University of Bologna - Department of Mathematics

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

descriptor **library** **Programming Guide - Version 3.0**

G. Casciola

Department of Mathematics
University of Bologna

Bologna 2007

Abstract

This report presents the *descriptor* library. It is designed to describe a virtual 3D scene by making a scene-graph. From this scene-graph it is possible to obtain a photorealistic image of the 3D scene using a ray tracing algorithm.

G. Casciola
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 is *descriptor* ?

descriptor is a library designed to describe a 3D scene. Its functions must be called up using a program in C language. They make a graph of the scene that can be saved on a file. From this scene-graph it is possible to obtain a realistic view of the 3D scene using a ray tracing algorithm through hrayt (see [XCRAYT07]). This manual describes the functions to insert and position objects within the scene, to create attributes and textures to link them to objects and to create and position light sources.
Recently, descriptor library has been extended to design little animations.

Chapter 2

Classes of objects

We will define the following classes which enable the functions to be used effectively. Every item of data is defined as ITEM and always belongs to one of the following classes:

A primitive, list, hierarchy;

B light, primitive, list, hierarchy;

C attribute.

where items of class A can have an attribute, items of class B can be trandformed and items of class C are attributes.

primitive: consists in a trimmed or untrimmed NURBS surface or in a NURBS object (a set of surfaces).

list: is a list of primitives that describes how primitives have been clustered.

hierarchy: is a list of lists (graph) that describes how primitives and lists have been clustered; this organization simplifies a lot of operations and allows the hrayt program, operating on the model, to perform efficiently.

light: different types of light sources;

attribute: solid and color properties for objects described geometrically by primitives.

The following is a list of functions included in library along with a short description.

Chapter 3

descriptor library functions

3.1 Item instantiation

```
ITEM read_nurbs(char *filename)
```

reads a NURBS surface (.db file) or a trimmed NURBS surface (.dbe file); the parameter is the filename.

```
ITEM create_nurbs(int order_u,
                  int order_v,
                  int num_vertices_u,
                  int num_vertices_v,
                  VEC_vector_t **vertices,
                  VEC_real_t **coefficients,
                  VEC_real_t *knots_u,
                  VEC_real_t *knots_v,
                  BOOLEAN is_surface,
                  BOOLEAN normal_inside,
                  Trimming_tree_t trimming_tree,
                  char *name)
```

creates a NURBS surface or a trimmed NURBS surface with the given parameters.

```
ITEM read_obj(char *filename,
              char *name)
```

reads a NURBS object (.obj file); the parameters are the object file name and the list name. Returns the list identifier.

```
    ITEM create_copy(ITEM item,
                     char *name)
```

creates a copy of `item` and its name in the hierarchy will be `name`; every kind of ITEM can be duplicated. Returns the new identifier.

```
    ITEM create_list(ITEM item,
                     char *name)
```

creates a new list with only one element; the list name in the hierarchy will be `name`. `item` must belong to class A. Returns the list identifier.

```
    ITEM insert_in_list(ITEM list,
                        ITEM item)
```

inserts `item` in the list identified by `list`; returns the new list identifier. `item` must belong to class A. Note that `item` is just inserted in the list and not duplicated.

```
    BOOLEAN save_scene(char *name,
                       char *title,
                       ITEM hierarchy)
```

saves the model of the scene (`hierarchy`) on the file `name` with extension .md; if the file `name` already exists, it will be ovewritten. `title` is the name used as header in the scene graph file `name.md`. `hierarchy` is the name list of lists defined; it is used in *XCRayt* environment.
Returns TRUE if it is all ok, otherwise FALSE.

```
    BOOLEAN save_scene_ani(int nframe,
                           char *ani_name,
                           char *name,
                           char *title,
                           ITEM hierarchy)
```

Create the file `ani_name` with extension .ani containing the animation defined by `nframe` frames. Saves the current model of the scene (`hierarchy`) on the file `name` with extension .md; if these files already exist, they will be ovewritten. `title` is the name used as header in the scene graph file `name.md`. `hierarchy` is the name list of lists defined; it is used in *XCRayt* environment.
Returns TRUE if it is all ok, otherwise FALSE.

