University of Bologna - Department of Mathematics

Piazza di Porta S.Donato, 5 - 40127 - Bologna

trim library
Programming Guide - Version 1.0

G. Casciola G. De Marco

Department of Mathematics
University of Bologna

Bologna 2000

Abstract

This report describes the *trim* library. It is designed to a real-time rendering of NURBS surfaces, trimmed NURBS surfaces, but a 3D mesh too, in Xwindow environment.

G. Casciola G. De Marco
Department of Mathematics, University of Bologna, P.zza di Porta S.Donato 5, Bologna, Italy. E-mail: casciola@dm.unibo.it.

Contents

CHAPTER 1

What's *trim* ?

trim is a graphics library written in ANSI C language for Unix and Xwindow environment in order to render NURBS surfaces, trimmed NURBS surfaces, but a simple 3D mesh too, in real time. The trimmed NURBS surfaces are frequently used in many *computer graphics* applications. Fundamentally, they are the output of Surface/Surface Intersection algorithms and are used to represent the boundary of complex solid bodies, resulting by boolean operations. The rendering of surfaces can be very realistic by applying a ray-tracing algorithm, but this is slow and not useful for a real-time rendering. Usually, a real-time rendering algorithm starts by a tessellation of the surface, that is by an its piecewise planar approximation or more simply by a 3D mesh of the surface consisting in a set of 3D surface points, a list of edges and a list of facets (usually quadrilateral or triangular). These algorithms are less realistic, but more faster. The *trim* library is based on this approach and implements some strategies to tessellate a trimmed NURBS surface and different methods to render the tessellation as wire-frame, hidden-line and shading; the library supports graphics mode with 8, 16, 24 and 32 bit-colors. This report is a programming guide for a user who want to use it in applications where a real-time rendering is useful also without the support of a graphic hardware.

Chapter 2

Library functions

2.1 Read surfaces

```
int Trim_read_nurbs(FILE *f,
                    nurbs_surface **nurbs)
int Trim_read_trimmed_nurbs(FILE *f,
                            Trimmed_nurbs_t **tnurbs)
```

These functions read, respectively, from an `f` file, a NURBS surface (*.db* format) and a trimmed NURBS surface (*.tree* format), managing also the suitable memory allocation; if an error occurs, the functions return a non null value (see section 2.5).

```
void Trim_free_nurbs(nurbs_surface *nurbs)
void Trim_free_trimmed_nurbs(Trimmed_nurbs_t *tnurbs)
```

These functions allow us to deallocate the used memory for a surface.

2.2 Tessellation

Before to proceed with the visualization it is necessary to tessellate the surfaces. The tessellation parameter control the grid type (uniform or adaptive) to be used for the parametric surface domain, the number of isoparametric lines in the grid andi, in the case of a trimmed surface, a tolerance aproximation value for the *trimming curves*. While the parameters NU and NV in the uniform grid case and SRT in the adaptive grid case, control the approximation level of the surface in terms of number of isoparametric lines, instead, the CRT tolerance controls the number of points of the trimming curves to be used for the tessellation.

```
void Trim_set_params(Sub_type_t sub,
                     int nu,
                     int nv,
                     REAL srt,
                     REAL crt,
                     Trim_par_t *par)
```

This function sets `par` with the tessellation parameters: `sub` defines the grid type (uniform or adaptive); `nu` and `nv` define respectively the number of isoparametric lines in the direction U and V in the case of a uniform grid; `srt` defines the adaptive subdivision tolerance in the case of an adaptive grid; finally `crt` defines the approximation tolerance for the trimming curves, in the case of a trimmed surface.

```
void Trim_get_params(Trim_par_t *par,
                     Sub_type_t *sub,
                     int *nu,
                     int *nv,
                     REAL *srt,
                     REAL *crt)
```

This function gets the tessellation parameters `sub`, `nu`, `nv`, `srt` and `crt` from `par` previously setted or initialized by default.

Description	Par	Values
grid type	sub	_TRIM_UNIFORM, _TRIM_ADAPTIVE
no. U isopar.	nu	[_TRIM_MINISO,_TRIM_MAXISO]
no. V isopar.	nv	[_TRIM_MINISO,_TRIM_MAXISO]
surface tol. (SRT)	srt	>0
curve tol. (CRT)	crt	>0

Table 2.1: tessellation Parameters

```
int Trim_nurbs_triangulation(nurbs_surface *nurbs,
                             Trim_par_t *par,
                             Trimmed_surface_t **tsurf)
int Trim_triangulation(Trimmed_nurbs_t *tnurbs,
                       Trim_par_t *par,
                       Trimmed_surface_t **tsurf)
```

These functions make, respectively, the parametric domain tessellation of a NURBS surface (`nurbs`) and of a trimmed NURBS surface (`tnurbs`) using the par parameters; they return the pointer to the memory area (tsurf) containing the tessellation data: parametric domain points, segments and polygons and grid type used. If an error occurs the functions return a non null value with the following meaning:

Error	Description
_TRIM_ERR_SRT	SRT value too small
_TRIM_ERR_GRID	no valid grid
_TRIM_ERR_CRT	CRT value too small

The _TRIM_ERR_SRT error occurs when an SRT value too small gives a number of isoparametric lines in U or V direction greater than _TRIM_MAXISO; the _TRIM_ERR_GRID error occurs when the generated grid does not satified the constraint to have, at the most, one intersection for each grid edge with the trimming curves; the _TRIM_ERR_CRT error occurs when the distance between consecutive trimming curve points is geater than the CRT required tolerance;

```
void Trim_new_surface(nurbs_surface *nurbs,
                      Trimmed_surface_t *tsurf)
```

This function computes, starting from the parametric domain tessellation (points, segments and polygons) the NURBS surface tessellation by mapping the domains points on the surface defined by nurbs (in the case of a trimmed NURBS surface thi parameter is simply the field nurbs of the structure Trimmed_nurbs_t).

```
void Trim_delete_surface(Trimmed_surface_t *tsurf)
```

This function deallocates the tessellated surface tsurf.

