VISUALIZADOR DE FECHO CONVEXO TRIDIMENSIONAL

Relatório de Projeto

Luciano Silva

12 de dezembro de 1997

Sumário

1.1 Introdução 2
1.2 Ray-tracing 2
1.3 Estrutura do visualizador 3
1.4 Estruturas de dados da cena 3
1.5 Lançamento de raios 7
1.6 Cálculo de intersecções com as faces 10
1.7 Inicialização dos parâmetros de cena 14
1.8 Arquivos de cabeçalhos auxiliares 16
  1.8.1 Definições comuns aos módulos 16
  1.8.2 Macros de vetores 17
1.9 Formato do arquivo de entrada 18

1.1 Introdução

O objetivo deste projeto é a implementação de um sistema de visualização tridimensional para objetos triangularizados. Como aplicação imediata, tem-se um sistema para visualização da saída de um algoritmo para fechos convexos tridimensionais, baseado em embrulho-pra-presente.

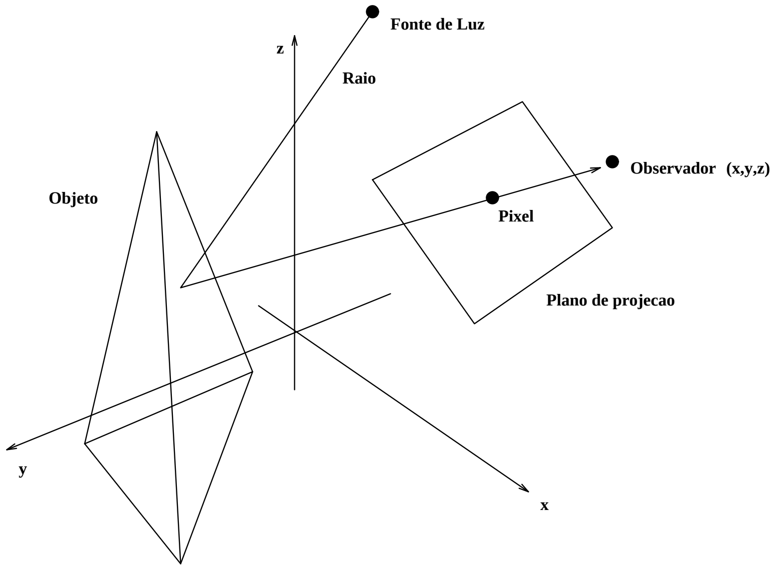
Para a visualização, adotou-se uma implementação tipo traçado de raios (*ray-tracing* [1] [4]) pelos seguintes motivos:

- estruturas de dados simples
- cálculos de interseções no contínuo, evitando certos problemas com projeções
- fácil extensão para faces não-triangulares
- fácil adaptação para super-amostragem de raios, atenuando os efeitos de aliasing

Existem vários outros algoritmos de visualização de objetos poligonais, tais como os bem conhecidos *scan-line com z-buffer* e *depth-sort* [2] [3]. Apesar de serem, geralmente, mais rápidos que a técnica de *ray-tracing*, exigem um pré-processamento custo de projeções e algumas estruturas de dados complexas.

1.2 Ray-tracing

A técnica de traçado de raios (*ray-tracing*) baseia-se no traçado inverso do caminho da luz emitida por um ponto numa cena tridimensional e que atinge o olho do observador:



O plano de projeção representa, na implementação, a janela onde será mostrada a cena e é discreto(os pontos são os pixels da tela).

A idéia do algoritmo é bem simples:

```
1. Para linha i do plano de projeção
2.  Para cada coluna j do plano de projeção
3.      Lançar um raio do observador passando por (i,j)
4.      Para cada objeto em cena
5.        Calcular a intersecção raio-objeto
6.      Se ocorreu alguma intersecção
7.        Pinte a cor do pixel (i,j) com a cor do objeto
          que produziu a interseccção mais próxima do
          usuário
8.      Senão
9.        Pinte a cor do pixel (i,j) com a cor do fundo
          especificado para a cena
```

O complexidade deste algoritmo é :

$$O(n^2)\sum_{i=1}^{q} c_{IntersecObj_i}$$

onde n é a resolução do plano de projeção e $c_{IntersecObj_i}$ é o custo de intersecção $raio - objeto_i$. Existem várias técnicas para redução desta complexidade quadrática, baseadas na exploração de coerência entre os raios e a disposição dos objetos em cena. O projeto do visualizador não implementa esta redução.

1.3 Estrutura do visualizador

O projeto do visualizador foi implementado numa máquina Macintosh Quadra 605, na linguagem C, utilizando o ambiente de desenvolvimento Metrowerks CodeWarrior.