3.2 Transformations

A set of standard transformation functions is provided to translate, scale, rotate and shear objects. The name functions that end with `abs` are absolute transformations, the others are relative transformations i.e. composed with the previous transformation. All the following functions need a class B item.

```
BOOLEAN set_scale(ITEM item,
                  VEC_real_t sx,
                  VEC_real_t sy,
                  VEC_real_t sz)
BOOLEAN set_scale_abs(ITEM item,
                      VEC_real_t sx,
                      VEC_real_t sy,
                      VEC_real_t sz)
```

scale `item` by `sx`, `sy` and `sz` along the respective axes. The scale is relative to the origin of the coordinates system.

```
BOOLEAN set_translate(ITEM item,
                      VEC_real_t tx,
                      VEC_real_t ty,
                      VEC_real_t tz)
BOOLEAN set_translate_abs(ITEM item,
                          VEC_real_t tx,
                          VEC_real_t ty,
                          VEC_real_t tz)
```

translate `item` by `tx`, `ty` and `tz` along the respective axes.

```
BOOLEAN set_x_rotate(ITEM item,
                     VEC_real_t angle,
                     int deg_fg)
BOOLEAN set_x_rotate_abs(ITEM item,
                         VEC_real_t angle,
                         int deg_fg)
BOOLEAN set_y_rotate(ITEM item,
                     VEC_real_t angle,
                     int deg_fg)
BOOLEAN set_y_rotate_abs(ITEM item,
                         VEC_real_t angle,
                         int deg_fg)
BOOLEAN set_z_rotate(ITEM item,
                     VEC_real_t angle,
                     int deg_fg)
BOOLEAN set_z_rotate_abs(ITEM item,
                         VEC_real_t angle,
                         int deg_fg)
```

rotate `item` by an angle around the x axes (y or z); if `deg_fg` = 0, the angle is measured in radiants, otherwise in degrees.

```
     BOOLEAN set_x_shear(ITEM item,
                         VEC_real_t a,
                         VEC_real_t b)
     BOOLEAN set_x_shear_abs(ITEM item,
                             VEC_real_t a,
                             VEC_real_t b)
     BOOLEAN set_y_shear(ITEM item,
                         VEC_real_t a,
                         VEC_real_t b)
     BOOLEAN set_y_shear_abs(ITEM item,
                             VEC_real_t a,
                             VEC_real_t b)
     BOOLEAN set_z_shear(ITEM item,
                         VEC_real_t a,
                         VEC_real_t b)
     BOOLEAN set_z_shear_abs(ITEM item,
                             VEC_real_t a,
                             VEC_real_t b)
```

shear `item` by `a` and `b` with respect to the x axes (y or z); for example, `set_x_shear` transforms P = (x, y, z) in Q = (x', y', z') where: x' = x, y' = y + ax, z' = z + bx.

```
     BOOLEAN set_transform(ITEM item,
                           MAT_matrix_t mat)
```

transforms `item` by the transformation matrix `mat`. `mat` is a 4x3 matrix and represents a 4x4 transformation matrix where the last colum is (0, 0, 0, 1).

3.3 Attributes

Attributes are considered items and can be created, manipulated and linked to objects.

```
     ITEM create_attribute(char *name)
```

creates a new attribute with the default values listed below. A return value equal to UNDEF_ITEM indicates an error.

```
void set_attribute(ITEM item,
                   ITEM attribute)
```

sets the attribute for `item` with `attribute`.

```
BOOLEAN set_color(ITEM item,
                  float r,
                  float g,
                  float b,
                  float pa,
                  float pd)
```

sets the color in RGB coordinates and the proportion of ambient and diffuse reflection. The parameters `r`, `g` and `b` determine the color. `pa`=0 means that no ambient light is reflected, `pa`=1 means that all ambient light is reflected. `pd` controls the proportion of diffuse reflection and its meaning is similar to that of `pa`.

```
BOOLEAN set_reflectivity(ITEM item,
                         float ps,
                         float n)
```

sets the proportion of specular reflected light (`ps`) and Phong's exponent (`n`) for highlighting from direct light sources. the greater the exponent, the narrower the highlight.