2.3 Information

In order to the 3D visualization of one or more surfaces it is useful to compute their bounding box.

```
void Trim_nurbs_bbox(nurbs_surface *nurbs,
                     Bbox_t *bb )
```

This function computes the bb *bounding box* of a NURBS surface nurbs.

```
void Trim_surface_bbox(Trimmed_surface_t *tsurf,
                       Bbox_t *bb)
```

This function computes the bb *bounding box* of a tessellated surface tsurf.

```
void Trim_surface_list_bbox(Trimmed_surface_t **vtsurf,
                            int nsurf,
                            Bbox_t *bb)
```

This function computes the bb *bounding box* of a tessellated surface list whose pointers are stored in the array vtsurf with nsurf elements.

2.4 Visualization

The *trim* library provides a class of functions for the visualization of tessellated surfaces within their parametric domains. Before these functions have to be called, an initialization of the graphics library is necessary.

2.4.1 Initialization

```
void Trim_init(Display *disp,
               Colormap map)
void Trim_close(void)
```

These functions begin and end the call to the *trim* library graphics functions.

2.4.2 Parametric Domain

```
void Trim_draw_domain(TWin_t *win
                      BOOL fill)
```

This function draws the parametric domain $[0, 1] \times [0, 1]$ in the specified window win; the parameter fill sets if the whole domain defines an active region (fill==TRUE) or a non active region (fill==FALSE) of a surface.

```
void Trim_draw_curve(TWin_t *win,
                     Trimming_curve_t *curve,
                     BOOL hole,
                     BOOL fill)
```

This function draws in the window win the trimming curve specified by curve; the parameter hole points out the region type defined by the trimming curve (TRUE for non active region, FALSE for active).

```
void Trim_draw_tree(TWin_t *win,
                    Trimmed_nurbs_t *tnurbs,
                    BOOL fill)
```

This function visualize in the window `win` the *trimming curve tree* containing, in alternate levels, the curve definining the active and non active regions of the parametric domain of the trimmed NURBS surface `tnurbs`; the parameter `fill` specifies if the regions inside the curve must be drawn in a given colour to hilight them, or not.

```
void Trim_draw_grid(TWin_t *win,
                    Grid_par_t *grid)
```

This function visualize the specified grid `grid` in the window `win`; usually this parameter represents the used grid for the tessellation of a surface and therefore points out the field `grid` of the structure `Trimmed_surface_t` containing the tesselation data.

```
void Trim_draw_triangulation(TWin_t *win,
                             Trimmed_surface_t *tsurf)
```

This function draws the polygons of the tesselation `tsurf` of the parametric domain of a surface.

```
void Trim_draw_trimmed_domain(TWin_t *win,
                              Trimmed_nurbs_t *tnurbs,
                              Trimmed_surface_t *tsurf)
```

This function visualize both the *trimming curve tree* and the computed tessellation `tsurf` for the parametric domain of the trimmed NURBS surface `tnurbs`.

2.4.3 Tassellated Surfaces

After the tesselation of the surfaces has been realized, the library is able to visualize all the tessellated surfaces with one of the following methods: *wire-frame*, *depth cueing*, *hidden line*, *Flat shading*, *Gouraud shading* and *Phong shading*.
The shading methods require the definition of the light position and the object attributes, that is *ambient reflection*, *diffuse reflection*, *specular reflection* and the *Phong exponent*. The last two parameters are effective only for the *Phong shading* method. In fact this method results more realistic respect the others, but at the same time results the "slowest".

```
void Trim_default_visual(Visual_par_t *vis)
```

This function sets the structure `vis` fields with default parameters of visualization. The structcure `Visual_par_t` allows us to define: the visualization method (`mode`), the coordinate system for the light (`sys`), the light position (`pos`), and finally the functions to be used for the perpective projection of the 3D points (see table 2.2). More precisely the field `persp_par` is the pointer to the function which sets the projection parameters and `persp_pro` is the pointer to the function which computes the projection. The default values of these two fields are the pointer to two built in function of the *trim* library:

```
void Trim_persp3_par(View_par_t *view,
                     int vxmin,
                     int vymin,
                     unsigned int width,
                     unsigned int height)
void Trim_persp3(point_r3 *pt,
                 int *vx,
                 int *vy,
                 REAL *pz)
```

For detail about the implementation of the `Visual_par_t` type, you can see chapter 3.

Description	Field	Default	Values
visual method	mode	- * - - - 	_TRIM_WIREFRAME _TRIM_DEPTHCUEING _TRIM_HIDDENLINE _TRIM_FLAT _TRIM_GOURAUD _TRIM_PHONG
light system	sys	- *	_TRIM_OBJSYS _TRIM_COPSYS
lihght position	pos.teta pos.fi	-30.0 -15.0	[-2π,2π] [$-\pi$,π]
projection funct. par.	persp_par	Trim_persp3_par	
projection funct.	persp_pro	Trim_persp3	

Table 2.2: Visualization parameters (`Visual_par_t`)

```
void Trim_default_attrib(Obj_attrib_t *attr)
```

This function sets `attr` with the defualt attributes (see. table 2.3).

```
void Trim_draw_axes(TWin_t *win,
                    Visual_par_t *vis,
                    View_par_t *view)
```

This function draws in the window `win`, the Cartesian axes, using the visualization parameters `vis` and the view parameters `view` (see table 2.4).

