -12 \\ c_2 &= 9 + 5 - 12 = 2 \end{aligned} \quad \text{e portanto:} \quad \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 5 \\ -12 \\ 2 \end{bmatrix}$$

Resolvendo $Ux = c$

Resolvemos agora o sistema $Ux = b$

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ -12 \\ 2 \end{bmatrix}$$

E obtemos:

$$x_2 = 2$$

$$x_1 = \frac{-12 + 4}{-8} = 1$$

$$x_0 = \frac{5 - 1 - 2}{2} = 1$$

A solução do nosso sistema será:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

<u>CAPÍTULO II – VISUALIZAÇÃO E APLICAÇÕES GRÁFICAS 2D</u>	3
1- PONTOS E RETAS NO OPENGL	3
1.1 – A Tela do Computador	3
1.2 – Cores	3
1.3 – Introdução ao OpenGL	4
1.4 – Exemplo: Plotar um ponto na tela utilizando as bibliotecas do OpenGL	4
1.5 – Comando: gluOrtho2D	6
1.6 – Exemplo: Plotar uma reta unindo dois pontos	7
1.6.1 – Algoritmo ingênuo	7
1.6.2 – Retas no Opengl	8
1.7 – Exemplo: Plotar o gráfico de uma função	9
2 – TECLADO E MOUSE (Callbacks)	12
2.1 – Introdução	12
2.2 – Teclado	12
2.3 – Exemplo: Utilização do teclado no programa funções	12
2.3.1 – Módulo <code>funcao011.h</code>	12
2.3.2 – Módulo <code>funcao010.cpp</code>	13
2.3.3 – Módulo <code>funcao011.cpp</code>	13
2.4 – Mouse	15
2.4.1- Interrupções a partir do mouse	15
2.4.2- Aplicações: Realizando o “zoom” do gráfico	17
3 – CURVAS PARAMÉTRICAS	23
3.1 – Introdução	23
3.2 – Exemplo: Visualização de Curvas Paramétricas	23
3.2.1 – Módulo <code>curvas.cpp</code>	23
3.2.2 – Módulo <code>plota.cpp</code>	25
3.2.3 – Módulo <code>dominio.cpp</code>	26
3.2.4 – Exercício	27
3.3 – Curvas na forma Polar	28
3.4 – Exemplo: Visualização de Curvas Polares	28
3.5 – Exercícios	29
4 – RETAS E POLÍGONOS NO OPENGL	30
4.1 – Retas e Polígonos	30
4.2 – Exemplo: Visualização dos Métodos Numéricos de Integração	31
4.2.1 - Módulo <code>Intvis01.c</code>	31
4.2.2 - Módulo <code>Intvis02.cpp</code>	33
4.2.3 - Módulo <code>Intvis03.cpp</code>	34
5 – CURVAS IMPLÍCITAS	36
5.1 – Introdução	36
5.2 – Visualização de Curvas Implícitas	36
5.3 – Programa Curva Implícita	37
5.3.1 – Módulo <code>main.cpp</code>	37
5.3.2 – Módulo <code>graphics.cpp</code>	38

5.3.3 – Módulo dominio.cpp.....	41
5.4 –Exercício	42
6 – PROCESSAMENTO DE IMAGEM.....	42
6.1 – Introdução	42
7 – FRACTAIS	46
7.1 – Conjuntos auto semelhantes	46
7.2 – Dimensão Hausdorff e o conceito de fractal.....	47
7.3 – Exemplos de fractais.....	47
7.3.1- Triângulo de Sierpinski	48
7.3.2- Triângulo de Sierpinski utilizando Monte Carlo.....	52
7.3.3- “Fern” utilizando Monte Carlo.....	54
7.3.4- Curva de Koch.....	56

CAPÍTULO II – VISUALIZAÇÃO E APLICAÇÕES GRÁFICAS 2D

1- PONTOS E RETAS NO OPENGL

1.1 – A Tela do Computador

A tela do computador pode ser considerada uma matriz de células discretas (Pixels), cada qual pode estar acesa ou apagada.

0,1	1,1	2,1	...	
0,0	1,0	2,0	...	

A definição da tela varia conforme o monitor e a placa gráfica. As definições básicas encontradas na maioria dos monitores são:

640 x 480
800 x 600
1024 x 768
1280 x 1024

1.2 – Cores

A cada pixel associamos uma cor. Para obter uma cor, o monitor envia certa combinação de vermelho, verde e azul (RGB). O número de cores possíveis varia conforme o hardware. Cada pixel tem uma mesma quantidade de memória para armazenar suas cores. O buffer de cores (Color Buffer) é uma porção da memória reservada para armazenar as cores em cada pixel. O tamanho deste buffer é usualmente medido em bits. Um buffer de 8 bits pode exibir 256 cores diferentes simultaneamente. Conforme a capacidade da placa gráfica podemos ter:

8 bits –	256 cores
(High Color) 16 bits –	65.536 cores
(True Color) 24 bits –	16.777.216 cores
(True Color) 32 bits –	4.294.967.296 cores

Existem duas formas básica de acessar as cores no OpenGL: RGB e Modo Indexado. Trabalharemos sempre em formato RGB. No formato RGB você deve informar as intensidades de Vermelho, Verde e Azul desejadas. Estas intensidades devem variar entre 0.0 a 1.0. A tabela abaixo mostra como obter as cores básicas:

Cores	R	G	B
Vermelho	1.0	0.0	0.0
Verde	0.0	1.0	0.0
Azul	0.0	0.0	1.0
Amarelo	1.0	1.0	0.0
Cyan	0.0	1.0	1.0
Magenta	1.0	0.0	1.0
Branco	1.0	1.0	1.0
Preto	0.0	0.0	0.0

1.3 – Introdução ao OpenGL

O sistema gráfico OpenGL (GL significa Graphics Library) é uma biblioteca (de aproximadamente 350 comandos) para aplicações gráficas. O OpenGL foi desenvolvido pela Silicon Graphics (SGI) voltado para aplicações de computação gráfica 3D, embora possa ser usado também em 2D. As rotinas permitem gerar primitivas (pontos, linhas, polígonos, etc) e utilizar recursos de iluminação 3D.

O OpenGL é independente do sistema de janelas, ou seja, suas funções não especificam como manipular janelas. Isto permite que o OpenGL possa ser implementado para diferentes sistemas: X Window System (Unix), Windows 95 e NT, OS/2, Macintosh, etc.

1.4 – Exemplo: Plotar um ponto na tela utilizando as bibliotecas do OpenGL

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

void redesenha()
{
    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);
    glBegin(GL_POINTS);
        glVertex2f(200.0,200.0);
    glEnd();
    glFlush();
}

void main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(400,400); /* Antes do glutCreateWindow */
    glutInitWindowPosition(1,1);
    glutCreateWindow("Ponto");
    gluOrtho2D(0,399,399,0); /* Apos CreateWindow */
    glutDisplayFunc(redesenha); /* Esta funcao e necessaria, caso
                                contrario o opengl nao consegue
                                criar a janela */

    glutMainLoop();
}
```

Vamos comentar comando a comando:

`glutInit(&argc,argv);`
É utilizado para iniciar a biblioteca GLUT.

`glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);`
Quando uma janela é criada, seu tipo é determinado pelo Display Mode. O tipo da janela inclui um conjunto de características desejadas. Neste caso temos três:

GLUT_SINGLE: Buffer simples

GLUT_RGBA...: Utilizaremos o modo RGBA.

GLUT_DEPTH.: Buffer de profundidade (utilizado em remoção de superfícies escondidas).

`glutInitWindowSize(400,400);`
Indica o tamanho da janela a ser aberta (em pixels).

`glutInitWindowPosition(1,1);`
Indica a posição inicial da janela.

`glutCreateWindow("Ponto");`

Cria uma janela para o OpenGL denominada Ponto

```
gluOrtho2D(0,399,399,0);
```

Este comando estabelece a escala da tela. Sua sintaxe é:

```
gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);
```

```
glutDisplayFunc(redesenha);
```

Este comando registra que a função `void redesenha()` será a rotina a ser chamada sempre que a janela necessita ser redesenhada.

```
glutMainLoop();
```

Inicia o gerenciamento da janela e mantém o programa em loop, aguardando por eventos.

Quando a função `redesenha` é chamada temos o seguinte resultado:

```
glClearColor(0.0,0.0,0.0,0.0);
```

Indica cor para ser utilizada no fundo da tela.

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Pinta os buffers indicados com a cor do `glClearColor()`

```
glColor3f(1.0,0.0,0.0);
```

Define o vermelho como cor atual.

```
glBegin(GL_POINTS);
```

```
    glVertex2f(200.0,200.0);
```

```
glEnd();
```

Plota um ponto na posição (200,200) na tela.

```
glFlush();
```

Imprime o conteúdo do buffer na tela.

Exercícios:

- 1) Comente as seguintes linhas na função `redesenha` e veja o resultado:

```
    glClearColor(0.0,0.0,0.0,0.0);
```

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

- 2) Acrescente um contador para verificar quantas vezes a função `redesenha` é chamada.

```
void redesenha()
{
    static int i = 0;

    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);
    glBegin(GL_POINTS);
        glVertex2f(200.0,200.0);
    glEnd();
    glFlush();
    printf(" %d ",i++);
}
```

- 3) Comente a seguinte linha do código e veja o resultado.

```
    glutDisplayFunc(redesenha);
```

1.5 – Comando: `gluOrtho2D`

Na maioria de nossas aplicações desejamos nos referenciar a um ponto na tela, não por coordenadas correspondendo as dimensões informadas no comando `glutInitWindowSize()`, mas sim levando-se em conta o domínio de visualização relacionado ao problema. Para isso, o comando `gluOrtho2D()` realiza a mudança para o sistema de coordenadas desejado.

Esta tarefa é realizada fazendo a correspondência entre os intervalos em questão:

$$[X_{\min}, X_{\max}] \rightarrow [0, DIM_x - 1]$$

$$[Y_{\min}, Y_{\max}] \rightarrow [0, DIM_y - 1]$$

Assim:

$$x = X_{\min} + t(X_{\max} - X_{\min}) \quad \text{e segue que } P_x = \frac{x - X_{\min}}{X_{\max} - X_{\min}} (DIM_x - 1)$$
$$P_x = 0 + t(DIM_x - 1)$$

$$\text{e identicamente para a coordenada y: } P_y = \frac{y - Y_{\min}}{Y_{\max} - Y_{\min}} (DIM_y - 1).$$

O programa abaixo ilustra o efeito do `gluOrtho2D`.

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

#define DIMX 401
#define DIMY 601

float xmin = -1;
float xmax = 1;
float ymin = -1;
float ymax = 1;

void converte(float L, float R, float T, float B, float *x, float *y)
{
    /* Faz a funcao do glortho */
    *x = ((*x - L)/(R-L)*(DIMX-1));
    *y = ((*y - B)/(T-B)*(DIMY-1));
}

void display()
{
    float x = 0.5; /* Ponto que gostaríamos de converter */
    float y = 0.5;

    converte(xmin, xmax, ymax, ymin, &x, &y);
    printf("x=%f y=%f\n", x, y);

    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);
    glBegin(GL_POINTS);
        glVertex2f(x, y);
    glEnd();
    glFlush();
    glutSwapBuffers();
}

void main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(DIMX, DIMY);
```

```

    glutInitWindowPosition(50,50);
    glutCreateWindow("Ortho");
    gluOrtho2D(0,DIMX-1,DIMY-1,0);/* Matriz de pontos DIMx x DIMy */
    glutDisplayFunc(display);
    glutMainLoop();
}

```

1.6 – Exemplo: Plotar uma reta unindo dois pontos

Como exemplo vamos desenhar uma reta (não vertical) unindo dois pontos (x0,y0) e (x1,y1).
A equação da reta que passa por dois pontos é:

$$y = \frac{y1 - y0}{x1 - x0}(x - x0) + y0$$

1.6.1 – Algoritmo ingênuo

A primeira idéia de como resolver este problema é proposto pelo programa abaixo. Este é um exemplo ingênuo de como desenhar a reta que passa por dois pontos dados. Vamos discutir a seguir os principais problemas deste programa e as possíveis soluções.

```

/* ----- */
/* Exemplo ingênuo de como plotar a reta definida por dois pontos */
/* ----- */

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

int    pontos;
float  x0,y0,x1,y1;

float Reta_dois_pontos(float x)
{
    float y;

    y = (y1-y0)/(x1-x0)*(x-x0) + y0;

    return(y);
}

void display()
{
    int    i    ;
    float  x,y;

    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);
    for (i=0;i<pontos;i++)
    {
        x = x0 + i * (x1 - x0)/pontos;
        y = Reta_dois_pontos(x);
        glBegin(GL_POINTS);
            glVertex2f(x,y);
        glEnd();
    }
    glFlush();
    glutSwapBuffers();
}

```

```

void main(int argc, char **argv)
{
    printf("x0 = ");
    scanf("%f",&x0);
    printf("\ny0 = ");
    scanf("%f",&y0);
    printf("\nx1 = ");
    scanf("%f",&x1);
    printf("\ny1 = ");
    scanf("%f",&y1);
    printf("\nPontos = ");
    scanf("%d",&pontos);

    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(400,400);
    glutInitWindowPosition(50,50);
    glutCreateWindow("Ponto");
    glOrtho(0,399,399,0,-1,1);
    glutDisplayFunc(display);
    glutMainLoop();
}

```

Algumas desvantagens do programa proposto:

- 1) Este método requer operações em ponto flutuante (float ou double) para cada pixel. Isto acarreta em um algoritmo lento, se comparado a um algoritmo que opera somente com números inteiros. Para o caso da reta, existe um tal algoritmo que utiliza somente aritmética com números inteiros. Este algoritmo foi desenvolvido por Jack E. Bresenham na década de 60 e será descrito adiante.
- 2) O usuário estabelece o número de pontos da reta a serem plotados entre os dois pontos. Podem ocorrer dois casos: faltarem pontos (a reta fica pontilhada), sobrarem pontos (neste caso o algoritmo faz contas desnecessárias). O ideal é o próprio programa se encarregar de determinar o número de pontos necessários e suficientes para resolver o problema.
- 3) O caso particular da reta vertical $x = K$ (onde K é constante) não pode ser plotado.