O visualizador é composto de 5 módulos:

triangulo.c	Rotinas de cálculo de alocação das faces triangulares e cálculos de intersecção.
listas.c	Manipulação da lista de objetos da cena.
raios.c	Rotinas de lançamento de raios na cena.
janelas.c	Rotinas de manipulação de janelas de exibição e desenho de ponto.
main.c	Leitura de dados das faces, inicialização dos parâmetros descritores da cena (posição do observador, tamanho do plano de projeção, etc.) e chamada da função de visualização.

A menos do módulo **janelas.c**, todos os outros módulos foram escritos em C padrão, promovendo pouca dificuldade para migração para outros sistemas. Para portar o programa para outros sistemas, basta reescrever os métodos gráficos deste módulo.

1.4 Estruturas de dados da cena

As estruturas de dados são definidas pelo arquivo *estrutura.h*, mostrado a seguir:

```
/***********************************************************************************

  PROJETO  :    VISUALIZADOR DE FECHO CONVEXO TRIDIMENSIONAL

  HEADER   :    ESTRUTURA.H

  OBJETIVO :    DEFINICAO DAS ESTRUTURAS DE DADOS BASICAS DO VISUALIZADOR

 ***********************************************************************************/


#pragma once


/*************** ESTRUTURAS PARA CARACTERISTICAS GERAIS DA CENA *********************/

typedef struct{ /* VETOR TRIDIMENSIONAL */
    long double x,   /* COORDENADAS DO VETOR */
           y,
           z;
} VETOR;

typedef struct{ /* ESTRUTURA PARA COR */
     long double r,  /* COMPONENTES R,G,B DA COR */
                 g,
                 b;
} COR;

typedef struct{ /* ESTRUTURA PARA RAIO */
     VETOR origem,
           direcao;
} RAIO;

typedef struct{  /* ESTRUTURA PARA CARACTERISTICAS DO OBSERVADOR */
     VETOR localizacao,   /* LOCALIZACAO DO OBSERVADOR */
           direcao,       /* FOCO DO OBSERVADOR */
           view_cima,     /* VETORES DIRETORES DA JANELA DE OBSERVACAO */
           view_direita;
} OBSERVADOR;


typedef struct{  /* ESTRUTURA PARA DIMENSOES DO DISPLAY */
     unsigned comprimento,
              altura;
} DISPLAY;


/************** ESTRUTURAS PARA DEFINICAO DOS OBJETOS *******************************/

 typedef enum {      /*  TIPO PARA AS FACES DO FECHO */
         TRIANGULO /* OBJETO TRINAGULO */

 } TIPO_OBJETO;

 struct OBJETO_TRIANGULO{   /* GEOMETRIA DO OBJETO TRIANGULO */
        VETOR v1,
              v2,
```

```
            v3;
        VETOR normal;      /* PLANO CONTENDO O TRIANGULO */
        long double distancia; /* DISTANCIA DA ORIGEM */
};

typedef struct NO_OBJETO{   /* DEFINICAO GENERICA DE UM NO' DA LISTA LIGADA DE OBJETOS */
     struct NO_OBJETO *proximo;   /* PROXIMO OBJETO */
     union{
           struct OBJETO_TRIANGULO triangulo;
           } geom_obj;  /* GEOMETRIA DO OBJETO */
     /* METODOS GERAIS PARA OS OBJETOS */
     long double (*interseccao)(struct NO_OBJETO *,RAIO *); /* INTERSECCAO COM OBJETO */
     COR cor;  /* PROPRIEDADES FISICAS DA SUPERFICIE DO OBJETO */
 } OBJETO;
```

Cada objeto presente na cena é representado por um nó de uma lista ligada simples, contendo:

- um apontador para o próximo objeto na cena
- o tipo de objeto (triangular, quadrado, etc.)
- um método para cálculo de intersecção raio-objeto
- um atributo de cor para o objeto, no padrão cubo RGB [0-1]