Description	Field	Default	Values
	color.c[0]	239	[0,255]
RGB colour	color.c[1]	239	[0,255]
	color.c[2]	239	[0,255]
ambient reflection	Ambient_rfl	0.3	[0,1]
diffuse reflection	Diffuse_rfl	0.5	[0,1]
specular reflection	Specular_rfl	0.2	[0,1]
Phong exponent	Gloss	10.0	≥ 0

Table 2.3: Object attributes (`Obj_attrib_t`)

Description	Field
center of projection (COP)	`cop`
center of the scene or of the circumscribe sphere	`cs`
ray of the circumscribe sphere	`r`
distance projection plane and COP (0.0 automatic)	`di`
half edge window (0.0 automatic)	`da`
x coordinate of the window center	`ox`
y coordinate of the window center	`oy`

Table 2.4: View parameters

```
int Trim_visual_create(TWin_t *win,
                       Visual_par_t *vis,
                       Obj_attrib_t *attr,
                       Trimmed_surface_t **vtsurf,
                       int nsurf
                       int *id)
```

This function does the prepocessing steps for a new 3D visualization of a list of tessellated surfaces whose pointers are stored in the array `vtsurf` with `nsurf` elements, using the parameters `vis`. The object attributes `attr` are effective only for the shading methods while the `id` parameter allows us to identifie the new visulization. If an error occurs, the function returns a non null value (see section 2.5).

```
int Trim_visual_do( int id,
                    View_par_t *view )
```

This function computes the image of the desired `id` visualization using the view parameters `view`.

```
void Trim_visual_destroy( int id );
```

This function destroyes the computed data in the preprocessing steps for the `id` visualization.

2.5 Error Functions

Some *trim* library functions, when an error occurs, update suitable variables. These variables are fields of the global structure `_trim_error` and contain the error type occurred and line code where the error has been encountered. The following table 2.5 shows the error codes generated by the *trim* library functions.

Error	Description
_TRIM_ERR_NONE=0	no error
_TRIM_ERR_FREAD	reading error
_TRIM_ERR_FWRITE	writing error
_TRIM_ERR_GENERIC	generic error
_TRIM_ERR_SRT	SRT value too small
_TRIM_ERR_CRT	CRT value too small
_TRIM_ERR_GRID	grid not valid
_TRIM_ERR_VIS	visualiz. params. not valid
_TRIM_ERR_ATTR	object attributes not valid
_TRIM_ERR_VIEW	view params. not valid
_TRIM_ERR_INIT	visual not inizialized
_TRIM_ERR_DISP	display depth not supported

Table 2.5: Error codes

```
int Trim_error_number(void)
```

This function, if an error occurs, returns a value different by _TRIM_ERR_NONE.

```
char *Trim_error_msg(void)
```

This function returns a message corresponding to the error code occurred.

```
void Trim_error_clear(void)
```

This function resets the error variables.

Chapter 3

Implemented Types

In the following, some implemented types in the *trim* library, are reported.

```
typedef int Poledge_t[2];
typedef int Polvert_t[4];

typedef struct {
  BYTE c[3]; /* R G B */
} Color_t;

typedef struct {
  REAL *u;  /* U iso. */
  REAL *v;  /* V iso. */
  int nu;   /* U num. iso. */
  int nv;   /* V num. iso. */
} Grid_par_t;

typedef struct {
  Sub_type_t sub_type;  /* grid type */
  union {
    struct {
      int nu;           /* U num. iso  (uniform grid) */
      int nv;           /* V num. iso  (uniform grid) */
    } niso;
    REAL srt;           /* SRT nurbs (adaptive grid) */
  } sub_par;            /* grid parameters */
  REAL crt;             /* CRT nurbs (tessellation) */
} Trim_par_t;

typedef struct {
  int npt;           /* num. of points */
```

```
  int nseg;          /* num. of segments */
  int npol;          /* num. of polygons */
  struct {
    point_r2 *pt;    /* tessellation points [1..npt] */
    Poledge_t *seg;  /* tessellation segments [1..nseg] */
    Polvert_t *pol;  /* tessellation polygons [1..npol] */
  } domain;
  struct {
    point_r3 *pt;    /* tessellation points [1..npt] */
    Poledge_t *seg;  /* tessellation segments [1..nseg] */
    Polvert_t *pol;  /* tessellation polygons[1..npol] */
  } surface;
  Grid_par_t *grid;  /* grid parameters */
  Trim_par_t par;    /* trimming curve parameters*/
} Trimmed_surface_t;

typedef struct {
  REAL teta;  /* teta angle respect to XZ */
  REAL fi;    /* fi angle respect to XY */
} Light_pos_t;

typedef struct {
  Color_t color;     /* colour */
  REAL Ambient_rfl;  /* % of ambient reflected ligth */
  REAL Diffuse_rfl;  /* % of diffuse reflected light */
  REAL Specular_rfl; /* % of specular reflected light */
  REAL Gloss;        /* Phong exponent */
} Obj_attrib_t;

typedef struct {
  point_r3 cop; /* center of projection */
  point_r3 cs;  /* center of the sphere circumscribed */
  REAL r;       /* ray of the sphere */
  REAL di;      /* distance projection plane/COP (0.0 automatic) */
  REAL da;      /* half edge window (0.0 automatic) */
  REAL ox;      /* x coordinate window center*/
  REAL oy;      /* y coordinate window center */
} View_par_t;

typedef struct {
  Drawing_mode_t mode; /* visualization method */
  Light_sys_t sys;     /* light system */
  Light_pos_t pos;     /* light position */
                       /* parameters function for projection */
```

```
  void (*persp_par)(View_par_t *view, int vxmin, int vymin,
    unsigned int widht, unsigned int height);
                       /* function projection */
  void (*persp_pro)(point_r3 *pt, int *vx, int *vy, REAL *pz);
} Visual_par_t;

typedef struct {
  int xpos;              /* x screen position */
  int ypos;              /* y screen position */
  int vxmin;             /* x window origin */
  int vymin;             /* y window origin */
  unsigned int width;    /* window width */
  unsigned int height;   /* window height */
  Window winid;          /* window id */
  Window pixid;          /* pixmap id */
  Drawable id;           /* default id to be used */
  BOOL flag;             /* set id with winid or pixid */
} TWin_t;
```