Description	Name	Default	Range
r, g, b		1, 1, 1	[0,1][0,1][0,1]
ambient reflection	pa	0.3	[0,1]
diffuse reflection	pd	0.7	[0,1]
specular reflection	ps	0	[0,1]
Phong's specular exponent	n	0	≥ 0
max trasparency	maxt	0	[0,1]
min trasparency	mint	0	[0,1]
trasparency power factor	tpwr	0	≥ 0
refraction index	ri	1	see Table 2

Table 3.1: attribute

```
BOOLEAN set_trasparency(ITEM item,
                        float maxt,
                        float mint,
                        int tpwr,
                        float ri)
```

sets the transparency and the index of refraction `ri` (see table 2). The parameters specify a maximum (`maxt`) and a minimun (`mint`) transparency and a transparency power factor (`tpwr`). The transparency for a given point is determined according to the following formula:

$$t = (maxt - mint) * (1 - (1 - cos(alpha))^{tpwr}) + mint$$

Transparency t varies over an object according to the *alpha* angle between the incident vector and the normal to the surface object at the point considered. If *tpwr* is set to zero transparency will not vary across the object. Values of *tpwr* that are near 5 or 6 already simulate thin objects.

Materials	Refraction Index
ethyl alcool	1.36
water	1.33
air (1 atm, 20^o)	1.0003
bisulphide of carbon	1.63
calcite	1.486 - 1.658
diamond	2.42
dolomite	1.5 - 1.681
ethane	1.36
ice	1.309 - 1.313
polystyrene	1.55
quartz	1.544 - 1.553
molten quartz	1.46
sodium chloride	1.53
siderite	1.635 - 1.875
sugar in water 30^o	1.38
sugar in water 80^o	1.49
empty space	1.0
glass crown	1.52
glass (dense flint)	1.66
glass (flint)	1.72 - 1.89
wurtzite	2.356 - 2.378
sapphire	1.77

Table 3.2: refraction index for some materials

3.4 Textures

A procedural texture is an analytical function defined in 3D space, that is, a function which assigns some visual property to every point in a space. Thus, when an object is placed in that space, it acquires visual properties depending on where it is located. In other words, properties such as reflectivity, refractive, color, transparency, etc. defined for a surface can vary within the texture space. The texture space has its own coordinate system; the user decides where to place an object in this space using the transformation functions described above. The steps needed are:

1. create object;
2. transform object (scale, rotate, translate) to put into the texture space;
3. assign chosen texture to the object;
4. apply inverse transform to object;
5. position object, as desired, in the scene.

```
BOOLEAN set_angle_texture(ITEM item,
                          ITEM attribute1,
                          ITEM attribute2,
                          int sections)
```

divides the space into angular sections alternating between `attribute1` and `attribute2`. The angular sections radiate from the y-axis. `sections` specifies the number of sections (> 0) in which to divide the 360 degrees about the y-axis.

```
BOOLEAN set_cross_texture(ITEM item,
                          ITEM attribute1,
                          ITEM attribute2,
                          double x_wid,
                          double y_wid)
```

is defined by the x and z coordinates of an object, thus no change occurs along the y-axis. The cross texture is defined as a bar of width `x_wid`, which lies parallel to the x-axis, and a bar of width `z_wid` which lies parallel to the z-axis. `x_wid` and `z_wid` must belong to [0,1]. These bars are defined within the positive unit square, but the texture space coordinates are first absolute-valued and then calculated modulo to 1. Thus, the resulting texture is outlined squares repeated to infinity, with the x and z axes having bars which are twice as thick.

```
BOOLEAN set_cube_texture(ITEM item,
                         ITEM attribute1,
                         ITEM attribute2)
```