As funções de manipulação da lista ligada são definidas no arquivo *listas.c*, listada a seguir:

```
/*************************************************************************************

  PROJETO  :    VISUALIZADOR DE FECHO CONVEXO TRIDIMENSIONAL

  MODULO   :    LISTAS.C

  OBJETIVO :    FUNCOES DE MANIPULACAO DA LISTA DE OBJETOS

*************************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "estrutura.h"
#include "listas.h"

OBJETO *lista_objetos;  /* APONTADOR PARA INICIO DA LISTA DE OBJETOS */
int NUM_OBJETOS=0;           /* NUMERO DE OBJETOS NA LISTA                */


void adiciona_objeto(OBJETO **lista,OBJETO *item)
/*************************************************************************************

  FUNCAO    : adiciona_objeto.

  OBJETIVO  : adiciona um objeto `a lista de objetos.
```

```
  ENTRADA   : lista- apontador para lista de objetos
              item - apontador para objeto especifico.

  SAIDA     : nenhuma.

  AUTOR     : Luciano Silva

  DATA      : 02.11.97

  REVISAO   :                            REVISADO POR :

 *************************************************************************************/
{
 if(*lista==NULL){
    *lista=item;
    item->proximo=NULL;
 }
 else{
      item->proximo=*lista;
      *lista=item;
      }
 return;
}



void lista_livre(void)
/*************************************************************************************

  FUNCAO    : lista_livre.

  OBJETIVO  : libera toda a memoria ocupada pela lista de objetos.

  ENTRADA   : nenhuma.

  SAIDA     : nenhuma.

  AUTOR     : Luciano Silva

  DATA      : 02.11.97

  REVISAO   :                            REVISADO POR :

 *************************************************************************************/
{
  OBJETO *lista_aux;     /* APONTADOR AUXILIAR PARA PERCORRER A LISTA */

  while(lista_objetos){  /* PERCORRE TODA A LISTA DESALOCANDO OS OBJETOS */
    lista_aux=lista_objetos;
    lista_objetos=lista_objetos->proximo;
    free(lista_aux);
  }
  NUM_OBJETOS=0;
  lista_objetos=NULL;
}
```

Nesta implementação, foram considerados somente objetos triangulares, podendo ser facilmente expandida para outros objetos.

As outras estruturas são bem mais simples e representam os raios lançados na cena, as características do relacionadas aos observador(posição, ponto focado e orientação do plano de projeção) e estruturas auxiliares, como vetores e cores.

1.5 Lançamento de raios

Para se lançar raios na cena, primeiramente precisa-se produzir um raio. Para isto, são necessários a posição do observador, o ponto (i, j) observado no plano de projeção e orientação do plano de projeção.

Após produzido o raio, passa-se à tarefa de intersecção deste raios com os objetos em cena, para descobrir a intersecção mais próxima do observador, se houver.

Para se mostrar o que realmente é visto, precisa-se realizar as tarefas acima para todo o ponto (i, j) do plano de projeção.