Chapter 4

Example Programs

4.1 Example 1: *xtreeview*

This example program uses the *trim* library to visualize a trimmed NURBS surface (.tree format).

```
/* xtreeview.c */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/keysym.h>

#ifndef __USE_BSD
#define __USE_BSD
#endif
#include <math.h>

#include "trim/trim.h"
#include "trim/test/xtview_a.h"

static Trimmed_nurbs_t *tnurbs;      /* trimmed nurbs surface */
static Trimmed_surface_t *vtsurf[1]; /* tessellation data */

static int visid;
static Display *disp;
static TWin_t win;
static GC gc;
static Colormap map;
static unsigned long color[2];
```

```
int main( int argc, char **argv )
{
  FILE *f;
  REAL diag,ccr,stima_srt,stima_crt;
  Trim_par_t par;
  Bbox_t bb;

  if ( argc!=2 ) {
    printf( "Usage: xtview treefile \n" );
    printf( "  treefile  trimmed NURBS surface (.tree)\n\n" );
    exit( 1 );
  }
  if ( (f=fopen(argv[1], "r" ))==NULL ) {
    fprintf( stderr, "xtview: Cannot open %s\n", argv[1] );
    exit( 1 );
  }

  if( Trim_read_trimmed_nurbs(f, &tnurbs) ) {
    fprintf( stderr, "xtview: Bad file format %s\n", argv[1] );
    exit( 1 );
  }
  fclose( f );

  Trim_nurbs_bbox( tnurbs->nurbs, &bb );

  /* surface SRT estimate */
  ccr = 0.5*sqrt(SQR(bb.xmax-bb.xmin)+SQR(bb.ymax-bb.ymin)+SQR(bb.zmax-bb.zmin));
  stima_srt = ccr/200.0;
  /* trimming curves CRT estimate */
  diag = sqrt(SQR(bb.xmax-bb.xmin)+SQR(bb.ymax-bb.ymin)+SQR(bb.zmax-bb.zmin));
  stima_crt = diag/30.0;

  Trim_set_params( _TRIM_ADAPTIVE, 0, 0, stima_srt, stima_crt, &par );
  if( Trim_triangulation( tnurbs, &par, &vtsurf[0] ) ) {
    printf( "xtview: %s\n", Trim_error_msg() );
    exit( 1 );
  }
  Trim_new_surface( tnurbs->nurbs, vtsurf[0] );

  init_X( &disp, &map );
  Trim_init( disp, map );

  win.xpos = XW, win.ypos = YW;
```

```
  win.vxmin = 0, win.vymin = 0;
  win.width = WW, win.height = HW;
  win.flag = _TRIM_DEFPIX;
  new_window( disp, &win, &gc, map );

  visual_do( disp, &win, gc, vtsurf );

  Trim_delete_surface( vtsurf[0] );

  Trim_close();
  XFreePixmap( disp, win.pixid );
  XDestroyWindow( disp, win.winid );
  XFreeGC( disp, gc );
  XCloseDisplay( disp );
  return 0;
}


/*---------------- local functions ------------------------*/

static
void init_X( Display **disp, Colormap *map )
{
  if ( (*disp=XOpenDisplay(""))==NULL ) {
    fprintf( stderr, "xtview: Cannot open display\n" );
    exit( 1 );
  }
  *map = DefaultColormap( *disp, DefaultScreen(*disp) );
  color[0] = _trim_grey[0];
  color[1] = _trim_grey[_TRIM_NGREY-1];
}


static
void new_window( Display *disp, TWin_t *win, GC *gc, Colormap clmap )
{
  Window root;
  XSizeHints win_h;
  XSetWindowAttributes att;

  root = DefaultRootWindow( disp );
  *gc = XCreateGC( disp, root, 0, 0 );
  win_h.x = win->xpos, win_h.y = win->ypos;
  win_h.width = win->vxmin+win->width-1, win_h.height = win->vymin+win->height-1;
```

```
  win_h.flags = USPosition | PSize;
  att.backing_store=Always;
  win->pixid = _TRIM_NO_ID;
  win->winid = XCreateSimpleWindow( disp, root, win_h.x, win_h.y,
       (unsigned int)win_h.width, (unsigned int)win_h.height, 10,
       color[1], color[0] );
  XChangeWindowAttributes( disp, win->winid, CWBackingStore, &att );
  XSetStandardProperties( disp, win->winid, "xtview", NULL, None, 0, 0, &win_h );
  XSetWindowColormap( disp, win->winid, clmap );
  win->pixid = XCreatePixmap( disp, win->winid,
       (unsigned int)win_h.width, (unsigned int)win_h.height,
       (unsigned int)DefaultDepth( disp, DefaultScreen( disp ) ) );
  win->id = (win->flag==_TRIM_DEFWIN) ? win->winid : win->pixid;
  XSelectInput( disp, win->winid, KeyPressMask );
  XMapRaised( disp, win->winid );
  XFlush( disp );
}


static
BOOL new_cop( point_r3 *cop, point_r3 *cs, int dir )
{
  point_r3 copt;
  REAL ro,teta,fi;

  copt.x = cop->x - cs->x;
  copt.y = cop->y - cs->y;
  copt.z = cop->z - cs->z;

  ro = sqrt(SQR(copt.x)+SQR(copt.y)+SQR(copt.z));
  teta = atan2(copt.y,copt.x);
  fi = acos(copt.z/ro);

  switch(dir) {