1.6.2 – Retas no Opengl

O OpenGL dispõe em sua biblioteca interna de um comando que plota uma reta por dois pontos dados. Este comando é descrito abaixo:

```

void display()
{
    float x,y;

    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);
    glBegin(GL_LINES);
        glVertex2f(x0,y0);
        glVertex2f(x1,y1);
    glEnd();
    glFlush();
}

```

1.7 – Exemplo: Plotar o gráfico de uma função

Vamos começar com uma versão bem simplificada, para plotar o gráfico de uma função $y=f(x)$.

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

float funcao(float x)
{
    return(x*x);
}

void display()
{
    float x,y;

    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);
    for(x=-1;x<1;x+=0.01)
    {
        y = funcao(x);

        glBegin(GL_POINTS);
            glVertex2f(x,y);
        glEnd();
    }
    glFlush();
}

void main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(400,400);
    glutInitWindowPosition(50,50);
    glutCreateWindow("Ponto");
    gluOrtho2D(-1,1,-1,1);
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Vamos melhorar este programa:

- 1) Acrescentando os eixos.
- 2) Criando uma variável pontos que permita melhorar a discretização.
- 3) Separa os programa em dois módulos:
 funcao01.cpp: funções básicas de inicialização e controle do glut.
 funcao02.cpp: funções gráficas definidas com a tarefa específica de plotar o gráfico da função.

```

/* * * * * * * * * * * * * */
/* Modulo: Funcao01.cpp      */
/* * * * * * * * * * * * * */

#include <gl\glut.h>
#include <stdio.h>
#include <math.h>
#include "funcao02.h"

void redesenha()
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    plota_eixo();
    plota_funcao();
    glFlush();
    glutSwapBuffers();
}

void main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(400,400);
    glutInitWindowPosition(50,50);
    glutCreateWindow("Funcao");
    glutDisplayFunc(redesenha);
    gluOrtho2D(-5,5,-5,5);
    glutMainLoop();
}

```

```

/* * * * * * * * * * * * * */
/* Modulo: funcao02.cpp      */
/* * * * * * * * * * * * * */

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

#define PI 3.1415926535897932384626433832795

float xmin    = -5.0;
float xmax    =  5.0;
float ymin    = -5.0;
float ymax    =  5.0;
int  pontos   = 400;

/* ----- */
float funcao(float x)
/* ----- */
{
    return(sin(x));
}

/* ----- */
void plota_eixo()
/* ----- */
{

```



```

        glColor3f(0.0,1.0,0.0);
        glBegin(GL_LINES);
            glVertex2f(xmin,0);
            glVertex2f(xmax,0);
            glVertex2f(0,ymin);
            glVertex2f(0,ymax);
        glEnd();
    }

    /* ----- */
void plota_funcao()
/* ----- */
{
    int    i    ;
    float dx ;
    float x,y;

    dx = (xmax - xmin)/pontos;

    glColor3f(1.0,0.0,0.0);

    x = xmin;
    for(i=0;i<pontos;i++)
    {
        y = funcao(x);

        glBegin(GL_POINTS);
            glVertex2f(x,y);
        glEnd();

        x = x + dx;
    }
}

```

```

/* funcao02.h */

float funcao(float x);

void plota_eixo();

void plota_funcao();

```

Nossa próxima etapa é permitir que o usuário possa alterar o domínio de visualização da função interativamente. Na próxima sessão apresentamos algumas das principais interrupções do glut que nos permitirão executar esta tarefa.

2 – TECLADO E MOUSE (Callbacks)

2.1 – Introdução

O usuário pode interagir com o programa de duas formas principais: através do Mouse ou Teclado. Para isso o GLUT dispõe de dois tipos de funções (que denominamos Callbacks) específicas para habilitar a utilização do teclado e do mouse. Vamos descrevê-las a seguir.

2.2 – Teclado

Para registrar ocorrências no teclado o GLUT dispõe da função:

```
void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y))
```

Esta função determina que quando uma tecla for pressionada, o controle do programa deve passar a função definida no campo `(*func)` e esta função receberá como parâmetros de entrada, a tecla pressionada (`unsigned char key`), e a posição do mouse (`int x, int y`). O exemplo abaixo exemplifica uma aplicação para o programa **funções**.

2.3 – Exemplo: Utilização do teclado no programa funções.

Como exemplo vamos acrescentar no programa **funções** a possibilidade de alterarmos o domínio da função durante a execução do programa. Assim podemos estabelecer que quando o usuário pressionar a tecla “D” ou “d”, o programa interrompe e solicita na janela DOS as novas informações sobre o domínio da função.

Vamos reorganizar o programa, modularizando as principais rotinas do programa em arquivos diferentes.

Nosso programa terá dois módulos principais:

- `funcao010.cpp`: Mantém as rotinas de visualização do OpenGL
- `funcao011.cpp`: Mantém as rotinas relativas ao desenho da função.

Teremos ainda o módulo:

- `funcao011.h`: Este módulo deve ter apenas o cabeçalho das funções do programa `funcao011.cpp`.

A seguir apresentamos o código de cada um destes módulos.

2.3.1 - Módulo `funcao011.h`

```
/* * * * * * * * * * * * * * */
/* Modulo: funcao011.h          */
/* * * * * * * * * * * * * * */

float funcao(float x);

void plota_eixo();

void plota_funcao();

void entra_dominio();
```

2.3.2 - Módulo funcao010.cpp

Este módulo contém as rotinas básicas do OpenGL e a abertura do programa (void main()). Observe que a função glutKeyboardFunc(le_tecla); informa que quando alguma tecla é pressionada o controle do programa deve passar a rotina void le_tecla(unsigned char key, int x, int y).

```
/* * * * * * * * * * * * * * * */
/* Modulo: funcao010.cpp          */
/* * * * * * * * * * * * * * * */

#include <gl\glut.h>
#include <stdio.h>
#include <math.h>
#include "funcao011.h"

void display()
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    plota_eixo();
    plota_funcao();
    glFlush();
}

void le_tecla(unsigned char key, int x, int y)
{
    switch(key)
    {
        case 'D':
        case 'd':
            entra_dominio();
            display();
            break;
    }
}

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

2.3.3 - Módulo funcao011.cpp

Neste módulo separamos as rotinas que trabalham com a função ser desenhada.

```
/* * * * * * * * * * * * * * * */
/* Modulo: funcao011.cpp          */
/* * * * * * * * * * * * * * * */
#include <stdio.h>
#include <math.h>
#include <gl\glut.h>
```

```

float xmin,ymin,xmax,ymax,incremento;

float funcao(float x)
{
    return(sin(x));
}

void plota_eixo()
{
    glColor3f(0.0,1.0,0.0);
    glBegin(GL_LINES);
        glVertex2f(xmin,0);
        glVertex2f(xmax,0);
        glVertex2f(0,ymin);
        glVertex2f(0,ymax);
    glEnd();
}

void plota_funcao()
{
    float x,y;

    glColor3f(1.0,0.0,0.0);
    for(x=xmin;x<xmax;x+=incremento)
    {
        y = funcao(x);

        glBegin(GL_POINTS);
            glVertex2f(x,y);
        glEnd();
    }
}

void entra_dominio()
{
    int pontos;

    printf("xmin = ");
    scanf("%f",&xmin);
    printf("\nymin = ");
    scanf("%f",&ymin);
    printf("\nxmax = ");
    scanf("%f",&xmax);
    printf("\nymax = ");
    scanf("%f",&ymax);
    printf("\n Total de pontos =");
    scanf("%d",&pontos);

    glLoadIdentity();
    gluOrtho2D(xmin,xmax,ymin,ymax);

    incremento = fabs(xmax-xmin)/pontos;
}

```

2.4 – Mouse

O GLUT é capaz de obter três tipos de ocorrências diferentes a partir do mouse. Vamos descrevê-las em seguida e discutir uma interessante aplicação para o estudo de gráficos de funções.

2.4.1- Interrupções a partir do mouse

a) glutMouseFunc

```
void glutMouseFunc(void (*func)(int button, int state, int x, int y))
```

Este evento detecta quando algum botão do mouse foi pressionado. Quando isto ocorre, o programa executa a rotina definida em `void (*func)`. Os parâmetros desta rotina podem receber os seguintes valores:

- **button** : informa qual botão do mouse foi pressionado, sendo atribuído com um dos seguintes valores: GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON.
- **state**: informa quando o botão foi pressionado e quando o botão foi solto, sendo atribuído com dois possíveis valores: GLUT_DOWN ou GLUT_UP.
- **X,y**: informa a posição do mouse na janela quando o botão foi pressionado.

Como exemplo, vamos alterar o programa **função** anterior, e acrescentar outra forma de interrupção para alterar o domínio. Assim se o usuário pressionar o botão esquerdo do mouse, o programa solicita na janela DOS o novo domínio de visualização.

Para implementar esta alteração só precisamos mudar o programa `funcao010.cpp`. Seu novo código é apresentado a seguir:

```
/* * * * * * * * * * * * * * */
/* Modulo: funcao010.cpp      */
/* * * * * * * * * * * * * * */

#include <gl\glut.h>
#include <stdio.h>
#include <math.h>
#include "funcao011.h"

void display()
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    plota_eixo();
    plota_funcao();
    glFlush();
}

void le_tecla(unsigned char key, int x, int y)
{
    switch(key)
    {
        case 'D':
        case 'd':
            entra_dominio();
            display();
            break;
    }
}
```

```

void le_botao_mouse(int b,int state,int x, int y)
{
    switch(b) {
        case GLUT_RIGHT_BUTTON:
            switch(state) {
                case GLUT_DOWN:
                    printf("Botão direito pressionado em: x = %d y = %d \n",x,y);
                    break;
                case GLUT_UP:
                    printf("Botão solto em: x = %d y = %d \n",x,y);
                    break;
            }
            break;
    }
}

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

```

b) glutMotionFunc

```
void glutMotionFunc(void (*func)(int x, int y))
```

Este evento detecta o movimento do mouse enquanto algum botão do mouse está pressionado. Quando isto ocorre, o programa executa a rotina definida em void (*func) e informa em x,y a posição do mouse na janela.

No exemplo abaixo, a rotina le_botao_movimento_mouse(int x, int y) imprime a posição do mouse enquanto mantemos um de seus botões pressionados.

```

/* * * * * * * * * * * */
/* Programa mouse01.cpp */
/* * * * * * * * * * * */

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

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

void le_botao_movimento_mouse(int x, int y)
{
    printf("Botao+movimento x = %d y = %d \n",x,y);
}

```

```

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

```

c) glutPassiveMotionFunc

```
void glutPassiveMotionFunc(void (*func)(int x, int y))
```

Este evento detecta o movimento do mouse quando nenhum botão do mouse está pressionado. Quando isto ocorre, o programa executa a rotina definida em void (*func) e informa em x,y a posição do mouse na janela.

No exemplo abaixo, a rotina le_movimento_mouse(int x, int y) imprime a posição do mouse quando movimentamos o mouse dentro da janela OpenGL.

```

/* * * * * * * * * * * */
/* Programa mouse02.cpp */
/* * * * * * * * * * * */

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

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

void le_movimento_mouse(int x, int y)
{
    printf("Movimento x = %d y = %d \n",x,y);
}

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

```

2.4.2- Aplicações: Realizando o “zoom” do gráfico

No processo de visualização das funções, ou mesmo da integração numérica, muitas vezes desejamos alterar o domínio do nosso gráfico. Para isso selecionamos a tecla “D” e “d” como uma interrupção do teclado para que pudéssemos informar o novo domínio. Uma forma mais ágil seria selecionar com o

mouse interativamente um retângulo que gostaríamos de visualizar. É o que faremos no próximo programa.

Para isso utilizaremos a função `glutMotionFunc(funcao)` para selecionarmos a região desejada. Quando o botão direito do mouse é pressionado, marcamos o ponto inicial da região. Enquanto o mouse está em movimento (com o botão direito pressionado), desenhamos o retângulo desejado. Quando o botão do mouse é solto, obtemos o ponto final da região e atualizamos o domínio da função.

Observe que as coordenadas obtidas pelo `glutMotionFunc` precisam ser convertidas para o sistema de coordenadas indicado nos parâmetros do `gluOrtho2D`. Para isso, basta considerarmos a transformação inversa a aplicada na seção 1.5:

$$x = X_{\min} + t(X_{\max} - X_{\min}) \quad \text{e segue que } x = X_{\min} + \frac{P_x}{DIM_x - 1}(X_{\max} - X_{\min})$$

$$P_x = 0 + t(DIM_x - 1)$$

e identicamente para a coordenada y.

Vamos incorporar ao programa `Funcao.cpp` a nova ferramenta de zoom.