Todas estas tarefas são implementadas pelo módulo *raios.c*, descrito a seguir:

```
/************************************************************************************

   PROJETO  :    VISUALIZADOR DE FECHO CONVEXO TRIDIMENSIONAL

   MODULO   :    RAIOS.C

   OBJETIVO :    FUNCOES DE LANCAMENTO DE RAIOS E CALCULO DE ILUMINACAO

 ************************************************************************************/

#include <stdio.h>
#include <math.h>
#include "janelas.h"
#include "comum.h"
#include "vetores.h"
#include "estrutura.h"
#include "raios.h"

extern OBJETO *lista_objetos;      /* LISTA LIGADA DE OBJETOS       */
DISPLAY tela;                      /* CARACTERISTICAS DA TELA       */
COR fundo;                         /* COR DE FUNDO DA TELA          */
OBSERVADOR observador;             /* OBSERVADOR DA CENA            */

long double temp;                  /* VARIAVEL USADA PELAS MACROS DE VETORES  */

void lanca_raio_cena(RAIO *raio,COR *cor){
/************************************************************************************

   FUNCAO    : lanca_raio_cena.

   OBJETIVO  : lanca um raio na cena composta pelos objetos.

   ENTRADA   : raio - raio
               cor  - apontador para nova cor do ponto

   SAIDA     : cor - nova cor do ponto
```

```
   AUTOR     : Luciano Silva

   DATA      : 03.11.97

   REVISAO   :                             REVISADO POR :

  ************************************************************************************/
 long double interseccao,   /* PARAMETRO DE INTERSECCAO */
        interseccao_menor;  /* PARAMETRO DA INTERSECCAO MAIS PROXIMA */

 OBJETO *obj_mais_proximo,  /* OBJETO CUJA INTERSECCAO E' A MAIS PROXIMA DO OBSERVADOR */
        *lista_objetos_aux; /* APONTADOR PARA PERCORRER A LISTA DE OBJETOS */
 VETOR ponto;               /* PONTO DE INTERSECCAO COM O OBJETO */

 cor->r=cor->g=cor->b=0.0;        /* COR DO PONTO IGUAL A COR DO FUNDO */

 interseccao_menor=MAXIMO;
 obj_mais_proximo=NULL;
 lista_objetos_aux=lista_objetos;
 while(lista_objetos_aux){        /* CALCULA INTERSECCAO COM TODOS OS OBJETOS EM CENA */
     interseccao=(lista_objetos_aux->interseccao(lista_objetos_aux,raio));
   if((interseccao>0) && (interseccao<interseccao_menor)){
     interseccao_menor=interseccao;
     obj_mais_proximo=lista_objetos_aux;
  }
  lista_objetos_aux=lista_objetos_aux->proximo;
 }
 if(interseccao_menor==MAXIMO){  /* SE NAO HOUVE INTERSECCAO */
   cor->r=fundo.r;
   cor->g=fundo.g;
   cor->b=fundo.b;
 }
 else{   /* CALCULA A COR DO PONTO */
      cor->r=obj_mais_proximo->cor.r;
      cor->g=obj_mais_proximo->cor.g;
      cor->b=obj_mais_proximo->cor.b;
     }
  return;

}



void produz_raio(long double x,long double y,RAIO *raio)
/************************************************************************************

   FUNCAO    : produz_raio.

   OBJETIVO  : produz um novo raio a ser lancado na cena, a partir da especificacao de
               um ponto que esta' sendo escaneado na cena.

   ENTRADA   : (x,y) - coordenada da tela que esta' sendo escaneada.
               raio  - apontador que contera' o novo raio.

   SAIDA     : raio  - novo raio.

   AUTOR     : Luciano Silva
```

```
   DATA      : 03.11.97

   REVISAO   :                              REVISADO POR :

  ************************************************************************************/

{
 long double conv_x,    /* FATORES DE CONVERSAO DE ESCALA DA TELA */
             conv_y;
 VETOR vet_aux1,        /* VETORES AUXILIARES PARA CALCULO DO RAIO */
       vet_aux2;

 conv_x=(x-(long double) tela.comprimento/2.0)/(long double) tela.comprimento;
 conv_y=(((long double) (tela.altura-1)-y)-(long double) tela.altura/2.0)/(long double) tela.altura;
 escala(observador.view_cima,conv_y,vet_aux1);
 escala(observador.view_direita,conv_x,vet_aux2);
 adicao(vet_aux1,vet_aux2,raio->direcao);
 adicao(observador.direcao,raio->direcao,raio->direcao);
 normaliza(raio->direcao,raio->direcao);
 raio->origem.x=observador.localizacao.x;
 raio->origem.y=observador.localizacao.y;
 raio->origem.z=observador.localizacao.z;
 return;
}

void renderiza_cena(void)
/************************************************************************************

   FUNCAO    : renderiza_cena.

   OBJETIVO  : executa o tracado de raio na cena.

   ENTRADA   : nenhuma.

   SAIDA     : nenhuma.

   AUTOR     : Luciano Silva

   DATA      : 03.11.97

   REVISAO   :                              REVISADO POR :

  ************************************************************************************/

{
 int x,y;     /* COORDENADA DA TELA QUE ESTAO SENDO ESCANEADAS */
 RAIO raio;   /* RAIO PRODUZIDO */
 COR cor;     /* COR DO PONTO */

 for(x=0;x<tela.altura;x++){          /* VARRE TODAS A TELA */
   for(y=0;y<tela.comprimento;y++){
     produz_raio(y,x,&raio);           /* PRODUZ RAIO */
     lanca_raio_cena(&raio,&cor); /* LANCA RAIO NA CENA */
     desenha_pixel(x,y,cor);           /* DESENHA O PONTO */
   }
  }
```

```
  return;
}
```