  case 1:
    fi -= M_PI/ROTSTEP;
    if (fi<0.1) return FALSE;
    break;
  case 2:
    if (teta>2*M_PI) teta -= 2*M_PI;
    teta += M_PI/ROTSTEP;
    break;
  case 3:
    fi += M_PI/ROTSTEP;
```

```
    if (fi>3.1) return FALSE;
    break;
  case 4:
    if (teta<0) teta += 2*M_PI;
    teta -= M_PI/ROTSTEP;
    break;
  }
  cop->x = cs->x+(REAL)(ro*sin(fi)*cos(teta));
  cop->y = cs->y+(REAL)(ro*sin(fi)*sin(teta));
  cop->z = cs->z+(REAL)(ro*cos(fi));
  return TRUE;
}


static
void visual_do( Display *disp, TWin_t *win, GC gc, Trimmed_surface_t **data )
{
  XEvent xevent;
  Obj_attrib_t attr;
  Visual_par_t vis;
  View_par_t view;
  BOOL quit=FALSE;
  Bbox_t bb;
  REAL cr;
  point_r3 cop_o;
  BOOL cop_flag=TRUE;

  printf( "\nKeys: LEFT, RIGHT, UP, DOWN -rotate object\n" );
  printf( "      KP4, KP6, KP8, KP2    -object point\n" );
  printf( "      KP+                   -zoom in\n");
  printf( "      KP-                   -zoom out\n");
  printf( "      w (Wire Frame)  d (depth cueing)\n");
  printf( "      h (Hidden Line)     f (Flat Shading)\n");
  printf( "      g (Gouraud Shading) p (Phong Shading) \n");
  printf( "      ESC                   -exit program\n");

  Trim_surface_bbox( data[0], &bb);
  cr = 0.5 * sqrt(SQR(bb.xmax-bb.xmin) + SQR(bb.ymax-bb.ymin) +
  SQR(bb.zmax-bb.zmin));
  view.cs.x = (bb.xmax+bb.xmin)/2.0;
  view.cs.y = (bb.ymax+bb.ymin)/2.0;
  view.cs.z = (bb.zmax+bb.zmin)/2.0;
  view.cop.x = view.cs.x+2*cr;
  view.cop.y = view.cs.y;
```

```
view.cop.z = view.cs.z+cr;
view.r = cr;
cop_o.x = view.cop.x - view.cs.x;
cop_o.y = view.cop.y - view.cs.y;
cop_o.z = view.cop.z - view.cs.z;
view.di = 0.0;
view.da = view.r/2.0;
view.ox = view.oy = 0.0;

Trim_default_attrib( &attr );
Trim_default_visual( &vis );
vis.mode = _TRIM_DEPTHCUEING;
vis.pos.teta = 0.0;

Trim_visual_create( win, &vis, &attr, data, 1, &visid );

do {
  if( cop_flag ) {
    XSetForeground( disp, gc, color[0] );
    XFillRectangle( disp, win->pixid, gc, win->vxmin, win->vymin,
    win->width, win->height );

    Trim_visual_do( visid, &view );

    XCopyArea( disp, win->pixid, win->winid, gc, 0, 0,
    win->width, win->height, 0, 0 );
    XFlush( disp );
  }
  do {
    XNextEvent( disp, &xevent );
  } while( (xevent.type!=KeyPress) );
  switch( XLookupKeysym( &xevent.xkey, 0 ) ) {
  case XK_Escape:
    quit = TRUE;
    break;
  case XK_w:
    vis.mode = _TRIM_WIREFRAME;
    Trim_visual_create( win, &vis, &attr, data, 1, &visid );
    break;
  case XK_d:
    vis.mode = _TRIM_DEPTHCUEING;
    Trim_visual_create( win, &vis, &attr, data, 1, &visid );
    break;
  case XK_h:
```

```
    vis.mode = _TRIM_HIDDENLINE;
    Trim_visual_create( win, &vis, &attr, data, 1, &visid );
    break;
  case XK_f:
    vis.mode = _TRIM_FLAT;
    Trim_visual_create( win, &vis, &attr, data, 1, &visid );
    break;
  case XK_g:
    vis.mode = _TRIM_GOURAUD;
    Trim_visual_create( win, &vis, &attr, data, 1, &visid );
    break;
  case XK_p:
    vis.mode = _TRIM_PHONG;
    Trim_visual_create( win, &vis, &attr, data, 1, &visid );
    break;
  case XK_Left:
    cop_flag = new_cop( &view.cop, &view.cs, 4 );
    break;
  case XK_Right:
    cop_flag = new_cop( &view.cop, &view.cs, 2 );
    break;
  case XK_Up:
    cop_flag = new_cop( &view.cop, &view.cs, 1 );
    break;
  case XK_Down:
    cop_flag = new_cop( &view.cop, &view.cs, 3 );
    break;
  case XK_KP_Add:
    cop_flag = TRUE;
    view.da -= view.r/20.0;
    break;
  case XK_KP_Subtract:
    cop_flag = TRUE;
    view.da += view.r/20.0;
    break;
  case XK_KP_Left:
    cop_flag = TRUE;
    view.ox -= view.r/20.0;
    break;
  case XK_KP_Right:
    cop_flag = TRUE;
    view.ox += view.r/20.0;
    break;
  case XK_KP_Up:
```

```
      cop_flag = TRUE;
      view.oy += view.r/20.0;
      break;
    case XK_KP_Down:
      cop_flag = TRUE;
      view.oy -= view.r/20.0;
      break;
    default:
      cop_flag = FALSE;
      break;
    }
  } while ( !quit );

  Trim_visual_destroy( visid );
}
```