```
/* * * * * * * * * * * * * * * * * */
/* Modulo: Funcao.cpp (com zoom)      */
/* * * * * * * * * * * * * * * * * */

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

int mov = 0;          /* Detecta o movimento do mouse */

int xv1,xv2,yv1,yv2; /* Domínio da nova janela      */

extern int DIMX;
extern int DIMY;

void redesenha()
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    plota_eixo();
    plota_funcao();
    if (mov == 1)
        plota_retangulo(xv1,yv1,xv2,yv2);
    glFlush();
    glutSwapBuffers();
}

void le_botao_movimento_mouse(int x,int y)
{
    xv2 = x;
    yv2 = y;
    redesenha();
}

void le_botao_mouse(int b,int state,int x,int y)
{
    switch(b) {
        case GLUT_LEFT_BUTTON:
            switch(state) {
                case GLUT_DOWN:
                    xv1 = x;
                    yv1 = y;
            }
    }
}
```



```

                                mov = 1;
                                break;
                                case GLUT_UP:
                                    mov = 0;
                                    xv2 = x;
                                    yv2 = y;

recalcula_dominio(xv1,yv1,xv2,yv2);

                                redesenha();
                                break;

                                }
                                break;

        }
    }

void le_tecle(unsigned char key, int x, int y)
{
    switch(key)
    {
        case 'D':
        case 'd':
            entra_dominio();
            redesenha();
            break;

    }
}

void main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(DIMX,DIMY);
    glutInitWindowPosition(50,50);
    glutCreateWindow("Funcao");
    glutDisplayFunc(redesenha);
    glutKeyboardFunc(le_tecle);
    glutMouseFunc(le_botao_mouse);
    glutMotionFunc(le_botao_movimento_mouse);
    glutMainLoop();
}

```

```

/* * * * * * * * * * * * * * * * * */
/* Modulo: plota.cpp (com zoom) */
/* * * * * * * * * * * * * * * * * */
#include <stdio.h>
#include <math.h>
#include <gl\glut.h>
#include "funcao.h"
#include "dominio.h"

#define PI 3.1415926535897932384626433832795

extern float xmin ;
extern float ymin ;
extern float xmax ;
extern float ymax ;
extern int pontos;

extern int DIMX;
extern int DIMY;

/* ----- */

```

```

float funcao(float x)
/* ----- */
{
    return(sin(x));
}

/* ----- */
void plota_eixo()
/* ----- */
{
    glColor3f(0.0,1.0,0.0);
    glBegin(GL_LINES);
        glVertex2f(xmin,0);
        glVertex2f(xmax,0);
        glVertex2f(0,ymin);
        glVertex2f(0,ymax);
    glEnd();
}

/* ----- */
void plota_funcao()
/* ----- */
{
    int    i    ;
    float dx ;
    float x,y;

    dx = (xmax - xmin)/pontos;

    glColor3f(1.0,0.0,0.0);

    x = xmin;
    for(i=0;i<pontos;i++)
    {
        y = funcao(x);

        glBegin(GL_POINTS);
            glVertex2f(x,y);
        glEnd();

        x = x + dx;
    }
}

/* ----- */
void plota_retangulo(int x,int y,int xv1,int yv1)
/* ----- */
{
    float t;
    float retxmin,retxmax,retymin,retymax;

    retxmin = converte(xv1,xmin,xmax,DIMX);
    retxmax = converte(x ,xmin,xmax,DIMX);
    retymin = converte(y ,ymax,ymin,DIMY);
    retymax = converte(yv1,ymax,ymin,DIMY);

    glColor3f(1.0,1.0,1.0);
    glBegin(GL_LINE_LOOP);
        glVertex2f(retxmin,retymin);
        glVertex2f(retxmax,retymax);
        glVertex2f(retxmax,retymax);

```

```

        glVertex2f(retxmax, retymax);
    glEnd();
}

```

```

/* * * * * * * * * * * * * * * * */
/* Modulo: dominio.cpp                */
/* * * * * * * * * * * * * * * * */
#include <gl\glut.h>
#include <stdio.h>

int DIMX = 400;
int DIMY = 400;

float xmin = -1.0;
float ymin = -1.0;
float xmax = 1.0;
float ymax = 1.0;
int pontos = 400;

/* ----- */
float converte(float p, float min, float max, int dim)
/* ----- */
{
    float x;

    x = min + p/(dim - 1) * (max-min);
    return (x);
}

/* ----- */
void entra_dominio()
/* ----- */
{
    printf("xmin = ");
    scanf("%f", &xmin);
    printf("\nymin = ");
    scanf("%f", &ymin);
    printf("\nxmax = ");
    scanf("%f", &xmax);
    printf("\nymax = ");
    scanf("%f", &ymax);
    printf("\n Total de pontos =");
    scanf("%d", &pontos);

    glLoadIdentity();
    gluOrtho2D(xmin, xmax, ymin, ymax);
}

/* ----- */
void recalcula_dominio(float xv_1, float yv_1, float xv_2, float yv_2)
/* ----- */
{
    float t;
    float xmin1, xmax1;
    float ymin1, ymax1;

    xmin1 = converte(xv_1, xmin, xmax, DIMX);
    xmax1 = converte(xv_2, xmin, xmax, DIMX);
    xmin = xmin1;
    xmax = xmax1;
}

```

```
        ymin1 = converte(yv_2,ymax,ymin,DIMY);
        ymax1 = converte(yv_1,ymax,ymin,DIMY);
        ymin = ymin1;
        ymax = ymax1;

    glLoadIdentity();
    gluOrtho2D(xmin,xmax,ymin,ymax);
}
```

3 – CURVAS PARAMÉTRICAS

3.1 – Introdução

Considere uma curva C representando a trajetória de uma partícula P , de tal forma que a posição $P(x, y)$ da partícula é conhecida em cada instante de tempo t .

Assim as coordenadas x e y são conhecidas como funções da variável t de modo que:

$$\begin{aligned}x &= x(t) \\ y &= y(t)\end{aligned}$$

Estas são as equações paramétricas da curva C e t é denominado parâmetro. Como exemplo de curvas temos:

a) Circunferência de centro na origem e raio 1:

$$\begin{aligned}x &= \cos(t) \\ y &= \sin(t) \quad \text{onde } 0 \leq t \leq 2\pi\end{aligned}$$

b) Ciclóide (curva traçada por um ponto da circunferência quando o círculo rola sobre uma reta):

$$\begin{aligned}x &= t - \sin(t) \\ y &= 1 - \cos(t)\end{aligned}$$

3.2 – Exemplo: Visualização de Curvas Paramétricas

Vamos implementar um programa que visualize curvas paramétricas. Observe que a diferença principal para o programa funções é que tanto a variável x , quanto y devem ser calculadas em função do parâmetro t . Vamos aproveitar uma boa parte da estrutura já implementada no programa **função**. O programa está dividido em dois módulos principais:

- **curva.cpp**: Mantém as rotinas de visualização do OpenGL
- **plota.cpp**: Rotinas relativas ao desenho da curva.
- **dominio.cpp**: Rotinas relativas ao domínio e zoom.

3.2.1 – Módulo **curvas.cpp**

Este módulo é idêntico ao módulo **funcao010.cpp**. A única diferença é que as chamadas são feitas para as novas funções de desenho das curvas.

```
/* * * * * * * * * * * * * * * * */
/* Modulo: Curva.cpp (com zoom) */
/* * * * * * * * * * * * * * * * */

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

int mov = 0; /* Detecta o movimento do mouse */

int xv1,xv2,yv1,yv2; /* Domínio da nova janela */

extern int DIMX;
extern int DIMY;
extern float xmin;
extern float ymin;
extern float xmax;
extern float ymax;

void redesenha()
{
    glClearColor(0.0,0.0,0.0,0.0);
```

```

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
        plota_eixo();
        plota_curva();
        if (mov == 1)
            plota_retangulo(xv1,yv1,xv2,yv2);
        glFlush();
        glutSwapBuffers();
    }

void le_botao_movimento_mouse(int x,int y)
{
    xv2 = x;
    yv2 = y;
    redesenha();
}

void le_botao_mouse(int b,int state,int x,int y)
{
    switch(b) {
        case GLUT_LEFT_BUTTON:
            switch(state) {
                case GLUT_DOWN:
                    xv1 = x;
                    yv1 = y;
                    mov = 1;
                    break;
                case GLUT_UP:
                    mov = 0;
                    xv2 = x;
                    yv2 = y;
                    recalcula_dominio(xv1,yv1,xv2,yv2);
                    redesenha();
                    break;
            }
            break;
    }
}

void le_tecle(unsigned char key, int x, int y)
{
    switch(key)
    {
        case 'D':
        case 'd':
            entra_dominio();
            redesenha();
            break;
    }
}

void main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(DIMX,DIMY);
    glutInitWindowPosition(50,50);
    glutCreateWindow("Funcao");
    glutDisplayFunc(redesenha);
    glutKeyboardFunc(le_tecle);
    glutMouseFunc(le_botao_mouse);
    glutMotionFunc(le_botao_movimento_mouse);
    gluOrtho2D(xmin,xmax,ymin,ymax);
}

```

```

        glutMainLoop();
    }

```

3.2.2 – Módulo **plota.cpp**

Neste módulo implementamos o desenho do gráfico das curvas paramétricas.

```

/* * * * * * * * * * * * * * * * * */
/* Módulo: plota.cpp (com zoom) */
/* * * * * * * * * * * * * * * * */
#include <stdio.h>
#include <math.h>
#include <gl\glut.h>
#include "curva.h"
#include "dominio.h"

#define PI 3.1415926535897932384626433832795

extern float xmin ;
extern float ymin ;
extern float xmax ;
extern float ymax ;
extern int pontos;

extern int DIMX;
extern int DIMY;

/* ----- */
void curva(float *x,float *y,float t)
/* ----- */
{
    // *x = sin(3*t);
    // *y = sin(4*t);
    // *x = sin(t+sin(t));
    // *y = cos(t+cos(t));
    // *x = t+2*sin(2*t);
    // *y = t+2*cos(5*t);
    // *x = cos(t)-cos(80*t)*sin(t);
    // *y = 2*sin(t)-sin(80*t);
    // *x = 31 * cos(t) - 7 * cos(31.0/7 * t);
    // *y = 31 * sin(t) - 7 * sin(31.0/7 * t);
    *x = cos(t) + 0.5 * cos(7*t) + 1/3.0 * sin(17*t);
    *y = sin(t) + 0.5 * sin(7*t) + 1/3.0 * cos(17*t);
}

/* ----- */
void plota_eixo()
/* ----- */
{
    glColor3f(0.0,1.0,0.0);
    glBegin(GL_LINES);
        glVertex2f(xmin,0);
        glVertex2f(xmax,0);
        glVertex2f(0,ymin);
        glVertex2f(0,ymax);
    glEnd();
}

/* ----- */
void plota_curva()
/* ----- */
{

```

```

    int i ;
    float t,dt ;
    float tmin = -5;
    float tmax = 2 * PI;
    float x,y;

    dt = (tmax - tmin)/pontos;

    glColor3f(1.0,0.0,0.0);

    t = tmin;
    for(i=0;i<pontos;i++)
    {

        curva(&x,&y,t);

        glBegin(GL_POINTS);
            glVertex2f(x,y);
        glEnd();

        t = t + dt;
    }
}

/* ----- */
void plota_retangulo(int x,int y,int xv1,int yv1)
/* ----- */
{
    float retxmin,retxmax,retymin,retymax;

    retxmin = converte(xv1,xmin,xmax,DIMX);
    retxmax = converte(x ,xmin,xmax,DIMX);
    retymin = converte(y ,ymax,ymin,DIMY);
    retymax = converte(yv1,ymax,ymin,DIMY);

    glColor3f(1.0,1.0,1.0);
    glBegin(GL_LINE_LOOP);
        glVertex2f(retxmin,retymin);
        glVertex2f(retxmin,retymax);
        glVertex2f(retxmax,retymax);
        glVertex2f(retxmax,retymin);
    glEnd();
}

```

3.2.3 – Módulo dominio.cpp

```

/* * * * * * */
/* Modulo: dominio.cpp */
/* * * * * * */
#include <gl\glut.h>
#include <stdio.h>

int DIMX = 400;
int DIMY = 400;

float xmin = -5.0;
float ymin = -5.0;
float xmax = 5.0;
float ymax = 5.0;
int pontos = 8000;

/* ----- */
float converte(float p,float min,float max,int dim)

```



```

/* ----- */
{
    float x;

    x = min + p/(dim - 1) * (max-min);
    return (x);
}

/* ----- */
void entra_dominio()
/* ----- */
{

    printf("xmin = ");
    scanf("%f",&xmin);
    printf("\nymin = ");
    scanf("%f",&ymin);
    printf("\nxmax = ");
    scanf("%f",&xmax);
    printf("\nymax = ");
    scanf("%f",&ymax);
    printf("\n Total de pontos =");
    scanf("%d",&pontos);

    glLoadIdentity();
    gluOrtho2D(xmin,xmax,ymin,ymax);

}

/* ----- */
void recalcula_dominio(float xv_1,float yv_1,float xv_2,float yv_2)
/* ----- */
{

    float xmin1,xmax1;
    float ymin1,ymax1;

    xmin1 = converte(xv_1,xmin,xmax,DIMX);
    xmax1 = converte(xv_2,xmin,xmax,DIMX);
    xmin = xmin1;
    xmax = xmax1;

    ymin1 = converte(yv_2,ymin,ymax,DIMY);
    ymax1 = converte(yv_1,ymin,ymax,DIMY);
    ymin = ymin1;
    ymax = ymax1;

    glLoadIdentity();
    gluOrtho2D(xmin,xmax,ymin,ymax);

}

```

3.2.4 –Exercício

Como exercício implemente:

- 1) Curva cicloide (t assume qualquer valor real).
- 2) $x(t) = 3*t*t$, $y(t)=4*t*t*t$ (t assume qualquer valor real).
- 3) $x(t) = \cos(2*t)$, $y(t)= \sin(2*t)$ ($0 \leq t \leq 2*PI$) (Qual a diferença para a curva do programa ?)
- 4) $x(t) = \cos(t)$, $y(t)= \sin(2*t)$
- 5) $x(t) = 2 * \cos(t)$, $y(t)= 3 * \sin(t)$ ($0 \leq t \leq 2*PI$).
- 6) Como você poderia visualizar gráficos de funções em uma variável $y = f(x)$ com este programa ? Visualize $y=x*x$, $y = \sin(x)$, $y = \ln(x)$.

3.3 – Curvas na forma Polar

Para formar as coordenadas polares considere um ponto fixo O, denominado origem (ou polo) e um eixo partindo de O, denominado eixo polar. A cada ponto P do plano podemos associar uma par de coordenadas polares (r, θ) onde:

r:	distância orientada da origem ao ponto P.
θ :	ângulo entre o eixo polar e o segmento OP.

As coordenadas polares podem ser relacionadas com as coordenadas retangulares (ou cartesianas) através das expressões abaixo:

$$\begin{cases} r^2 = x^2 + y^2 \\ \operatorname{tg}(\theta) = \frac{y}{x} \\ \begin{cases} x = r \cos(\theta) \\ y = r \operatorname{sen}(\theta) \end{cases} \end{cases}$$

Como exemplo de curvas na forma polar temos:

a) Circunferência de centro na origem e raio 1:

$$r = 1$$

b) Reta passando na origem com coeficiente angular m:

$$\theta = m$$

c) Circunferência com centro em $P(0, 0.5)$ e raio 1:

$$r = \sin \theta$$

d) Cardióide

$$r = a(1 + \cos(\theta))$$

e) Espiral

$$r = a\theta$$

f) Rosácea

$$r = a \operatorname{sen}(n\theta)$$

3.4 – Exemplo: Visualização de Curvas Polares

Para visualizar as curvas polares, podemos utilizar o mesmo programa das curvas paramétricas. Para isso, considere uma curva dada na forma polar:

$$r = f(\theta)$$

Em coordenadas cartesianas temos:

$$\begin{cases} x = r \cos(\theta) \\ y = r \sin(\theta) \end{cases}$$

Substituindo r nas duas equações obtemos:

$$\begin{cases} x = f(\theta) \cos(\theta) \\ y = f(\theta) \sin(\theta) \end{cases}$$

Assim temos uma curva na forma paramétrica. Como exemplo vamos visualizar a curva do cardióide, alterando apenas a rotina `curva` do programa anterior:

```
/* ----- */
void curva(float t,float *x,float *y)
/* float t           (entrada) */
/* float *x          (saida) */
/* float *y          (saida) */
/* ----- */
/* ----- */
{
    *x = (1+cos(t))*cos(t);
    *y = (1+cos(t))*sin(t);
}
```

3.5 – Exercícios

- 1) Como exercício visualize as demais curvas dadas em coordenadas polares.
- 2) Visualize a curva dada em coordenadas polares por $r = \sec(\theta)$.
- 3) Faça o gráfico das seguintes curvas:

(a) $r = \sin\left(\frac{8\theta}{5}\right)$

(b) $r = \sin \theta + \sin^3(5\theta/2)$

(c) $r = 1 + 4 \cos(5\theta)$

(d) $r = \sec(3\theta)$

(e) $r = \sin(\theta/4)$

(f) $r = \frac{1}{\sqrt{\theta}}$

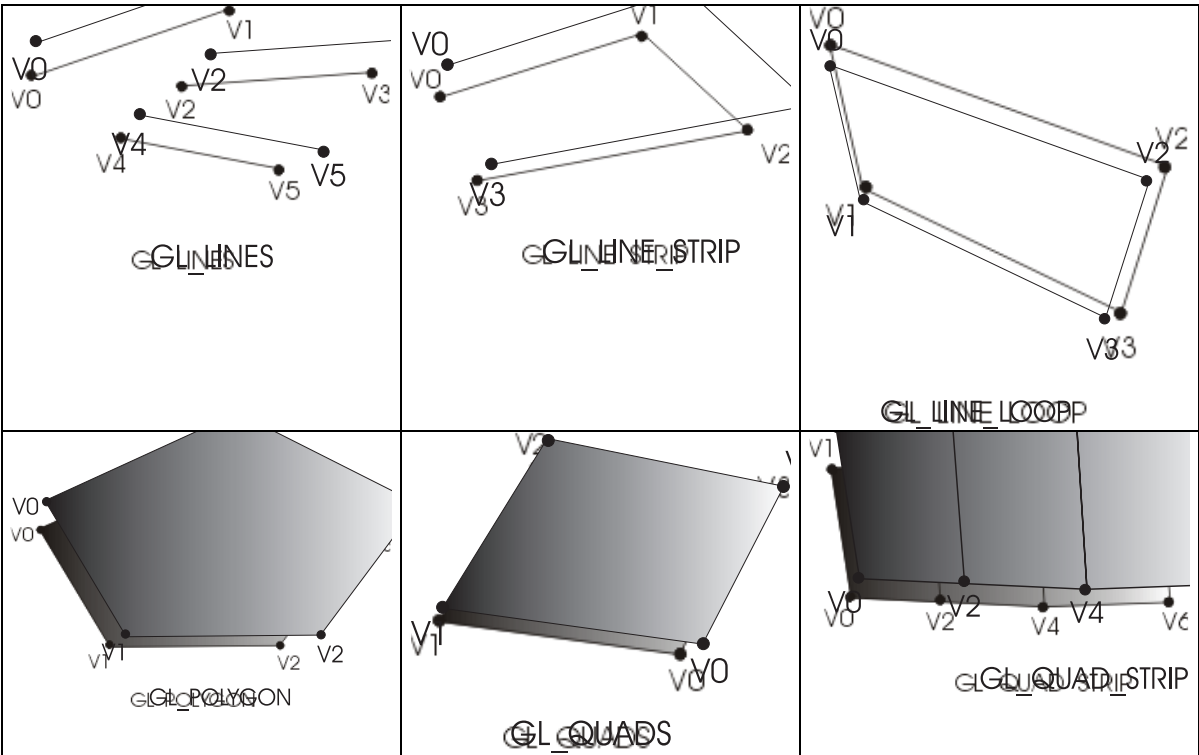
4 – RETAS E POLÍGONOS NO OPENGL

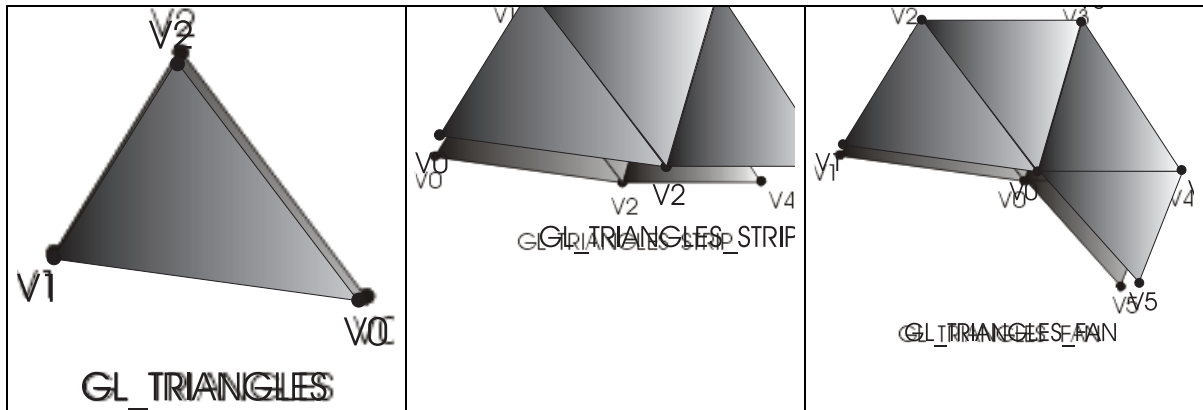
4.1 – Retas e Polígonos

Além de pontos e retas, o OpenGL possui no total 10 tipos de primitivas úteis. Todos os modelos da tabela abaixo devem ser utilizados iniciando com `glBegin(...)` e finalizando com `glEnd(...)`, por exemplo para o `GL_LINES` temos:

```
glBegin(GL_LINES);
    glVertex2f(1.0,1.0);
    glVertex2f(1.0,2.0);
    glVertex2f(2.0,-2.0);
    ...
glEnd();
```

GL_POINTS	Pontos individuais.
GL_LINES	Reta entre dois pontos.
GL_POLYGON	Polígono convexo .
GL_TRIANGLES	Tripla de vértices é interpretado como um triângulo.
GL_QUADS	Conjunto de quatro vértices interpretado como quadrilátero.
GL_LINE_STRIP	Sequência de retas.
GL_LINE_LOOP	Idêntico ao anterior, porém com uma reta unindo o primeiro e último vértice.
GL_TRIANGLE_STRIP	Lista de triângulos.
GL_TRIANGLE_FAN	Lista de triângulos com o primeiro vértice em comum.
GL_QUAD_STRIP	Lista de quadriláteros.





4.2 – Exemplo: Visualização dos Métodos Numéricos de Integração

Como exemplo de aplicação dos novos objetos vamos visualizar alguns métodos numéricos de integração: ponto à direita, ponto à esquerda e trapézio.

Nosso programa terá três módulos principais:

- Intvis01.cpp: Mantém as rotinas de visualização do OpenGL
- Intvis02.cpp: Mantém as rotinas relativas ao esboço do gráfico da função.
- Intvis03.cpp: Mantém as rotinas de visualização dos métodos numéricos de integração.

A seguir apresentamos o código de cada um destes módulos.

4.2.1 - Módulo Intvis01.c

```

/* * * * * * * * * * * * * * */
/* Modulo: Intvis01.c          */
/* * * * * * * * * * * * * * */

#include <gl\glut.h>
#include <stdio.h>
#include <math.h>
#include "Intvis02.h"
#include "Intvis03.h"

extern int tipo,visual;

void display()
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    plota_eixo();
    plota_funcao();
    plota_integral();
    glFlush();
    glutSwapBuffers();
}

void le_tecla(unsigned char key, int x, int y)
{
    switch(key)
    {
        case 'D':
        case 'd':
            entra_dominio();
            display();
            break;
    }
}

```

```

        case 'I':
        case 'i':
            entra_limites();
            display();
            break;

        case '0':
            tipo = 0;
            display();
            break;

        case '1':
            tipo = 1;
            display();
            break;
    }
}

void le_botao_mouse(int b,int state,int x, int y)
{
    switch(b) {
        case GLUT_RIGHT_BUTTON:
            switch(state) {
                case GLUT_DOWN:
                    visual = (visual + 1) % 2;
                    display();
                    break;

                case GLUT_UP:
                    break;
            }
            break;
    }
}

void main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(400,400);
    glutInitWindowPosition(50,50);
    glutCreateWindow("Ponto");
    glutDisplayFunc(display);
    glutKeyboardFunc(le_tecla);
    gluOrtho2D(-10.0,10.0,-10.0,10.0);
    glutMouseFunc(le_botao_mouse);
    glutMainLoop();
}

```

4.2.2 - Módulo Intvis02.cpp

```
/* * * * * * * * * * * * * * * */
/* Modulo: Intvis02.cpp           */
/* * * * * * * * * * * * * * * */

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

float xmin = -10.0;
float ymin = -10.0;
float xmax = 10.0;
float ymax = 10.0;
float incremento = 0.01;

float funcao(float x)
{
    return(sin(x));
}

void plota_eixo()
{
    glColor3f(0.0,1.0,0.0);
    glBegin(GL_LINES);
        glVertex2f(xmin,0);
        glVertex2f(xmax,0);
        glVertex2f(0,ymin);
        glVertex2f(0,ymax);
    glEnd();
}

void plota_funcao()
{
    float x,y;

    glColor3f(1.0,0.0,0.0);
    for(x=xmin;x<ymax;x+=incremento)
    {
        y = funcao(x);

        glBegin(GL_POINTS);
            glVertex2f(x,y);
        glEnd();
    }
}

void entra_dominio()
{
    int pontos;

    printf("xmin = ");
    scanf("%f",&xmin);
    printf("\nymin = ");
    scanf("%f",&ymin);
    printf("\nxmax = ");
    scanf("%f",&xmax);
    printf("\nymax = ");
    scanf("%f",&ymax);
    printf("\n Total de pontos =");
    scanf("%d",&pontos);
}
```

```

        glLoadIdentity();
        gluOrtho2D(xmin,xmax,ymin,ymax);

        incremento = fabs(xmax-xmin)/pontos;
    }

```

4.2.3 - Módulo Intvis03.cpp

Neste módulo separamos as rotinas que responsáveis pela visualização dos numéricos métodos de integração.

```

/* * * * * * * * * * * * * * */
/* Modulo: Intvis03.cpp          */
/* * * * * * * * * * * * * * */

#include <gl\glut.h>
#include <stdio.h>
#include "Intvis02.h"

float a = 0;
float b = 1;
int tipo = 0;
int visual = 0;
int divisoes = 5;

void Integral_retangulo()
{
    int i ;
    float x,y;
    float dx;

    dx = (b-a)/divisoes;
    for(i=0; i < divisoes; i++)
    {
        x = a+ i*dx;
        y = funcao(x);
        glColor3f(1.0,1.0,0.0);
        if (visual == 0)
            glBegin(GL_LINE_LOOP);
        else
            glBegin(GL_POLYGON);
        glVertex2f(x,y);
        glVertex2f(x+dx,y);
        glVertex2f(x+dx,0);
        glVertex2f(x,0);
        glEnd();
    }
}

void Integral_trapezio()
{
    int i ;
    float x,y1,y2;
    float dx;

```



```

    dx = (b-a)/divisoes;
    for(i=0; i < divisoes; i++)
    {
        x = a+ i*dx;
        y1 = funcao(x);
        y2 = funcao(x + dx);
        glColor3f(1.0,1.0,0.0);
        if (visual == 0)
            glBegin(GL_LINE_LOOP);
        else
            glBegin(GL_POLYGON);
        glVertex2f(x,y1);
        glVertex2f(x+dx,y2);
        glVertex2f(x+dx,0);
        glVertex2f(x,0);
        glEnd();
    }
}

void plota_integral()
{
    switch(tipo) {
        case 0:
            Integral_retangulo();
            break;
        case 1:
            Integral_trapezio();
            break;
    }
}