1.6 Cálculo de intersecções com as faces

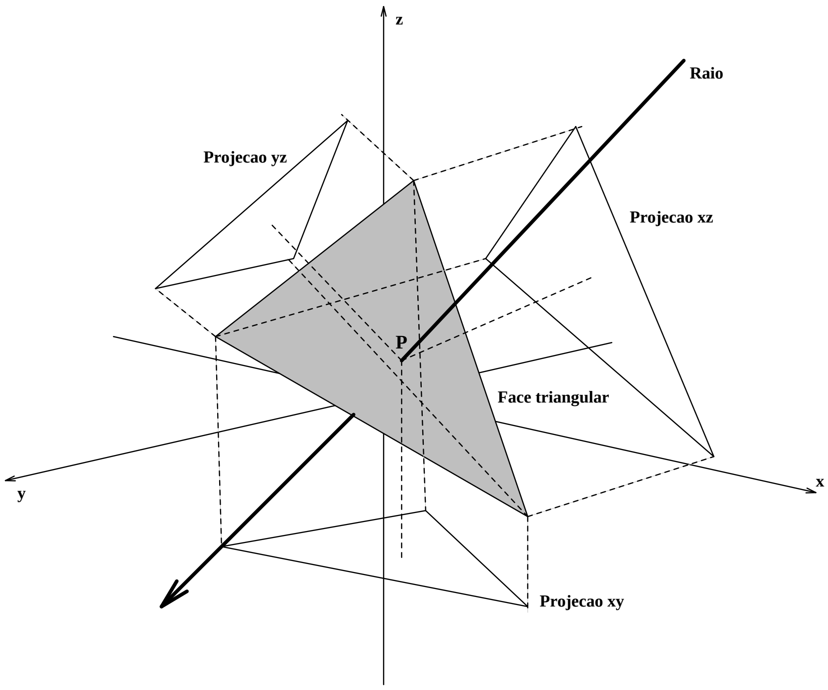
Para se verificar se um ponto de um objeto é um candidato a ser visto pelo observador, precisa-se saber como calcular a intersecção de um raio lançado na cena com o objeto.

No caso de objetos com faces triangulares, esta intersecção reduz-se ao caso de um teste interior/exterior tridimensional.

Este teste pode ser efetuado pela seguinte seqüência de passos:

1. Verificar se o raio intercepta o plano que contém a face triangular
2. Caso haja um ponto de interseccçã P, projeta-se a face triangular nos planos coordenados e verifica-se se a projeção de P está dentro de algum destes triângulos projetados.

A figura a seguir dá um exemplo:



No máximo, testamos as três projeções, pois algumas delas podem resultar em segmentos de retas, quando o plano que contém a face triangular é perpendicular a algum plano coordenado.