4.2 Example 2: *xmview*

This example program uses the *trim* library to visualize a 3D mesh (.m format).

```
/* xmview.c */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/keysym.h>

#ifndef __USE_BSD
#define __USE_BSD
#endif
#include <math.h>

#include "trim/trim.h"
#include "trim/test/xmview.h"

static Trimmed_surface_t *data[1]; /* tessellation data */

static int visid;
static Display *disp;
static TWin_t win;
```

```
static GC gc;
static Colormap map;
static unsigned long color[2];

int main( int argc, char **argv )
{
  FILE *f;

  if ( argc!=2 ) {
    printf( "Usage: xmview mfile \n" );
    printf( "  mfile  .m surface data\n\n" );
    exit( 1 );
  }
  if ( (f=fopen(argv[1], "r" ))==NULL ) {
    fprintf( stderr, "xmview: Cannot open %s\n", argv[1] );
    exit( 1 );
  }
  if ( !read_surface_data(f, &data[0]) ) {
    fprintf( stderr, "xmview: Bad file format %s\n", argv[1] );
    exit( 1 );
  }
  fclose( f );

  init_X( &disp, &map );
  Trim_init( disp, map );

  win.xpos = XW, win.ypos = YW;
  win.vxmin = 0, win.vymin = 0;
  win.width = WW, win.height = HW;
  win.flag = _TRIM_DEFPIX;
  new_window( disp, &win, &gc, map );

  visual_do( disp, &win, gc, data );

  free( data[0]->surface.pt );
  free( data[0]->surface.seg );
  free( data[0]->surface.pol );

  Trim_close();
  XFreePixmap( disp, win.pixid );
  XDestroyWindow( disp, win.winid );
  XFreeGC( disp, gc );
  XCloseDisplay( disp );
  return 0;
```

```
}


/*---------------- local functions ------------------------*/

static
void init_X( Display **disp, Colormap *map )
{
  if ( (*disp=XOpenDisplay(""))==NULL ) {
    fprintf( stderr, "xmview: Cannot open display\n" );
    exit( 1 );
  }
  *map = DefaultColormap( *disp, DefaultScreen(*disp) );
  color[0] = _trim_grey[0];
  color[1] = _trim_grey[_TRIM_NGREY-1];
}


static
void new_window( Display *disp, TWin_t *win, GC *gc, Colormap clmap )
{
  Window root;
  XSizeHints win_h;
  XSetWindowAttributes att;

  root = DefaultRootWindow( disp );
  *gc = XCreateGC( disp, root, 0, 0 );
  win_h.x = win->xpos, win_h.y = win->ypos;
  win_h.width = win->vxmin+win->width-1, win_h.height = win->vymin+win->height-1;
  win_h.flags = USPosition | PSize;
  att.backing_store=Always;
  win->pixid = _TRIM_NO_ID;
  win->winid = XCreateSimpleWindow( disp, root, win_h.x, win_h.y,
    (unsigned int)win_h.width, (unsigned int)win_h.height, 10,
    color[1], color[0] );
  XChangeWindowAttributes( disp, win->winid, CWBackingStore, &att );
  XSetStandardProperties( disp, win->winid, "xmview", NULL, None, 0, 0, &win_h );
  XSetWindowColormap( disp, win->winid, clmap );
  win->pixid = XCreatePixmap( disp, win->winid,
    (unsigned int)win_h.width, (unsigned int)win_h.height,
    (unsigned int)DefaultDepth( disp, DefaultScreen( disp ) ) );
  win->id = (win->flag==_TRIM_DEFWIN) ? win->winid : win->pixid;
  XSelectInput( disp, win->winid, KeyPressMask );
  XMapRaised( disp, win->winid );
```

```
  XFlush( disp );
}

static
Bool new(int v1, int v2, Poledge_t *seg, int nseg)
{
int i;
  for (i =1; i<=nseg; i++)
   if (seg[i][0]==v1)
    if (seg[i][1]==v2)
        return FALSE;

  return TRUE;
}

static
BOOL read_surface_data( FILE *f, Trimmed_surface_t **data )
{
  point_r3 *ver, *pt;
  Poledge_t *seg;
  Polvert_t *pol;
  int c,n,npt=0,npol=0,nseg=0;
  int i,j,v[3],temp;

printf("\n We are making Vertex, Edge and Face structure ...\n");
  *data = (Trimmed_surface_t *)malloc( sizeof(Trimmed_surface_t) );
  MEMORY_FAULT( *data==NULL );
  (*data)->surface.pt=pt=(point_r3 *)malloc( (MAXVER+1)*sizeof(point_r3) );
  MEMORY_FAULT( (*data)->surface.pt==NULL );
  (*data)->surface.seg=seg=(Poledge_t *)malloc( (MAXSEG+1)*sizeof(Poledge_t) );
  MEMORY_FAULT( (*data)->surface.seg==NULL );
  (*data)->surface.pol=pol=(Polvert_t *)malloc( (MAXFACE+1)*sizeof(Polvert_t) );
  MEMORY_FAULT( (*data)->surface.pol==NULL );

  REMARK( f );

  while( (c=fgetc(f))=='V' && !feof(f) ) {
    ungetc( c, f );
    GET_VERTNO( f, n );
    GENERIC_FAULT( n>MAXVER, "xmview: too many vertices" );
    ver = &pt[n];
    GET_VERTEX( f, ver->x, ver->y, ver->z );
    if( n>npt )
      npt = n;
```

```
  }
  if( !feof(f) ) ungetc( c, f );
  while( (c=fgetc(f))=='F' && !feof(f) ) {
    ungetc( c, f );
    GET_FACENO( f, n );
    GENERIC_FAULT( npol==MAXFACE, "xmview: too many faces" );
    npol++;
    GET_FACE( f, pol[npol][0], pol[npol][1], pol[npol][2] );
    pol[npol][3] = 0;

    for (i=0; i<=2; i++)
     v[i]=pol[npol][i];
    /* sort the face verteces */
    for (i=0; i<2; i++)
     for (j=1; j<=2; j++)
       if (v[i]>v[j])
        {temp=v[i]; v[i]=v[j]; v[j]=temp;}
    /* make the edge structure */
    if (new(v[0],v[1],seg,nseg))