void entra_limites()
{
    printf("Limites de Integracao \n");
    printf("a = ");
    scanf("%f",&a);
    printf("\nb = ");
    scanf("%f",&b);
    printf("\ndivisoes = ");
    scanf("%d",&divisoes);
}

```

5 – CURVAS IMPLÍCITAS

5.1 – Introdução

Já aprendemos na seção 3 como representar curvas na forma paramétrica. Vamos discutir agora outro tipo de representação muito utilizada para curvas: a representação implícita.

A equação implícita de uma curva descreve uma relação entre as coordenadas x e y dos pontos que pertencem a curva. Assim no plano xy a equação implícita de uma curva tem a forma :

$$f(x, y) = 0$$

Como exemplo a representação implícita de uma circunferência de raio 1 centrado na origem é dado por:

$$x^2 + y^2 - 1 = 0$$

Na forma paramétrica a mesma curva é representada por:

$$\begin{cases} x = \cos(t) \\ y = \sin(t) \end{cases} \quad 0 \leq t \leq 2\pi$$

Qual das duas representações é mais vantajosa em termos computacionais ? Na verdade ambas representações têm vantagens e desvantagens em comparação uma com a outra. Por exemplo, é muito simples determinar se um ponto $P(x_0, y_0)$ dado pertence ou não a uma curva dada na forma implícita.

Já na forma paramétrica é simples determinar pontos que pertençam a curva, para que se possa fazer uma representação gráfica da curva (como foi feito na seção anterior). Vamos agora resolver este último problema para uma curva dada na forma implícita, ou seja, vamos representar graficamente a curva implícita.

5.2 – Visualização de Curvas Implícitas

Vamos implementar um programa que visualize curvas implícitas. Partindo por exemplo da equação:

$$\sqrt{x^2 + y^2} - \frac{x^2 - y^2}{x^2 + y^2} + \sin(xy) = 0$$

Observe que não é simples exibir um conjunto de pontos que pertençam a esta curva. Vamos definir uma função de duas variáveis $z = f(x, y)$ utilizando a equação acima, da seguinte forma:

$$f : \mathbb{R}^2 \rightarrow \mathbb{R}$$

$$f(x, y) = \sqrt{x^2 + y^2} - \frac{x^2 - y^2}{x^2 + y^2} + \sin(xy)$$

Assim a curva inicial desejada, será a curva de nível $f(x, y) = 0$.

A estratégia para obter esta curva será a seguinte:

- Vamos estabelecer um domínio $[-2, 2] \times [-2, 2]$ no plano como partida (a priori não sabemos se existem ou não pontos da curva nesse domínio).
- Em seguida discretizamos este domínio, determinando uma matriz de 10x10 pontos $P_{ij}(x_{ij}, y_{ij})$, por exemplo.
- A cada três pontos, definimos um triângulo como na figura abaixo.
- Para cada ponto $P_{ij}(x_{ij}, y_{ij})$ calculamos $z_{ij} = f(x_{ij}, y_{ij})$.
- Para cada triângulo, observamos os sinais $V_i = \text{sign}(z_{ij})$ obtidos em cada vértice e temos as seguintes situações:
 - Se $V1 * V2 < 0$, então a função se anula em um ponto entre $V1$ e $V2$. Este ponto pode ser aproximado linearmente.
 - Se $V1 * V3 < 0$, então a função se anula em um ponto entre $V1$ e $V3$.

- Se $V2 * V3 < 0$, então a função se anula em um ponto entre V2 e V3.
- Se $V1 = 0$, então a função se anula exatamente sobre o vértice V1.
- Se $V2 = 0$, então a função se anula exatamente sobre o vértice V2.
- Se $V3 = 0$, então a função se anula exatamente sobre o vértice V3.
- Considerando que exatamente duas das condições acima se verificaram simultaneamente, aproximamos a curva nesse triângulo por um segmento de reta unindo os dois pontos obtidos.

5.3 – Programa Curva Implícita

O programa está dividido em dois módulos principais:

- `main.cpp`: Contém as rotinas de visualização do OpenGL
- `graphics.cpp`: Contém as rotinas relativas ao desenho da curva.
- `dominio.cpp`: Contém as rotinas básicas do domínio da função.

5.3.1 – Módulo `main.cpp`

Este módulo é idêntico ao módulo `funcao010.cpp`. A única diferença é que as chamadas são feitas para as novas funções de desenho das curvas.

```
#include <gl\glut.h>
#include <stdio.h>
#include <math.h>
#include "plota.h"
#include "dominio.h"

int mov = 0;          /* Detecta o movimento do mouse */

int xv1,xv2,yv1,yv2; /* Domínio da nova janela      */

extern int DIMX;
extern int DIMY;
extern float xmin;
extern float ymin;
extern float xmax;
extern float ymax;

void redesenha()
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    plota_eixo();
    plota_curva();
    if (mov == 1)
        plota_retangulo(xv1,yv1,xv2,yv2);
    glFlush();
    glutSwapBuffers();
}

void le_botao_movimento_mouse(int x,int y)
{
    xv2 = x;
    yv2 = y;
    redesenha();
}

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

```

```

switch(b) {
    case GLUT_LEFT_BUTTON:
        switch(state) {
            case GLUT_DOWN:
                xv1 = x;
                yv1 = y;
                mov = 1;
                break;
            case GLUT_UP:
                mov = 0;
                xv2 = x;
                yv2 = y;
                recalcula_dominio(xv1,yv1,xv2,yv2);
                redesenha();
                break;
        }
        break;
}

void le_tecla(unsigned char key, int x, int y)
{
    switch(key)
    {
        case 'D':
        case 'd':
            entra_dominio();
            redesenha();
            break;
    }
}

void main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(DIMX,DIMY);
    glutInitWindowPosition(50,50);
    glutCreateWindow("Funcao");
    glutDisplayFunc(redesenha);
    glutKeyboardFunc(le_tecla);
    glutMouseFunc(le_botao_mouse);
    glutMotionFunc(le_botao_movimento_mouse);
    gluOrtho2D(xmin,xmax,ymin,ymax);
    glutMainLoop();
}

```

5.3.2 – Módulo **graphics.cpp**

Neste módulo implementamos o desenho do gráfico das curvas implícitas. As funções `plota_eixo` e `entra_dominio` não sofrem alterações.

```

/* * * * * * * * * * * * * * * * * * * * */
/* Módulo: graphics.cpp (com zoom) */
/* * * * * * * * * * * * * * * * * * * * */
#include <stdio.h>
#include <math.h>
#include <gl\glut.h>
#include "curva.h"
#include "dominio.h"

#define PI 3.1415926535897932384626433832795

```

```

extern float xmin ;
extern float ymin ;
extern float xmax ;
extern float ymax ;
extern int pontos;

extern int DIMX;
extern int DIMY;

/* ----- */
void plota_eixo()
/* ----- */
{
    glColor3f(0.0,1.0,0.0);
    glBegin(GL_LINES);
        glVertex2f(xmin,0);
        glVertex2f(xmax,0);
        glVertex2f(0,ymin);
        glVertex2f(0,ymax);
    glEnd();
}

/* ----- */
void plota_retangulo(int x,int y,int xv1,int yv1)
/* ----- */
{
    float retxmin,retxmax,retymin,retymax;

    retxmin = converte(xv1,xmin,xmax,DIMX);
    retxmax = converte(x ,xmin,xmax,DIMX);
    retymin = converte(y ,ymax,ymin,DIMY);
    retymax = converte(yv1,ymax,ymin,DIMY);

    glColor3f(1.0,1.0,1.0);
    glBegin(GL_LINE_LOOP);
        glVertex2f(retxmin,retymin);
        glVertex2f(retxmin,retymax);
        glVertex2f(retxmax,retymax);
        glVertex2f(retxmax,retymin);
    glEnd();
}

/* ----- */
float funcao(float x,float y)
{
    return(x*x-y*y-0.1);
    return(sqrt(x*x+y*y)-(x*x-y*y)/(x*x+y*y)+sin(x*y));
    return(x*x+y*y-1);
}

/* ----- */

/* ----- */
void curva_implicita(float *xf,float *yf)
/* ----- */
{
    int j;
    int i = 0;
    float t;
    float x[3],y[3];
    float s[3];

```

```

glColor3f(0.0,0.0,1.0);
glBegin(GL_LINE_LOOP);
    glVertex2f(xf[0],yf[0]);
    glVertex2f(xf[1],yf[1]);
    glVertex2f(xf[2],yf[2]);
glEnd();

for(j=0;j<3;j++)
    s[j] = funcao(xf[j],yf[j]);

if ((s[0] * s[1]) < 0) {
    t = -s[0]/(s[1]-s[0]);
    x[i] = xf[0] + t * (xf[1]-xf[0]);
    y[i] = yf[0];
    i++;
}
if ((s[0] * s[2]) < 0) {
    t = -s[0]/(s[2]-s[0]);
    x[i] = xf[0] ;
    y[i] = yf[0] + t * (yf[2]-yf[0]);
    i++;
}
if ((s[1] * s[2]) < 0) {
    t = -s[1]/(s[2]-s[1]);
    x[i] = xf[1] + t * (xf[2]-xf[1]);
    y[i] = yf[1] + t * (yf[2]-yf[1]);
    i++;
}

for(j=0;j<3;j++) {
    if (s[j] == 0) {
        x[i] = xf[j];
        y[i] = yf[j];
        i++;
    }
}

if (i == 2) {
    glLineWidth(2.0);
    glColor3f(1.0,0.0,0.0);
    glBegin(GL_LINES);
        glVertex2f(x[0],y[0]);
        glVertex2f(x[1],y[1]);
    glEnd();
    glLineWidth(1.0);
}
}

/* ----- */
void plota_curva()
/* ----- */
{
    int i,j;
    float tx[3],ty[3];
    float x,y;
    float dx = (xmax - xmin)/pontos;
    float dy = (ymax - ymin)/pontos;

    glColor3f(1.0,0.0,0.0);
    x = xmin;
    for(i=0;i<pontos;i++)
    {

```

```

        y = ymin;
        for(j=0;j<pontos;j++)
        {
            tx[0] = x; tx[1] = x + dx; tx[2] = x;
            ty[0] = y; ty[1] = y; ty[2] = y + dy;
            curva_implicita(tx,ty);
            tx[0] = x + dx; tx[1] = x; tx[2] = x + dx;
            ty[0] = y + dy; ty[1] = y + dy; ty[2] = y;
            curva_implicita(tx,ty);
            y += dy;
        }
        x += dx;
    }
}

```

5.3.3 – Módulo dominio.cpp

```

/* * * * * * * * * * * * * * * * */
/* Modulo: dominio.cpp */
/* * * * * * * * * * * * * * * */
#include <gl\glut.h>
#include <stdio.h>

int DIMX = 400;
int DIMY = 400;

float xmin = -2.0;
float ymin = -2.0;
float xmax = 2.0;
float ymax = 2.0;
int pontos = 30;

/* ----- */
float converte(float p,float min,float max,int dim)
/* ----- */
{
    float x;

    x = min + p/(dim - 1) * (max-min);
    return (x);
}

/* ----- */
void entra_dominio()
/* ----- */
{
    printf("xmin = ");
    scanf("%f",&xmin);
    printf("\nymin = ");
    scanf("%f",&ymin);
    printf("\nxmax = ");
    scanf("%f",&xmax);
    printf("\nymax = ");
    scanf("%f",&ymax);
    printf("\n Total de pontos =");
    scanf("%d",&pontos);

    glLoadIdentity();
    gluOrtho2D(xmin,xmax,ymin,ymax);
}

```

```

/* ----- */
void recalcula_dominio(float xv_1,float yv_1,float xv_2,float yv_2)
/* ----- */
{
    float xmin1,xmax1;
    float ymin1,ymax1;

    xmin1 = converte(xv_1,xmin,xmax,DIMX);
    xmax1 = converte(xv_2,xmin,xmax,DIMX);
    xmin = xmin1;
    xmax = xmax1;

    ymin1 = converte(yv_2,ymax,ymin,DIMY);
    ymax1 = converte(yv_1,ymax,ymin,DIMY);
    ymin = ymin1;
    ymax = ymax1;

    glLoadIdentity();
    gluOrtho2D(xmin,xmax,ymin,ymax);
}

```

5.4 – Exercício

1) Como exercício implemente as seguintes curvas implícitas:

a) $x^2 + y^2 - 1 = 0$. c) $x^2 - y^2 + \frac{1}{10} = 0$.

b) $x^2 - y^2 = 0$. d) $4x^2 + 9y^2 = 1$

2) Implemente no programa uma rotina que imprima simultaneamente varias curvas de nivel de uma mesma função f , ou seja $f(x,y) = c$. Por exemplo, $f(x,y) = x^2 - y^2$.

6 – PROCESSAMENTO DE IMAGEM

6.1 – Introdução

Filtro da media e Gradiente

Imagem.cpp

```

/* * * * * * * * * * * */
/* Modulo: Imagem.cpp    */
/* * * * * * * * * * * */

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

void redesenha()
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    plota_imagem();
    glFlush();
    glutSwapBuffers();
}

void le_tecle(unsigned char key, int x, int y)
{
    switch(key)

```



```

    {

        case 'I':
        case 'i':
            le_imagem();
            redesenha();
            break;

        case 'M':
        case 'm':
            filtro_media();
            redesenha();
            break;

        case 'G':
        case 'g':
            filtro_gradiente();
            redesenha();
            break;

    }

}

void main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(512,512);
    glutInitWindowPosition(50,50);
    glutCreateWindow("Funcao");
    glutDisplayFunc(redesenha);
    gluOrtho2D(0,512,512,0);
    glutKeyboardFunc(le_tecla);
    glutMainLoop();
}

```

Linear.cpp

```

#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include "linear.h"

vector vecalloc (int dim)
{
    return (vector)malloc(sizeof(REAL)*dim);
}

matrix matalloc (int n, int m)
{
    int j;
    matrix M;
    M= (matrix)malloc (n * sizeof(vector));
    for (j=0; j<n;j++)
        M[j] = (vector) malloc (m * sizeof(REAL));
    return M;
}

void matfree (matrix mat, int m, int n)
{
    int i;

```

```

        for (i=0; i<n; i++)
            free (mat[i]);
        free (mat);
    }

void vecfree (vector v, int n)
{
    free(v);
}

```

plota.cpp

```

/* * * * * * * * * * * * * * */
/* Modulo: plota.cpp           */
/* * * * * * * * * * * * * * */

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

matrix A = NULL;
int      nx,ny;

/* ----- */
void le_imagem()
/* ----- */
{
    int      i,j;
    FILE     *fp;

    fp = fopen("lena.pgm","r");
    if (fp == NULL)
        printf("Arquivo nao encontrado\n");
    else {
        fscanf(fp,"%d %d",&nx,&ny);
        fscanf(fp,"%d",&i);
        A=matalloc(nx,ny);
        for(i=0;i<ny;i++) {
            for(j=0;j<nx;j++) {
                fscanf(fp,"%d",&(A[i][j]));
            }
        }
        fclose(fp);
    }
}