Estes testes são implementados pelo módulo *triângulo.c*, que contém, também, a tarefa adicional de alocar um objeto do tipo TRIANGULO.

```
/************************************************************************************

   PROJETO  :    VISUALIZADOR DE FECHO CONVEXO TRIDIMENSIONAL

   MODULO   :    TRIANGULO.C

   OBJETIVO :    FUNCOES DE MANIPULACAO E CONSTRUCAO DO OBJETO TRIANGULO

 ************************************************************************************/


 #include <stdlib.h>
 #include <stdio.h>
 #include <math.h>
 #include "estrutura.h"
 #include "comum.h"
 #include "vetores.h"
 #include "listas.h"
 #include "triangulo.h"


 extern OBJETO *lista_objetos;  /* LISTA DE FACES DE TRIANGULOS */

 int Esquerda(long double x1,long double y1,long double x2,long double y2,long double x3,long double y3){
 /************************************************************************************

   FUNCAO    : Esquerda.

   OBJETIVO  : Verifica se um ponto esta à esquerda de um segmento.

   ENTRADA   : (x1,y1) - Ponto de verificacao.
               (x2,y2)-(x3,y3) - segmento.

   SAIDA     : 0 - Nao esta a esquerda
               1 - Esta a esquerda.

   AUTOR     : Luciano Silva

   DATA      : 01.11.97

   REVISAO   :                            REVISADO POR :

  ************************************************************************************/

  long double determinante;
  determinante=(x1*y2+x2*y3+ x3*y1)-(x3*y2+x2*y1+x1*y3);
  if(determinante<=0.0)
    return(0);
  return(1);
 }


 int Dentro(long double x,long double y,long double x1,long double y1,
            long double x2,long double y2,long double x3,long double y3)
 /************************************************************************************

   FUNCAO    : Dentro.
```

```
  OBJETIVO  : Verifica se um ponto esta dentro de um triangulo bidimensional.

  ENTRADA   : (x,y) - Ponto.
              (x1,y1)-(x2,y2)-(x3,y3) - Triangulo.

  SAIDA     : 0 - Nao esta dentro.
              1 - Esta dentro.

  AUTOR     : Luciano Silva

  DATA      : 01.11.97

  REVISAO   :                            REVISADO POR :

 ***************************************************************************************/

{
 if (Esquerda(x,y,x1,y1,x2,y2)==1 &&
     Esquerda(x,y,x2,y2,x3,y3)==1 &&
     Esquerda(x,y,x3,y3,x1,y1)==1)
    return(1);
 return(0);
}

long double interseccao_triangulo(OBJETO *obj,RAIO *raio)
/***************************************************************************************

  FUNCAO    : interseccao_triangulo.

  OBJETIVO  : calcula a interseccao de um raio com um triangulo.

  ENTRADA   : obj  - apontador para definicao do triangulo.
              raio - apontador para definicao do raio.

  SAIDA     : parametro de interseccao.

  AUTOR     : Luciano Silva

  DATA      : 01.11.97

  REVISAO   :                            REVISADO POR :

 ***************************************************************************************/
{
 long double direcao,     /* FATORES DE INTERSECCAO */
             origem,
             T;
 long double x,y,z;

  produto_escalar(obj->geom_obj.triangulo.normal,raio->direcao,direcao);
  if(direcao==0.0)
    return(0.0);
  produto_escalar(obj->geom_obj.triangulo.normal,raio->origem,origem);
  origem=origem+obj->geom_obj.triangulo.distancia;
  origem=origem*(-1.0);
  T=origem/direcao;
```

```
  x=raio->origem.x+T*raio->direcao.x;
  y=raio->origem.y+T*raio->direcao.y;
  z=raio->origem.z+T*raio->direcao.z;


  if (Dentro(x,y,obj->geom_obj.triangulo.v1.x,obj->geom_obj.triangulo.v1.y,
                 obj->geom_obj.triangulo.v2.x,obj->geom_obj.triangulo.v2.y,
                 obj->geom_obj.triangulo.v3.x,obj->geom_obj.triangulo.v3.y)==1)
       return(T);
  else if  (Dentro(x,z,obj->geom_obj.triangulo.v1.x,obj->geom_obj.triangulo.v1.z,
                 obj->geom_obj.triangulo.v2.x,obj->geom_obj.triangulo.v2.z,
                 obj->geom_obj.triangulo.v3.x,obj->geom_obj.triangulo.v3.z)==1)
            return(T);
       else if (Dentro(y,z,obj->geom_obj.triangulo.v1.y,obj->geom_obj.triangulo.v1.z,
                 obj->geom_obj.triangulo.v2.y,obj->geom_obj.triangulo.v2.z,
                 obj->geom_obj.triangulo.v3.y,obj->geom_obj.triangulo.v3.z)==1)
             return(T);


  return(0.0);



}

OBJETO *aloca_objeto_triangulo(long double x1,long double y1,long double z1,
                               long double x2,long double y2,long double z2,
                               long double x3,long double y3,long double z3,
                               long double r,long double g,long double b)
/************************************************************************************

  FUNCAO    : aloca_objeto_triangulo.

  OBJETIVO  : declara um objeto do tipo triangulo.

  ENTRADA   : (x1,y1,z1) ,(x2,y2,z2), (x3,y3,z3) - vertices do triangulo
              (r,g,b) - cor do triangulo

  SAIDA     : apontador para estrutura do tipo OBJETO, com especificacao do triangulo.

  AUTOR     : Luciano Silva

  DATA      : 01.11.97

  REVISAO   :                             REVISADO POR :

 ************************************************************************************/
{

   OBJETO *triangulo=NULL;
   VETOR n1,
         n2;
   triangulo=malloc(sizeof(OBJETO));
   if(triangulo!=NULL){
     triangulo->interseccao=interseccao_triangulo;
     triangulo->geom_obj.triangulo.v1.x=x1;
     triangulo->geom_obj.triangulo.v1.y=y1;
```

```
      triangulo->geom_obj.triangulo.v1.z=z1;
      triangulo->geom_obj.triangulo.v2.x=x2;
      triangulo->geom_obj.triangulo.v2.y=y2;
      triangulo->geom_obj.triangulo.v2.z=z2;
      triangulo->geom_obj.triangulo.v3.x=x3;
      triangulo->geom_obj.triangulo.v3.y=y3;
      triangulo->geom_obj.triangulo.v3.z=z3;
      triangulo->cor.r=r;
      triangulo->cor.g=g;
      triangulo->cor.b=b;
      n1.x=x2-x1;
      n1.y=y2-y1;
      n1.z=z2-z1;
      n2.x=x3-x1;
      n2.y=y3-y1;
      n2.z=z3-z1;
      triangulo->geom_obj.triangulo.normal.x=n1.y*n2.z-n2.y*n1.z;
      triangulo->geom_obj.triangulo.normal.y=n1.z*n2.x-n1.x*n2.z;
      triangulo->geom_obj.triangulo.normal.z=n1.x*n2.y-n1.y*n2.x;
      normaliza(triangulo->geom_obj.triangulo.normal,triangulo->geom_obj.triangulo.normal);
      triangulo->geom_obj.triangulo.distancia=(-1.0) *(x1*triangulo->geom_obj.triangulo.normal.x+
                                              y1*triangulo->geom_obj.triangulo.normal.y+
                                              z1*triangulo->geom_obj.triangulo.normal.z);

      adiciona_objeto(&lista_objetos,triangulo);
    }
    return(triangulo);
  }
```