divides the texture space into cubes, alternating two different attributes. This texture is defined only for the positive octant of the space. Cube sides are parallel to the coordinate axes and have a length equal to unity.

```
BOOLEAN set_gradient_rand_texture(ITEM item,
                                  ITEM attribute1,
                                  ITEM attribute2,
                                  double axis,
                                  double resolution,
                                  double start,
                                  double end)
```

sets a texture that randomly chooses between `attribute1` and `attribute2`, but with a distribution along a gradient in the axis specified by `axis` (0 means x-axis, 1 y-axis and 2 z-axis); `resolution` is the range for the rand function. `start` and `end` specify, along the given axis, where this texture is applied.

```
BOOLEAN set_layer_texture(ITEM item,
                          ITEM attribute1,
                          ITEM attribute2)
```

divides the texture space into layers parallel to the xz plane, each with a thickness of one. The layers alternate the two attributes.

```
BOOLEAN set_octant_texture(ITEM item,
                           ITEM attribute1,
                           ITEM attribute2)
```

divides the texture space into 8 octants alternating the two attributes.

```
BOOLEAN set_shade_texture(ITEM item,
                          ITEM attribute1,
                          ITEM attribute2,
                          double axis,
                          double start,
                          double end)
```

makes an interpolation between the colors of `attribute1` and `attribute2`. The interpolation is calculated along the axis specified by `axis` (0 means the x-axis, 1 y-axis and 2 z-axis). `start` and `end` specify where to start with `attribute1` and end with `attribute2`.

```
BOOLEAN set_triangle_texture(ITEM item,
                             ITEM attribute1,
                             ITEM attribute2,
                             double x_wid,
                             double z_wid)
```

is defined by the x and z coordinates of an object, thus no changes occur along the y-axis. The triangle texture is defined as a rectangle of width `x_wid` along the x-axis and width `z_wid` along z-axis. `x_wid` and `z_wid` must belong to [0,1]. The rectangle is divided into two by a diagonal line, with `attribute1` on one side and `attribute2` on the other. This rectangle is repeated to infinity.

```
BOOLEAN set_weighted_rand_texture(ITEM item,
                                  ITEM attribute1,
                                  ITEM attribute2,
                                  ITEM attribute3,
                                  ITEM attribute4,
                                  int p1,
                                  int p2,
                                  int p3,
                                  int p4)
```

sets a texture that randomly chooses attributes 1, 2, 3 and 4, but each attribute has a certain probability of being chosen. Each attribute probability is specified by its corresponding p number: `p1`, `p2`, `p3` and `p4`. The probability can be calculated by its probability number divided by the sum of all the probability numbers.

```
BOOLEAN set_wood_texture(ITEM item,
                         ITEM attribute1,
                         ITEM attribute2)
```

is defined by the x and z coordinates of an object, thus no change occurs along the y-axis. The wood texture divides the texture space into concentric cylinders: the first has a radius equal to unity, the second a radius equal to two and so on, one more at every new cylinder. Attributes `attribute1` and `attribute2` are assigned alternatively to these cylinders.

3.5 Texture mapping

```
BOOLEAN set_projection_texture(ITEM item,
                               ITEM attribute1,
                                    char *image_name,
                                    int axis_flag,
                                    int r_flag,
                                    float pa,
                                    float pd)
```

This function allows an image to be attached to an object, like a postage stamp. The function needs an attribute in order to be defined, whose properties (color, reflection etc.) affect the final outcome. The image can be only a .hr file; to convert another image format to hr see *xmovie* package [XCRAYT07]. The object should be appropriately positioned in order to project the image. `r_flag` is a repetitive Boolean flag and `axis_flag` (0 for x-axis, 1 for y-axis and 2 for z-axis) sets the direction where to project. The `pa` and `pd` parameters are the contributions of ambient light and diffusion.

```
BOOLEAN set_domain_texture(ITEM it,
                           ITEM attribute1,
                           char *image_name,
                           float umin,
                           float umax,
                           float vmin,
                           float vmax,
                           int r_flag,
                           float pa,
                           float pd)
```