/* ----- */
void plota_imagem()
/* ----- */
{
    int i,j;

    if (A == NULL)
        return;

    for(i=0;i<ny;i++)
        for(j=0;j<nx;j++) {
            glColor3f(A[i][j]/255.0,A[i][j]/255.0,A[i][j]/255.0);
            glBegin(GL_POINTS);

```

```

        glVertex2d(j,i);
        glEnd();
    }
}

/* ----- */
void filtro_media()
/* ----- */
{
    int i,j;
    matrix B;

    if (A == NULL)
        return;

    B = malloc(ny,nx);
    for(i=0;i<ny;i++)
        for(j=0;j<nx;j++)
            B[i][j] = A[i][j];
    for(i=1;i<ny-1;i++)
        for(j=1;j<nx-1;j++) {
            A[i][j] = (B[i-1][j-1]+B[i-1][j]+B[i-1][j+1]+B[i][j-1]+B[i][j]+B[i][j+1]+B[i+1][j-1]+B[i+1][j]+B[i+1][j+1])/9.0;
        }

    matfree(B,nx,ny);
}

/* ----- */
void filtro_gradiente()
/* ----- */
{
    int i,j;
    matrix B,C,D;

    if (A == NULL)
        return;

    B = malloc(ny,nx);
    C = malloc(ny,nx);
    D = malloc(ny,nx);
    for(i=0;i<ny;i++)
        for(j=0;j<nx;j++)
            B[i][j] = A[i][j];
    for(i=1;i<ny-1;i++)
        for(j=1;j<nx-1;j++) {
            C[i][j] = (B[i-1][j-1] - B[i-1][j+1]
+2*B[i][j-1] -2*B[i][j+1] +B[i+1][j-1] -
B[i+1][j+1])/4.0;
            D[i][j] = (B[i-1][j-1]+2*B[i-1][j]+B[i-1][j+1]
-B[i+1][j-1]-2*B[i+1][j] -B[i+1][j+1])/4.0;

            A[i][j] = sqrt(D[i][j] * D[i][j] + C[i][j]*C[i][j]);
        }

    matfree(B,nx,ny);
    matfree(C,nx,ny);
    matfree(D,nx,ny);
}

```

7 – FRACTAIS

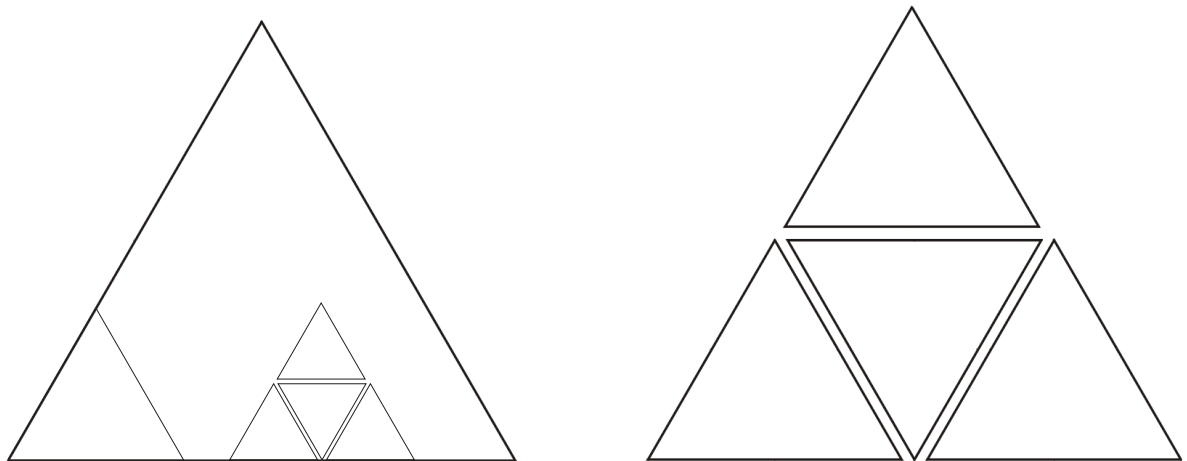
7.1 – Conjuntos auto semelhantes

Definição: Um subconjunto fechado e limitado $S \subset \mathbb{R}^2$, é dito ser auto-semelhante se pode ser expresso na forma:

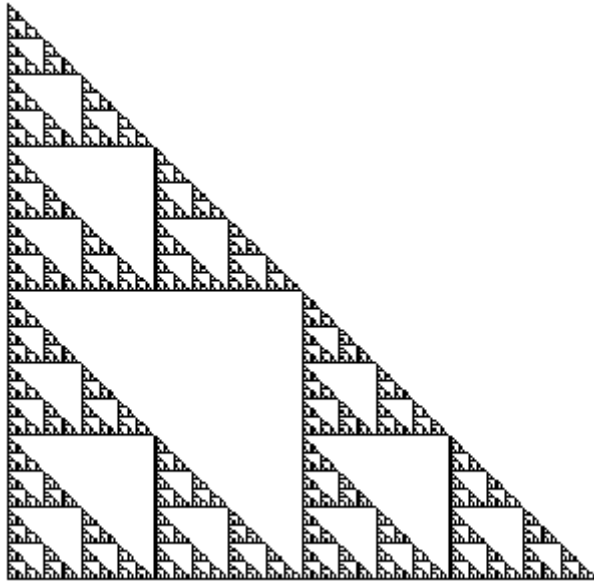
$$S = S_1 \cup S_2 \cup S_3 \cup \dots \cup S_k$$

onde $S_1, S_2, S_3, \dots, S_k$ são conjuntos não sobrepostos e cada um deles é congruente a S por mesmo fator de escala s ($0 < s < 1$).

Exemplo 1: Um triângulo pode ser expresso como a união de quatro triângulos congruentes e não sobrepostos. Cada um dos triângulos é congruente ao original por um fator $s = \frac{1}{2}$ e o triângulo é um conjunto auto-semelhante com $k = 4$.



Exemplo 2 (Triângulo de Sierpinski) : Este exemplo foi apresentado pelo matemático Waclaw Sierpinski (1882-1969). Neste exemplo, partindo de um triângulo, temos a união de três triângulos não sobrepostos (portanto $k = 3$), cada um dos quais é congruente ao original com um fator de escala $s = \frac{1}{2}$. Em seguida, o processo se repete para cada um dos três triângulos, e assim sucessivamente.



7.2 – Dimensão Hausdorff e o conceito de fractal

Definição: A dimensão Hausdorff de um conjunto auto-semelhante é definido por:

$$d_H(S) = \frac{\ln k}{\ln(1/s)}$$

onde $d_H(S)$ denota a dimensão Hausdorff.

Assim, considerando os exemplos anteriores teremos:

Exemplo 1: $d_H(S) = \frac{\ln(4)}{\ln(2)} = 2$

Exemplo 2: $d_H(S) = \frac{\ln(3)}{\ln(2)} = 1.585\dots$

Observe que no exemplo 1, a dimensão Hausdorff, coincide com a dimensão topológica usual, uma vez que uma região em \mathbb{R}^2 tem dimensão 2. Porém no exemplo 2, obtemos uma dimensão não inteira para o triângulo de Sierpinski. Partindo desta observação, Mandelbrot sugeriu em 1977 a seguinte definição:

Definição: Um fractal é um subconjunto do espaço euclidiano cuja dimensão Hausdorff é diferente da dimensão topológica.

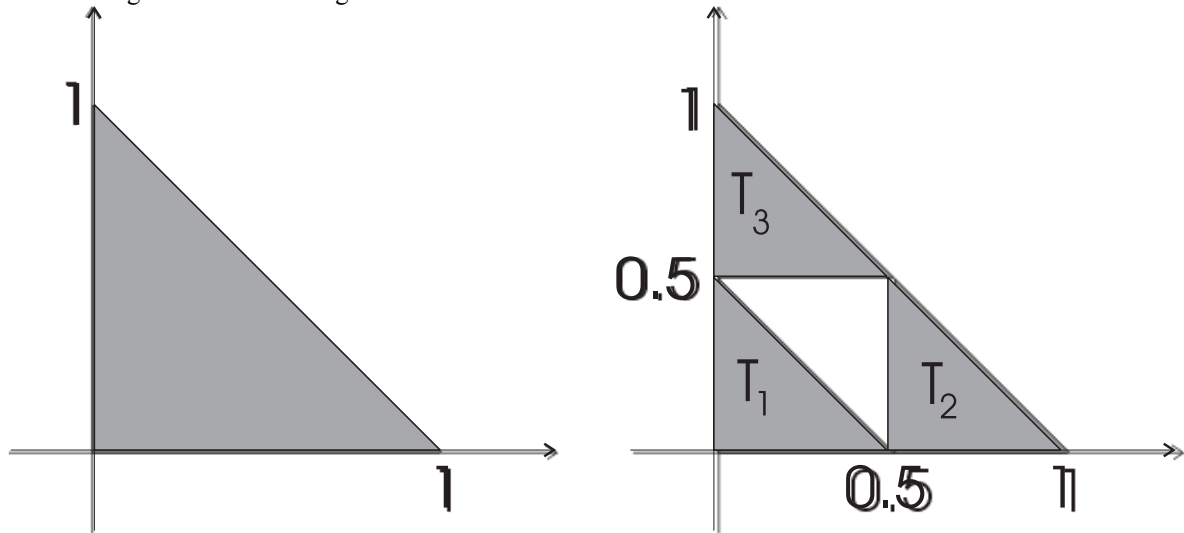
7.3 – Exemplos de fractais

Alguns dos principais exemplos que apresentaremos nesta seção foram gerados utilizando composição de transformações de rotação e escala seguido de uma possível translação. O formato geral da função é dado por:

$$T \begin{pmatrix} x \\ y \end{pmatrix} = s \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} \quad \text{onde } \begin{cases} s & \text{fator de escala} \\ \theta & \text{ângulo de rotação} \\ \begin{bmatrix} a \\ b \end{bmatrix} & \text{translação} \end{cases}$$

7.3.1- Triângulo de Sierpinski

Para obter o triângulo de Sierpinski, utilizaremos uma aplicação que a partir de um triângulo, obtém três novos triângulos conforme a figura abaixo:



Assim temos três funções, uma associada a cada triângulo:

$$T_1 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$T_2 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0 \end{bmatrix}$$

$$T_3 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}$$

O programa está dividido em dois módulos principais:

- `frac010.cpp`: Mantém as rotinas de visualização do OpenGL
- `frac011.cpp`: Mantém as rotinas relativas a implementação do triângulo de Sierpinski.

Temos ainda o módulo:

- `frac011.h`: Este módulo deve ter apenas a declaração das funções do programa `frac011.cpp`.

7.3.1.1- Módulo `frac010.cpp`

```
#include <gl\glut.h>
#include <stdio.h>
#include <math.h>
#include "fract011.h"

extern float xmax,xmin,ymax,ymin;
```

```

extern float incremento;
extern int pontos;

int xp[2],yp[2];
int tamx = 400;
int tamy = 400;
int st = -1;

void display()
{
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    plota_funcao(0.0,0.0,1.0,0.0,0.0,1.0,6);
    glFlush();
    glutSwapBuffers();
}

void le_tecle(unsigned char key, int x, int y)
{
    switch(key)
    {
        case 'I':
        case 'i':
            entra_dominio();
            break;
    }
}

void botao_mouse(int b,int state,int x, int y)
{
    float t;
    float xmin1,xmax1,ymin1,ymax1;

    switch(b) {
        case GLUT_LEFT_BUTTON:
            switch(state) {
                case GLUT_DOWN:
                    xp[0] = x;
                    yp[0] = y;
                    st = 0;
                    printf("down %d %d ",x,y);
                    break;
                case GLUT_UP:
                    xp[1] = x;
                    yp[1] = y;
                    printf("up %d %d ",x,y);
                    t = (float)xp[0] / tamx;
                    xmin1 = xmin + t * (xmax-xmin);
                    t = (float)xp[1] / tamx;
                    xmax1 = xmin + t * (xmax-xmin);
                    xmin = xmin1;
                    xmax = xmax1;
                    t = (float)yp[1] / tamy;
                    ymin1 = ymax - t * (ymax-ymin);
                    t = (float)yp[0] / tamy;
                    ymax1 = ymax - t * (ymax-ymin);
                    ymin = ymin1;
                    ymax = ymax1;
                    glLoadIdentity();
                    gluOrtho2D(xmin,xmax,ymin,ymax);
                    incremento = fabs(xmax-xmin)/pontos;
                    display();
            }
    }
}

```

```

                                break;
                                }
                                break;
        }
    }
}
void mov_mouse(int x, int y)
{
    float t;
    float xmin1,xmax1,ymin1,ymax1;

    t = (float)xp[0] / tamx;
    xmin1 = xmin + t * (xmax-xmin);
    t = (float)x      / tamx;
    xmax1 = xmin + t * (xmax-xmin);
    t = (float)y      / tamy;
    ymin1 = ymax - t * (ymax-ymin);
    t = (float)yp[0] / tamy;
    ymax1 = ymax - t * (ymax-ymin);
    display();
    glDrawBuffer(GL_FRONT);
    glColor3f(1.0,1.0,1.0);
    glBegin(GL_LINE_STRIP);
        glVertex2f(xmin1,ymin1);
        glVertex2f(xmin1,ymax1);
        glVertex2f(xmax1,ymax1);
        glVertex2f(xmax1,ymin1);
        glVertex2f(xmin1,ymin1);
    glEnd();
    glDrawBuffer(GL_BACK);
    glFlush();
}
void main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(tamx,tamy);
    glutInitWindowPosition(50,50);
    glutCreateWindow("Funcao");
    glutDisplayFunc(display);
    gluOrtho2D(xmin,xmax,ymin,ymax);
    glutKeyboardFunc(le_tecla);
    glutMouseFunc(botao_mouse);
    glutMotionFunc(mov_mouse);
    glutMainLoop();
}

```

7.3.1.2- Módulo frac011.cpp