1.7 Inicialização dos parâmetros de cena

Todos os parâmetros relacionados com a cena, tais como posição do observador, tamanho do plano de projeção, etc, são especificados no módulo *main.c*. Isto é fixo e o usuário não pode alterá-los no momento do uso, a menos que se faça uma mudança do código e uma recompilação.

```
/*****************************************************************************************

   PROJETO  :    VISUALIZADOR DE FECHO CONVEXO TRIDIMENSIONAL

   MODULO   :    MAIN.C

   OBJETIVO :    DEFINICAO DOS PARAMETROS DE AMBIENTE, LEITURA DAS FACES DO FECHO,
                 POSICIONAMENTO DOS FECHOS NA CENA, CHAMADA DA FUNCAO DE RENDERIZACAO
                 DAS FACES.

 *****************************************************************************************/

#include <stdio.h>
#include <console.h>
#include "janelas.h"
#include "vetores.h"
#include "estrutura.h"
#include "listas.h"
#include "triangulo.h"
#include "raios.h"
```

```c
#include "comum.h"




extern COR fundo;               /* COR DO FUNDO */
extern OBSERVADOR observador;   /* POSICIONAMENTO DO OBSERVADOR */
extern DISPLAY tela;            /* TELA DE PROJECAO */
extern OBJETO *lista_objetos;   /* LISTA DE FACES */




void main(int argc,char *argv[]){

  OBJETO *obj;    /* AUXILIAR DE ALOCACAO DAS FACES */
  float x1,y1,z1,x2,y2,z2,x3,y3,z3;  /* COORDENADAS DOS EXTREMOS DAS FACES */
  int num_faces,  /* NUMERO DE FACES */
      i,          /* CONTADOR DE LEITURA DAS FACES */
      cor=-1;      /* AUXILIAR DE MUDANCA DE COR DAS FACES */


  /* LEITURA DO NOME DO ARQUIVO DE ENTRADA */

  ccommand(&argv);

   /* INICIALIZACAO DO AMBIENTE GRAFICO DO MACINTOSH */
  InitMacintosh();
  SetUpWindow();


  /* LEITURA DAS FACES */

  scanf("%d",&num_faces);
  for(i=1;i<=num_faces;i++){
    scanf("%f %f %f %f %f %f %f %f %f",&x1,&y1,&z1,&x2,&y2,&z2,&x3,&y3,&z3);
    cor++;
    if(cor==0)
      obj=aloca_objeto_triangulo(x1,y1,z1,x2,y2,z2,x3,y3,z3,0.0,0.0,1.0);
    else obj=aloca_objeto_triangulo(x1,y1,z1,x2,y2,z2,x3,y3,z3,1.0,0.0,0.0);
    if(cor==1)
      cor=-1;
  }


  /* INICIALIZACAO DOS ATRIBUTOS DAS CENA */

  fundo.r=fundo.g=fundo.b=0;
  observador.localizacao.x=100;
  observador.localizacao.y=100;
  observador.localizacao.z=100;
  observador.direcao.x=-1;
  observador.direcao.y=-1;
  observador.direcao.z=-1;
  observador.view_cima.x=0;
  observador.view_cima.y=1;
```

```
  observador.view_cima.z=0;
  observador.view_direita.x=1.0;
  observador.view_direita.y=0;
  observador.view_direita.z=0;
  tela.comprimento=MAXIMO_X;
  tela.altura=MAXIMO_Y;

  /* VISUALIZACAO DA CENA */

  renderiza_cena();

  while(1);


}
```

1.8 Arquivos de cabeçalhos auxiliares

Existem dois arquivos de cabeçalho auxiliares, que estão presentes em quase todos os módulos:

- um de definição de constantes e macros básicas, chamado *comum.h*
- outro de macros para vetores, chamado *vetores.h*

A seguir, tem-se suas listagens:

1.8.1 Definições comuns aos módulos

```
/*****************************************************************************************

   PROJETO  :    VISUALIZADOR DE FECHO CONVEXO TRIDIMENSIONAL

   HEADER   :    COMUM.H

   OBJETIVO :    DEFINICOES DE TIPOS, LIMITES E MACROS COMUNS AOS MODULOS DO
                 TRACADOR DE RAIOS.

 *****************************************************************************************/


 #pragma once

 typedef unsigned char TIPO_BYTE;



 #define MAXIMO 1e10
 #define MINIMO 1e-03



 #define MIN(a,b) ((a)>(b)) ? (b):(a)
 #define MAX(a,b) ((a)>(b)) ? (a):(b)
```

```
#define QUADRADO(x) (x)*(x)

#define MAXIMO_X 400
#define MAXIMO_Y 400


extern long double temp;
```