This function allows an image to be applied to an object in according to its parameterization. The function needs an attribute in order to be defined, whose properties (color, reflection etc.) affect the final outcome. The image can be only a .hr file; to convert another image format to hr see *xmovie* package [XCRAYT07]. `umin`, `umax`, `vmin` and `vmax` define the rectangular surface parametric subdomain where consider and map the image. `r_flag` is a repetitive Boolean flag. The `pa` and `pd` parameters are the contributions of ambient light and diffusion.

```
BOOLEAN set_border_texture(ITEM it,
                           ITEM attribute1,
                           ITEM attribute2,
                           double umin,
                           double umax,
                           double vmin,
                           double vmax)
```

This function alternates `attribute1` with `attribute2` on a surfaces in according to a rectangular parametric subdomain defined by texttttumin, `umax`, `vmin` and `vmax`. More precisely inside the rectangle uses `attribute2` and outside `attribute1`.

3.6 Bump mapping

```
BOOLEAN set_bump_texture(ITEM it,
                         ITEM attribute1,
                         char *image_name,
                         float umin,
                         float umax,
                         float vmin,
                         float vmax,
                         int r_flag,
                         float pa,
                         float pd)
```

This function allows a grey color image to be used as a two-dimensional bump map to angularly perturb the surface normal. The function needs an attribute in order to be defined, whose properties (color, reflection etc.) affect the final outcome. The image can be only a .hr file; to convert another image format to hr see *xmovie* package [XCRAYT07]. `umin`, `umax`, `vmin` and `vmax` define the rectangular surface parametric subdomain where consider the bump mapping. `r_flag` is a repetitive Boolean flag. The `pa` parameter allows to scale the perturbation and must be between 0 and 1. The `pd` parameter can be 1 or -1 enabling surface to appear as if were wrinkled or dimped.

3.7 Light sources

```
ITEM create_distant_light(VEC_real_t dx,
                          VEC_real_t dy,
                          VEC_real_t dz,
                          float r,
                          float g,
                          float b,
                          float intensity,
                          char *name)
```

creates a light source at an infinite distance from the scene with rays pointing in the direction given by the vector (`dx`,`dy`,`dz`), color defined in the RGB space by the triplet `r`, `g`, `b` and intensity equal to the value of the parameter `intensity`. `dx`, `dy` and `dz` must not all be zero at the same time. `intensity` must be greater than zero. `intensity` should be less than one or, at most, just over one. Returns the ITEM identifier.

```
void set_distant_light(VEC_real_t dx,
                       VEC_real_t dy,
                       VEC_real_t dz,
                       float r,
                       float g,
                       float b,
                       float i,
                       ITEM it)
```

allows to redefine the parameters of the point light with ITEM `it` and previously created.

```
ITEM create_point_light(VEC_real_t cx,
                        VEC_real_t cy,
                        VEC_real_t cz,
                        float r,
                        float g,
                        float b,
                        float intensity,
                        VEC_real_t range,
                        char *name)
```

creates a point light source centered at (`cx`,`cy`,`cz`), with color defined in the RGB space by the triplet `r`, `g`, `b` and intensity equal to the value of the parameter `intensity` at the source and fading linearly to zero at a distance equal to `range`. `intensity` must be greater than zero. If `range` is less than or equal to zero, the light intensity will not be dependent on the distance travelled.

```
void set_point_light(VEC_real_t cx,
                     VEC_real_t cy,
                     VEC_real_t cz,
                     float r,
                     float g,
                     float b,
                     float i,
                     VEC_real_t max_range,
                     ITEM it)
```