```

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

float xmin = -2;
float ymin = -2;
float xmax = 2;
float ymax = 2;

float incremento = 0.01;

int pontos = 1000;

```



```

void funcao(float *x,float *y,float e,float f)
{
    *x = *x/2 + e;
    *y = *y/2 + f;
}

void plota_eixo()
{
    glColor3f(0.0,1.0,0.0);
    glBegin(GL_LINES);
        glVertex2f(xmin,0);
        glVertex2f(xmax,0);
        glVertex2f(0,ymin);
        glVertex2f(0,ymax);
    glEnd();
}

void plota_funcao(float x0,float y0,
                  float x1,float y1,
                  float x2,float y2,
                  int n)
{
    int i,j;
    float x[3][3],y[3][3];
    float e[3] = {0.0, 0.5, 0.0};
    float f[3] = {0.0, 0.0, 0.5};

    for(i=0;i<3;i++) {
        x[i][0] = x0;
        y[i][0] = y0;
        x[i][1] = x1;
        y[i][1] = y1;
        x[i][2] = x2;
        y[i][2] = y2;
    }

    for(i=0;i<3;i++)
        for (j=0;j<3;j++)
            funcao(&(x[j][i]),&(y[j][i]),e[j],f[j]);

    if (n == 0) {
        for (i=0;i<3;i++) {
            glColor3f(1.0,0.0,0.0);
            glBegin(GL_POLYGON);
                glVertex2f(x[i][0],y[i][0]);
                glVertex2f(x[i][1],y[i][1]);
                glVertex2f(x[i][2],y[i][2]);
            glEnd();
        }
        return;
    }
    else {
        for(i=0;i<3;i++)
            plota_funcao(x[i][0],y[i][0],x[i][1],y[i][1],
                        x[i][2],y[i][2],n-1);
    }
}

void entra_dominio()
{

```

```

    printf("xmin = ");
    scanf("%f",&xmin);
    printf("\nymin = ");
    scanf("%f",&ymin);
    printf("\nxmax = ");
    scanf("%f",&xmax);
    printf("\nymax = ");
    scanf("%f",&ymax);
    printf("\n Total de pontos =");
    scanf("%d",&pontos);

    glLoadIdentity();
    gluOrtho2D(xmin,xmax,ymin,ymax);

    incremento = fabs(xmax-xmin)/pontos;
}

```

7.3.1.2- Módulo **frac011.h**

```

void funcao(float *x,float *y,float e,float f);

void plota_eixo();

void plota_funcao(float x0,float y0,float x1,float y1,
                  float x2,float y2,int n);

void entra_dominio();

```

7.3.2- Triângulo de Sierpinski utilizando Monte Carlo

Este método utiliza iterações randômicas para gerar fractais utilizando o seguinte processo:

1- Defina as k transformações T_k (como descrito na seção 6.3) que descrevem o objeto a ser gerado.

2- Escolha um ponto arbitrário $\begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$.

3- Escolha arbitrariamente uma das k transformações e aplique no ponto escolhido:

$$T\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

4- Prossiga escolhendo aleatoriamente uma das k transformações e aplique no ultimo ponto obtido:

$$T\begin{pmatrix} x_{n-1} \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$

O programa proposto está dividido em dois módulos principais:

- **frac020.cpp**: Mantém as rotinas de visualização do OpenGL
- **frac021.cpp**: Mantém as rotinas relativas a implementação do triângulo de Sierpinski.

Temos ainda o módulo:

- **frac021.h**: Este módulo deve ter apenas a declaração das funções do programa **frac021.cpp**.

7.3.2.1- Módulo **frac020.cpp**

Este módulo é idêntico ao módulo frac010.cpp, com apenas duas linhas de alteração como descrito abaixo:

```
⋮  
#include <math.h>  
#include "frac021.h"  
⋮  
void display()  
{  
    ⋮  
    plota_funcao(1.5,-1.8);  
    ⋮  
}  
⋮
```

7.3.2.2- Módulo **frac021.cpp**

```
/* * * * * * * * * * * * * * * * */  
/* Triangulo Sierpinski Monte Carlo */  
/* frac021.cpp                      */  
  
#include <stdio.h>  
#include <math.h>  
#include <time.h>  
#include <gl\glut.h>  
  
float xmin = -2;  
float ymin = -2;  
float xmax = 2;  
float ymax = 2;  
  
float incremento = 0.01;  
  
int pontos = 1000;  
  
void funcao(float *x,float *y,float e,float f)  
{  
    *x = *x/2 + e;  
    *y = *y/2 + f;  
}  
  
void plota_eixo()  
{  
    glColor3f(0.0,1.0,0.0);  
    glBegin(GL_LINES);  
        glVertex2f(xmin,0);  
        glVertex2f(xmax,0);  
        glVertex2f(0,ymin);  
        glVertex2f(0,ymax);  
    glEnd();  
}
```

```

        glEnd();
    }

void plota_funcao(float x0, float y0)
{
    int i, j;
    float x[3][4], y[3][4];
    float e[3] = {0.0, 0.5, 0.0};
    float f[3] = {0.0, 0.0, 0.5};

    srand( (unsigned)time( NULL ) );

    for(i=0; i<20000; i++) {
        j = (int) (3.0 * ((float)rand()) / RAND_MAX);
        j = ( j > 2 ) ? 2 : j;
        funcao(&x0, &y0, e[j], f[j]);
        glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_POINTS);
            glVertex2f(x0, y0);
        glEnd();
    }
}

void entra_dominio()
{
    printf("xmin = ");
    scanf("%f", &xmin);
    printf("\nymin = ");
    scanf("%f", &ymin);
    printf("\nxmax = ");
    scanf("%f", &xmax);
    printf("\nymax = ");
    scanf("%f", &ymax);
    printf("\n Total de pontos =");
    scanf("%d", &pontos);

    glLoadIdentity();
    gluOrtho2D(xmin, xmax, ymin, ymax);

    incremento = fabs(xmax-xmin)/pontos;
}

```

7.3.3- “Fern” utilizando Monte Carlo

7.3.3.1- Módulo `frac030.cpp`

```

:
:
#include "fract031.h"
:
:
void display()
{
    int i;
    float x[4] = {0.0, 1.0, 1.0, 0.0};
    float y[4] = {0.0, 0.0, 1.0, 1.0};
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
}

```

```

        plota_funcao(x,y,8);
        glFlush();
    }
    :

```

7.3.3.2- Módulo frac031.cpp

```

/* * * * * * * * * * * */
/* Fractal Fern          */
/* fract031.cpp          */
/* * * * * * * * * * * */

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

float xmin = -0.2;
float ymin = -0.2;
float xmax = 2;
float ymax = 2;

float incremento = 0.01;

int pontos = 1000;

void T(float a11,float a12,float a21,float a22,float e,float f,float
*x1,float *y1)
{
    float xx;
    float yy;

    xx = a11 * *x1 + a12 * *y1 + e;
    yy = a21 * *x1 + a22 * *y1 + f;
    *x1 = xx;
    *y1 = yy;
}

void plota_funcao(float *x,float *y,int n)
{
    int i,j;
    float xx[4][4],yy[4][4];

    for(i=0;i<4;i++)
        for(j=0;j<4;j++)
        {
            xx[i][j] = x[j];
            yy[i][j] = y[j];
        }

    if (n == 0) {
        glColor3f(0.0,1.0,0.0);
        glBegin(GL_LINE_LOOP);
            glVertex2f(x[0],y[0]);
            glVertex2f(x[1],y[1]);
            glVertex2f(x[2],y[2]);
            glVertex2f(x[3],y[3]);
        glEnd();
        return;
    }
}

```

```

        else {
            for(i=0;i<4;i++) {
                T( 0.20,-0.26, 0.23, 0.22,0.400,
0.045,&(xx[0][i]),&(yy[0][i]));
                T( 0.85, 0.04,-0.04, 0.85,0.075,
0.180,&(xx[1][i]),&(yy[1][i]));
                T( 0.00, 0.00, 0.00, 0.16,0.500,
0.000,&(xx[2][i]),&(yy[2][i]));
                T(-0.15, 0.28, 0.26, 0.24,0.575,-
0.086,&(xx[3][i]),&(yy[3][i]));
            }
            plota_funcao(xx[0],yy[0],n-1);
            plota_funcao(xx[1],yy[1],n-1);
            plota_funcao(xx[2],yy[2],n-1);
            plota_funcao(xx[3],yy[3],n-1);
        }
    }

void entra_dominio()
{
    printf("xmin = ");
    scanf("%f",&xmin);
    printf("\nymin = ");
    scanf("%f",&ymin);
    printf("\nxmax = ");
    scanf("%f",&xmax);
    printf("\nymax = ");
    scanf("%f",&ymax);
    printf("\n Total de pontos =");
    scanf("%d",&pontos);

    glLoadIdentity();
    gluOrtho2D(xmin,xmax,ymin,ymax);

    incremento = fabs(xmax-xmin)/pontos;
}

```

7.3.4- Curva de Koch

7.3.4.1- Módulo **frac040.cpp**

```

#include <gl\glut.h>
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include "fract041.h"

extern float xmax,xmin,ymax,ymin;
extern float incremento;
extern int pontos;

int xp[2],yp[2];
int tamx = 400;
int tamy = 400;
int st = -1;

void display()
{
    int i;

```

```

        glClearColor(0.0,0.0,0.0,0.0);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
        plota_eixo();
        for(i=0;i<7;i++) {
            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
            plota_funcao(-1.0,1.0,1.0,1.0,i);
            plota_funcao( 1.0,1.0,0.0,-1.0,i);
            plota_funcao(0.0,-1.0,-1.0,1.0,i);
            glFlush();
            glutSwapBuffers();
            getch();
        }
        glFlush();
        glutSwapBuffers();
    }

void le_tecle(unsigned char key, int x, int y)
{
    switch(key)
    {
        case 'I':
        case 'i':
            entra_dominio();
            break;
    }
}

void botao_mouse(int b,int state,int x, int y)
{
    float t;
    float xmin1,xmax1,ymin1,ymax1;

    switch(b) {
        case GLUT_LEFT_BUTTON:
            switch(state) {
                case GLUT_DOWN:
                    xp[0] = x;
                    yp[0] = y;
                    st = 0;
                    printf("down %d %d ",x,y);
                    break;
                case GLUT_UP:
                    xp[1] = x;
                    yp[1] = y;
                    printf("up %d %d ",x,y);
                    t = (float)xp[0] / tamx;
                    xmin1 = xmin + t * (xmax-xmin);
                    t = (float)xp[1] / tamx;
                    xmax1 = xmin + t * (xmax-xmin);
                    xmin = xmin1;
                    xmax = xmax1;
                    t = (float)yp[1] / tamy;
                    ymin1 = ymax - t * (ymax-ymin);
                    t = (float)yp[0] / tamy;
                    ymax1 = ymax - t * (ymax-ymin);
                    ymin = ymin1;
                    ymax = ymax1;
                    glLoadIdentity();
                    gluOrtho2D(xmin,xmax,ymin,ymax);
                    incremento = fabs(xmax-xmin)/pontos;
                    display();
                    break;
            }
    }
}

```

```

        }
        break;
    }
}

void mov_mouse(int x, int y)
{
    float t;
    float xmin1,xmax1,ymin1,ymax1;

    t = (float)xp[0] / tamx;
    xmin1 = xmin + t * (xmax-xmin);
    t = (float)x / tamx;
    xmax1 = xmin + t * (xmax-xmin);
    t = (float)y / tamy;
    ymin1 = ymax - t * (ymax-ymin);
    t = (float)yp[0] / tamy;
    ymax1 = ymax - t * (ymax-ymin);
    display();
    glDrawBuffer(GL_FRONT);
    glColor3f(1.0,1.0,1.0);
    glBegin(GL_LINE_STRIP);
        glVertex2f(xmin1,ymin1);
        glVertex2f(xmin1,ymax1);
        glVertex2f(xmax1,ymax1);
        glVertex2f(xmax1,ymin1);
        glVertex2f(xmin1,ymin1);
    glEnd();
    glDrawBuffer(GL_BACK);
    glFlush();
}

void main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(tamx,tamy);
    glutInitWindowPosition(50,50);
    glutCreateWindow("Funcao");
    glutDisplayFunc(display);
    gluOrtho2D(xmin,xmax,ymin,ymax);
    glutKeyboardFunc(le_tecla);
    glutMouseFunc(botao_mouse);
    glutMotionFunc(mov_mouse);
    glutMainLoop();
}

```

7.3.4.2- Módulo frac041.cpp