1.8.2 Macros de vetores

```
/***************************************************************************************

   PROJETO  :    VISUALIZADOR DE FECHO CONVEXO TRIDIMENSIONAL

   HEADER   :    VETORES.H

   OBJETIVO :    DEFINICAO DAS MACROS DE MANIPULACAO DE VETORES

 ***************************************************************************************/


#pragma once
#include <math.h>

#define adicao(a,b,c){ (c).x=(a).x+(b).x;\
                       (c).y=(a).y+(b).y;\
                       (c).z=(a).z+(b).z;}


#define subtracao(a,b,c){ (c).x=(a).x-(b).x;\
                          (c).y=(a).y-(b).y;\
                          (c).z=(a).z-(b).z;}


#define inverte(b,c){(c).x=-(b).x;\
                     (c).y=-(b).y;\
                     (c).z=-(b).z;}


#define escala(a,k,c){ (c).x=(k)*(a).x;\
                       (c).y=(k)*(a).y;\
                       (c).z=(k)*(a).z;}

#define produto_escalar(a,b,c) {c=(a).x*(b).x+(a).y*(b).y+(a).z*(b).z;}

#define produto_vetorial(a,b,c) { (c).x=(a).y*(b).z-(a).z*(b).y;\
                                  (c).y=(a).z*(b).x-(a).x*(b).z;\
                                  (c).z=(a).x*(b).y-(a).y*(b).x;}


#define norma(a,c) {c=sqrt((a).x*(a).x+(a).y*(a).y+(a).z*(a).z);}



#define normaliza(a,n) { temp=sqrt((a).x*(a).x+(a).y*(a).y+(a).z*(a).z);\
                         (n).x=(n).x/temp;\
                         (n).y=(n).y/temp;\
```

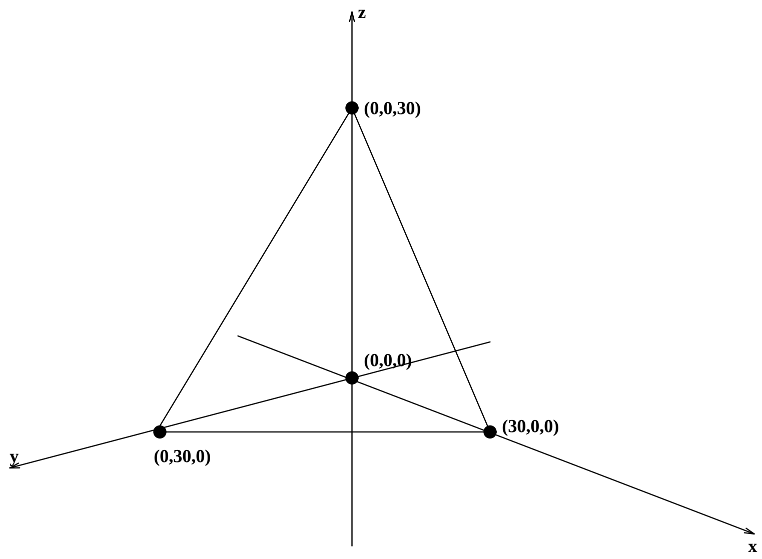
```
(n).z=(n).z/temp;}
```

1.9 Formato do arquivo de entrada

O formato de arquivo de entrada para o visualizador é o seguinte:

```
Número de faces triangulares
Vértice1Face1
Vértice2Face1
Vértice3Face1
....
Vértice1FaceN
Vértice2FaceN
Vértice3FaceN
```

onde os vértices de uma face precisam ser dados em alguma ordem (horária ou anti-horária).
Por exemplo, para o seguinte objeto:



teríamos o seguinte arquivo de entrada:

```
4
0 0 0
30 0 0
0 30 0
0 0 0
0 30 0
```

```
0 0 30
0 0 0
30 0 0
0 0 30
30 0 0
0 30 0
0 0 30
```

Cada face lida ou recebe a cor azul ou a cor vermelha, para melhor diferenciação.

Referências Bibliográficas

[1] A.S. Glassner.*An Introduction to Ray Tracing.* Academic Press, 1989.

[2] J.D. Foley, A. vanDam,S.K. Feiner, J.F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, 1996.

[3] A. Watt. *3-D Computer Graphics.* Addison-Wesley, 1993.

[4] A. Watt. *Advanced Animation and Rendering Techniques: Theory and Practice.* ACM Press, 1992.