     {nseg=nseg+1;
      seg[nseg][0]=v[0];
      seg[nseg][1]=v[1];
     }
    if (new(v[1],v[2],seg,nseg))
     {nseg=nseg+1;
      seg[nseg][0]=v[1];
      seg[nseg][1]=v[2];
     }
    if (new(v[0],v[2],seg,nseg))
     {nseg=nseg+1;
      seg[nseg][0]=v[0];
      seg[nseg][1]=v[2];
     }
  }
  (*data)->npt = npt;
  (*data)->nseg = nseg;
  (*data)->npol = npol;
  printf("\n Verteces: %d; Edges: %d; Faces: %d. \n",npt,nseg,npol);
  return TRUE;
}


static
BOOL new_cop( point_r3 *cop, point_r3 *cs, int dir )
```

```
{
  point_r3 copt;
  REAL ro,teta,fi;

  copt.x = cop->x - cs->x;
  copt.y = cop->y - cs->y;
  copt.z = cop->z - cs->z;

  ro = sqrt(SQR(copt.x)+SQR(copt.y)+SQR(copt.z));
  teta = atan2(copt.y,copt.x);
  fi = acos(copt.z/ro);

  switch(dir) {
  case 1:
    fi -= M_PI/ROTSTEP;
    if (fi<0.1) return FALSE;
    break;
  case 2:
    if (teta>2*M_PI) teta -= 2*M_PI;
    teta += M_PI/ROTSTEP;
    break;
  case 3:
    fi += M_PI/ROTSTEP;
    if (fi>3.1) return FALSE;
    break;
  case 4:
    if (teta<0) teta += 2*M_PI;
    teta -= M_PI/ROTSTEP;
    break;
  }
  cop->x = cs->x+(REAL)(ro*sin(fi)*cos(teta));
  cop->y = cs->y+(REAL)(ro*sin(fi)*sin(teta));
  cop->z = cs->z+(REAL)(ro*cos(fi));
  return TRUE;
}


static
void visual_do( Display *disp, TWin_t *win, GC gc, Trimmed_surface_t **data )
{
  XEvent xevent;
  Obj_attrib_t attr;
  Visual_par_t vis;
  View_par_t view;
```

```
BOOL quit=FALSE;
Bbox_t bb;
REAL cr;
point_r3 cop_o;
BOOL cop_flag=TRUE;

printf( "\nKeys: LEFT, RIGHT, UP, DOWN -rotate object\n" );
printf( "      KP4, KP6, KP8, KP2    -object point\n" );
printf( "      KP+                   -zoom in\n");
printf( "      KP-                   -zoom out\n");
printf( "      w (Wire Frame)      d (depth cueing)\n");
printf( "      h (Hidden Line)     f (Flat Shading)\n");
printf( "      g (Gouraud Shading) p (Phong Shading) \n");
printf( "      ESC                   -exit program\n");

Trim_surface_bbox( data[0], &bb);
cr = 0.5 * sqrt(SQR(bb.xmax-bb.xmin) + SQR(bb.ymax-bb.ymin) +
SQR(bb.zmax-bb.zmin));
view.cs.x = (bb.xmax+bb.xmin)/2.0;
view.cs.y = (bb.ymax+bb.ymin)/2.0;
view.cs.z = (bb.zmax+bb.zmin)/2.0;
view.cop.x = view.cs.x+2*cr;
view.cop.y = view.cs.y;
view.cop.z = view.cs.z+cr;
view.r = cr;
cop_o.x = view.cop.x - view.cs.x;
cop_o.y = view.cop.y - view.cs.y;
cop_o.z = view.cop.z - view.cs.z;
view.di = 0.0;
view.da = view.r/2.0;
view.ox = view.oy = 0.0;

Trim_default_attrib( &attr );
Trim_default_visual( &vis );
vis.mode = _TRIM_PHONG;
vis.pos.teta = 0.0;

Trim_visual_create( win, &vis, &attr, data, 1, &visid );

do {
  if( cop_flag ) {
    XSetForeground( disp, gc, color[0] );
    XFillRectangle( disp, win->pixid, gc, win->vxmin, win->vymin,
    win->width, win->height );
```

```
  Trim_visual_do( visid, &view );

  XCopyArea( disp, win->pixid, win->winid, gc, 0, 0,
  win->width, win->height, 0, 0 );
  XFlush( disp );
}
do {
  XNextEvent( disp, &xevent );
} while( (xevent.type!=KeyPress) );
switch( XLookupKeysym( &xevent.xkey, 0 ) ) {
case XK_Escape:
  quit = TRUE;
  break;
case XK_w:
  vis.mode = _TRIM_WIREFRAME;
  Trim_visual_create( win, &vis, &attr, data, 1, &visid );
  break;
case XK_d:
  vis.mode = _TRIM_DEPTHCUEING;
  Trim_visual_create( win, &vis, &attr, data, 1, &visid );
  break;
case XK_h:
  vis.mode = _TRIM_HIDDENLINE;
  Trim_visual_create( win, &vis, &attr, data, 1, &visid );
  break;
case XK_f:
  vis.mode = _TRIM_FLAT;
  Trim_visual_create( win, &vis, &attr, data, 1, &visid );
  break;
case XK_g:
  vis.mode = _TRIM_GOURAUD;
  Trim_visual_create( win, &vis, &attr, data, 1, &visid );
  break;
case XK_p:
  vis.mode = _TRIM_PHONG;
  Trim_visual_create( win, &vis, &attr, data, 1, &visid );
  break;
case XK_Left:
  cop_flag = new_cop( &view.cop, &view.cs, 4 );
  break;
case XK_Right:
  cop_flag = new_cop( &view.cop, &view.cs, 2 );
  break;
```

```
    case XK_Up:
      cop_flag = new_cop( &view.cop, &view.cs, 1 );
      break;
    case XK_Down:
      cop_flag = new_cop( &view.cop, &view.cs, 3 );
      break;
    case XK_KP_Add:
      cop_flag = TRUE;
      view.da -= view.r/20.0;
      break;
    case XK_KP_Subtract:
      cop_flag = TRUE;
      view.da += view.r/20.0;
      break;
    case XK_KP_Left:
      cop_flag = TRUE;
      view.ox -= view.r/20.0;
      break;
    case XK_KP_Right:
      cop_flag = TRUE;
      view.ox += view.r/20.0;
      break;
    case XK_KP_Up:
      cop_flag = TRUE;
      view.oy += view.r/20.0;
      break;
    case XK_KP_Down:
      cop_flag = TRUE;
      view.oy -= view.r/20.0;
      break;
    default:
      cop_flag = FALSE;
      break;
    }
  } while ( !quit );

  Trim_visual_destroy( visid );
}
```