```

/* * * * * * * * * * * */
/* Fractal Curva Koch */
/* fract041.cpp */

#include <stdio.h>
#include <math.h>
#include <time.h>
#include <gl\glut.h>

float xmin = -2;
float ymin = -2;
float xmax = 2;

```



```

float ymax = 2;

float incremento = 0.01;

int pontos = 1000;

void plota_eixo()
{
    glColor3f(0.0,1.0,0.0);
    glBegin(GL_LINES);
        glVertex2f(xmin,0);
        glVertex2f(xmax,0);
        glVertex2f(0,ymin);
        glVertex2f(0,ymax);
    glEnd();
}

void plota_funcao(float x0,float y0,float x1,float y1,int n)
{
    int i,j;
    float v[2];
    float xx0,yy0,xx1,yy1;

    xx0 = x0+1.0/3*(x1 - x0);
    yy0 = y0+1.0/3*(y1 - y0);
    xx1 = x0+2.0/3*(x1 - x0);
    yy1 = y0+2.0/3*(y1 - y0);
    if (n == 0) {
        glColor3f(1.0,1.0,0.0);
        glBegin(GL_LINES);
            glVertex2f(x0,y0);
            glVertex2f(x1,y1);
        glEnd();
    }
    else {
        v[0]=(y0-y1)/(2*sqrt(3.0));
        v[1]=(x1-x0)/(2*sqrt(3.0));
        plota_funcao(x0,y0,xx0,yy0,n-1);
        plota_funcao(xx0,yy0,(x0+x1)/2.0+v[0],(y0+y1)/2.0+v[1],n-1);
        plota_funcao((x0+x1)/2.0+v[0],(y0+y1)/2.0+v[1],xx1,yy1,n-1);
        plota_funcao(xx1,yy1,x1,y1,n-1);
    }
}

void entra_dominio()
{
    printf("xmin = ");
    scanf("%f",&xmin);
    printf("\nymin = ");
    scanf("%f",&ymin);
    printf("\nxmax = ");
    scanf("%f",&xmax);
    printf("\nymax = ");
    scanf("%f",&ymax);
    printf("\n Total de pontos =");
    scanf("%d",&pontos);

    glLoadIdentity();
    gluOrtho2D(xmin,xmax,ymin,ymax);
}

```

```
    incremento = fabs(xmax-xmin)/pontos;  
}
```

<u>CAPÍTULO III – VISUALIZAÇÃO E APLICAÇÕES GRÁFICAS 3D</u>	2
1- TRANSFORMAÇÕES DE VISUALIZAÇÃO	2
1.1 - Comandos de Auxílio	2
1.2- Exemplo: cubo unitário	3
1.3 - Transformações de Modelagem e Visualização	4
1.3.1- Translação	4
1.3.2- Rotação	4
1.3.3- Escala	4
1.3.4-Exemplo	5
1.4- Projeção Ortográfica	6
1.5- Ângulos de Euler	6
1.6 - Criando um Ambiente de Visualização 3D	7
1.6.1- Módulo Básico de Visualização	7
1.6.2- Alterando os ângulos de Euler	8
1.7 – Visualização de gráfico de funções $z = f(x, y)$	10
1.7.1 – Módulo funcao3D010.cpp	10
1.7.2 - Módulo funcao3D011.cpp	12
1.7.3 - Módulo funcao3D010.h	13
2- ILUMINAÇÃO	14
2.1– Criando Fontes de Luz	14
2.1.1– Cor	14
2.1.2– Posição	15
2.2– Selecionando o Modelo de Iluminação	15
2.2.1- Luz Ambiente Global	15
2.2.2 – Posição do observador local ou no infinito	15
2.2.3 – Iluminação nos dois lados das faces	15
2.2.4 – Habilitando a iluminação	15
2.3– Selecionando as Propriedades do Material	16
2.4– Exemplo	16
2.4.1- Módulo ilumina_main.cpp	16
2.4.2 – Módulo ilumina_funcao.cpp	20
2.4.3- Módulo ilumina_set.cpp	22
2.4.4 – Módulo ilumina_funcao.h	23
2.4.5 – Módulo ilumina_set.h	23
3- SUPERFÍCIES PARAMETRIZADAS	24
3.1– Visualização de superfícies na forma paramétrica	24
3.2– Exercícios	30
4- INTERPOLAÇÃO	31
4.1– Interpolação Utilizando o Método de Shepard	31
4.2– Propriedades do Método de Shepard	31
4.3– Programa Interpolação Método de Shepard	32

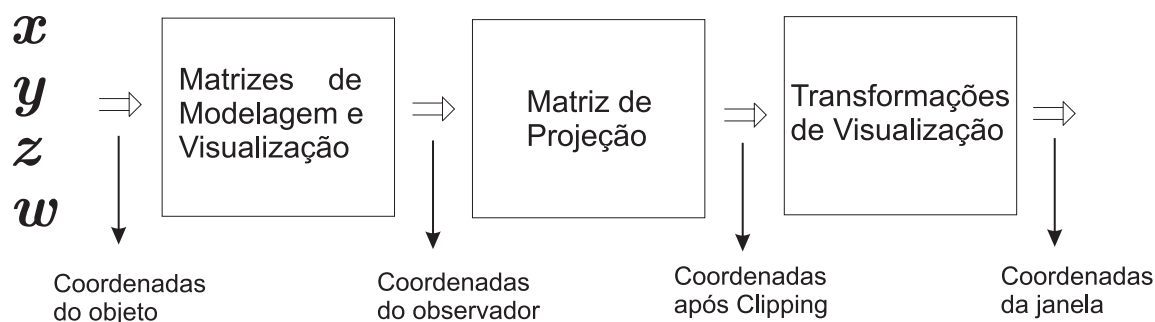
CAPÍTULO III – VISUALIZAÇÃO E APLICAÇÕES GRÁFICAS 3D

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.

```
GLoadIdentity(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 θ 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, desenhmos 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

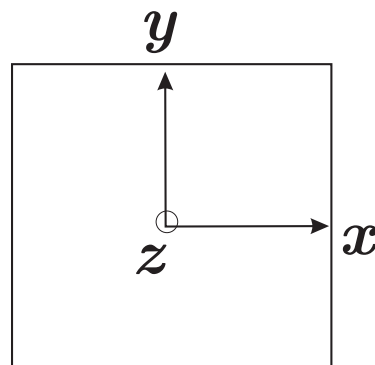
Em uma projeção ortográfica, nós estabelecemos um volume de visualização (que corresponde a um paralelepípedo retângulo). O objeto só será visualizado se ele estiver contido neste volume. O comando utilizado é:

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

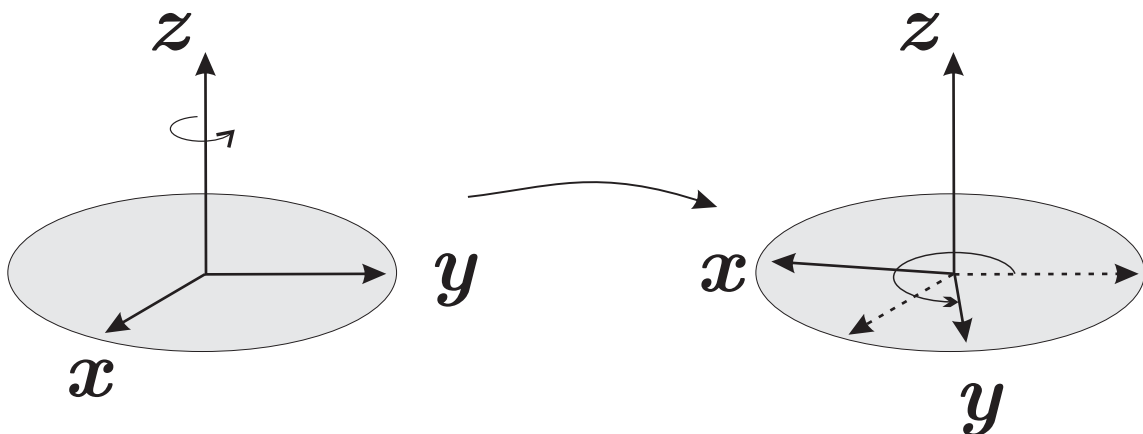
1.5- Ângulos de Euler

Um sistema de referência que irá nos auxiliar na montagem do ambiente 3D são os ângulos de Euler. Os Ângulos de Euler são definidos como três sucessivos ângulos de rotação através dos quais definiremos um novo sistema de coordenadas partindo das coordenadas cartesianas.

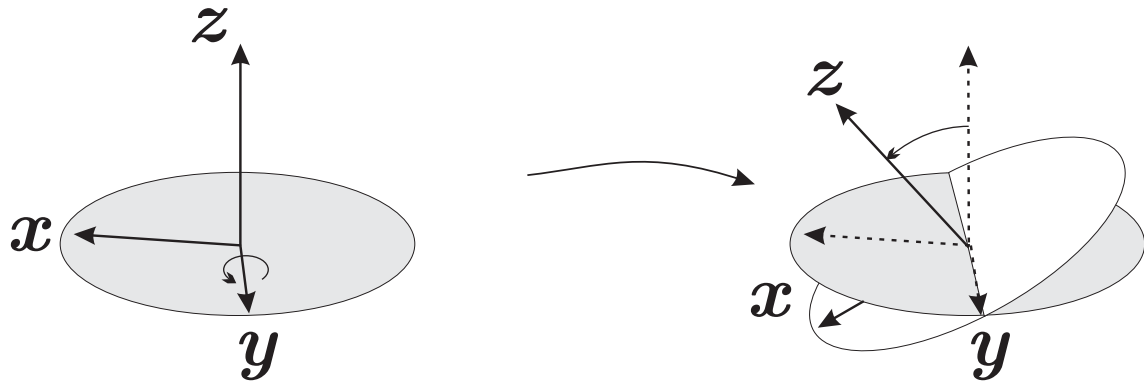
No OpenGL o sistema de coordenadas está posicionado conforme o desenho abaixo, onde o eixo x está na posição horizontal, o eixo y na posição vertical e o eixo z está apontando para fora da tela do computador.



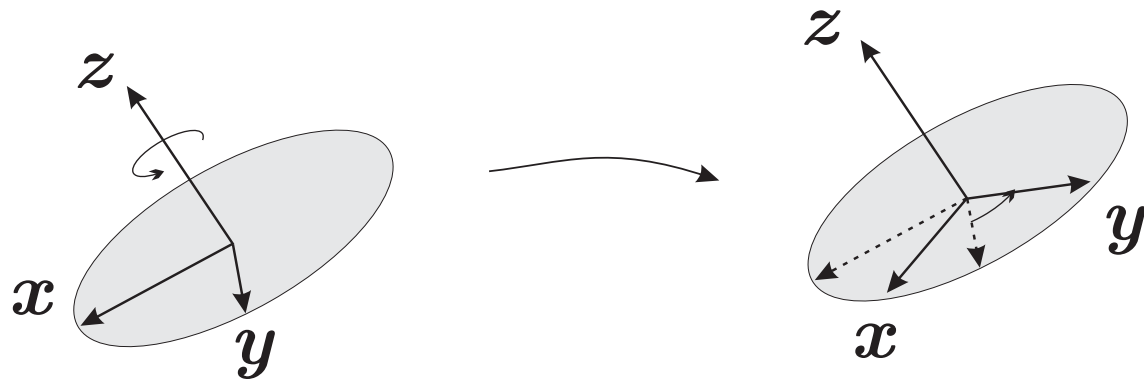
A sequência inicia rotacionando o sistema de coordenadas xyz por um ângulo θ no sentido anti-horário em torno do eixo z.



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



Por fim o sistema de coordenadas sofre uma nova rotação de ψ 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);
        glVertex3f(0.0,0.0,0.0);
    glEnd();
}
```

```

        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 $f: \mathbb{R}^2 \rightarrow \mathbb{R}$. 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 botoa_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_tecle);
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, GLfloat 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, luz_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 o 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 está 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_tecle);
    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

```

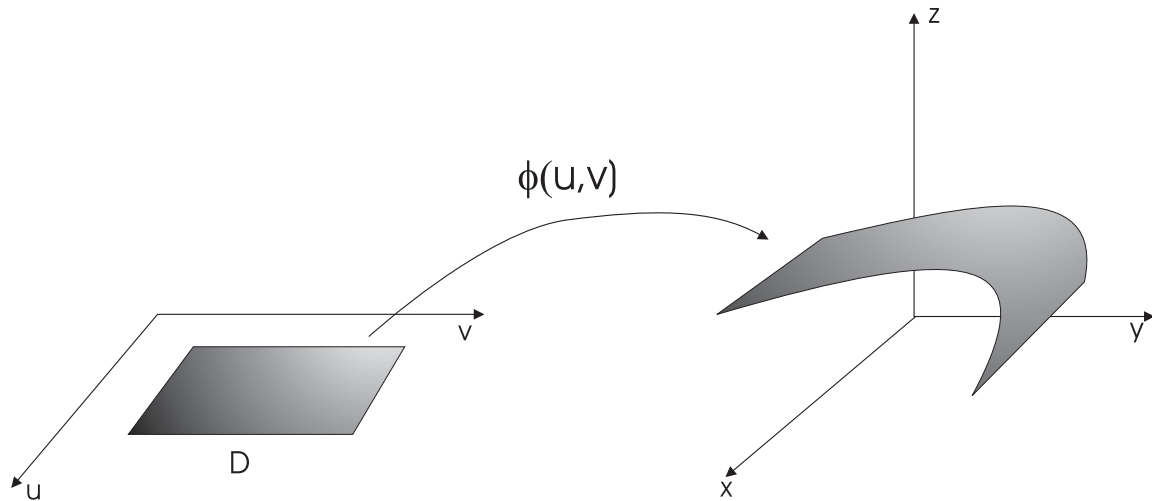
void desabilita_lighting();
void habilita_lighting();
void le_tecla(unsigned char key, int x, int y);

```

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 tridimensional, 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 $\phi: D \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$ onde D é algum domínio em \mathbb{R}^2 . A superfície S correspondente a função ϕ 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.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/*      Superfícies 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_teclela);
        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) $\begin{cases} x = \sin(\varphi) \cos(\theta) \\ y = \sin(\varphi) \sin(\theta) \\ z = \cos(\varphi) \end{cases}$ $0 \leq \varphi \leq \pi$ $0 \leq \theta \leq 2\pi$	b) $\begin{cases} x = 2u \cos(\theta) \\ y = 2u \sin(\theta) \\ z = 2u \end{cases}$ $0 \leq u \leq 4$ $0 \leq \theta \leq 2\pi$	c) $\begin{cases} x = 2u \cos(\theta) \\ y = 2u \sin(\theta) \\ z = 4u^2 \end{cases}$ $0 \leq u \leq 1$ $0 \leq \theta \leq 2\pi$	d) $\begin{cases} x = \cos(\theta) \\ y = \sin(\theta) \\ z = u \end{cases}$ $0 \leq u \leq 4$ $0 \leq \theta \leq 2\pi$
e) $\begin{cases} x = 2u \cos(\theta) \\ y = 2u \sin(\theta) \\ z = 2u \end{cases}$ $0 \leq u \leq 4$ $0 \leq \theta \leq 2\pi$	f) $\begin{cases} x = \cos(u) + v \cos(\frac{u}{2}) \cos(u) \\ y = \sin(u) + v \cos(\frac{u}{2}) \sin(u) \\ z = v \sin(\frac{u}{2}) \end{cases}$ $-0.2 \leq v \leq 0.2$ $0 \leq u \leq 2\pi$	g) $\begin{cases} x = u \\ y = v \\ z = \sqrt{u^2 + v^2} \end{cases}$ $0 \leq u \leq 1$ $0 \leq v \leq 1$	

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:

$$f(x_i, y_i) = z_i \quad i = 0, \dots, n-1$$

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

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

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

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

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();
}

```