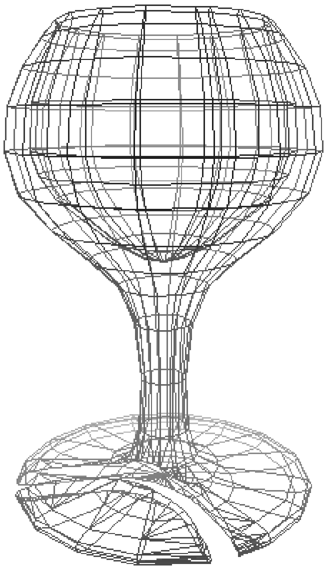

Figure 4.1: Trimmed NURBS surface visualized with the *xtreeview* program

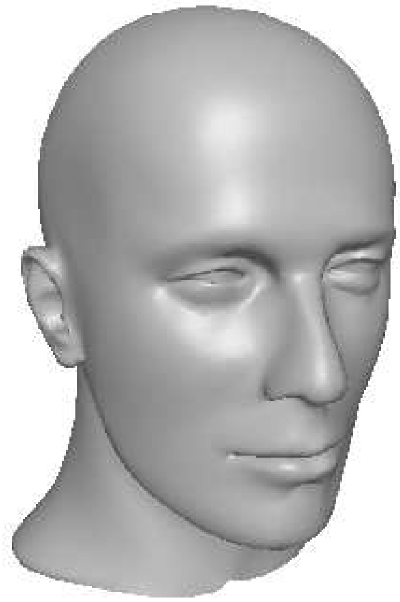

Figure 4.2: 3D mesh visualized with the *xmview* program

Bibliography

[XCMODEL00] G.Casciola, *xcmodel*: a system to model and render NURBS curves and surfaces; User's Guide - Version 1.0, Progetto MURST: "Analisi Numerica: Metodi e Software Matematico", Ferrara (2000), http://www.dm.unibo.it/~casciola/html/xcmodel.html