allows to redefine the parameters of the point light with ITEM `it` and previously created.

```
ITEM create_warn_light(VEC_real_t cx,
                       VEC_real_t cy,
                       VEC_real_t cz,
                       VEC_real_t dx,
                       VEC_real_t dy,
                       VEC_real_t dz,
                       float r,
                       float g,
                       float b,
                       float intensity,
                       VEC_real_t range,
                       int concentration,
                       VEC_real_t maximum_angle,
                       VEC_real_t flap[4],
                       char *name)
```

creates a light source centered at (`cx`,`cy`,`cz`), with color defined in the RGB space by the triplet `r`, `g`, `b` and intensity equal to the value of the parameter `intensity` at the source and fading linearly to zero at a distance equal to `range`. Moreover, `dx`, `dy` and `dz` are the components of a vector that specify a light direction in order to aim a light at a particular area of an object. `concentration` is used to define the way in which light intensity decreases away from the aim direction: a larger value means that the light decreases, when away from the aim direction, quicker than for smaller values. Finally `maximum_angle` and `flap` can be used to restrict the path of a light. `maximum_angle` defines a cone surrounding the light direction: the light intensity is set to zero if the angle between the aim direction and the direction of the ray to the point being illuminated is greater than the cone angle `maximum_angle`. `flap` is the plane equation:

$$flap[0]x + flap[1]y + flap[2]z + flap[3] = 0$$

Points on the same side of the light location with respect to the plane will be illuminated, while all the others will not. `dx`, `dy` and `dz` must define a non-null vector. `intensity` must be greater than zero. If the parameter `range` is less than or equal to zero, the light intensity won't decrease with distance. `concentration` must be non-negative. `angle` is given in degrees. `flap` must be a valid plane equation or be the null pointer; in the latter case no flap will be used to stop light.

```
void set_warn_light(VEC_real_t cx,
                         VEC_real_t cy,
                         VEC_real_t cz,
                         VEC_real_t dx,
                         VEC_real_t dy,
                         VEC_real_t dz,
                         float r,
                         float g,
                         float b,
                         float i,
                         VEC_real_t max_range,
                         int conc,
                         VEC_real_t angle,
                         int flap_on,
                         VEC_real_t fa,
                         VEC_real_t fb,
                         VEC_real_t fc,
                         VEC_real_t fd,
                         ITEM it)
```

allows to redefine the parameters of the warn light with ITEM `it` and previously created.

```
BOOLEAN set_ambient_light(float r,
                          float g,
                          float b,
                          float intensity)
```

sets the ambient light to have a color defined in the RGB space by the triplet `r`, `g`, `b` and intensity equal to the value of the parameter `intensity`. The ambient light is such that it illuminates an object equally from every direction. `intensity` must be greater than zero.

```
BOOLEAN set_background(float r,
                       float g,
                       float b,
                       BOOLEAN is_refl)
```

sets the scene background to have a color defined in the RGB space by the triplet `r`, `g`, `b`. `is_refl` gives a reflectent property to the background. `is_refl` is a boolean flag.

3.8 Animation Design

Recently, decriptor library has been extended to design little animations. This section describes the guide lines to follow to design a correct list of scene graphs. We remember that to describe a single scene we must insert and position objects within the scene, create attributes and textures and link them to the objects and create and position light sources. To design a scene graph of a list (frame list) we must understand what we must/can modify respect to the previous in the list.

3.8.1 Lights

A light must be initially created using one of the `create_xxx_light` functions and then it can be modified in position and parameters by using the `set_xxx_light` correspondent functions. Note that a light cannot be deleted; it is possible to act on the intensity parameter or on the ray of influence to reduce its effect.

3.8.2 Objects

An object must be loaded using one of the following functions: `read_nurbs`, `create_nurbs` or `read_obj`. The loaded objects will do part of the scene only if they will be inserted in it by the `inser_in_list` function. After an object was inserted in the scene removing it is not possible. This involves that in the design of an animation in which in the next frame we want to add objects it is possible to use the same list in which we insert the new ones, while if we want to remove objects in the next frame it will be necessary create a new one list with only the necessary objects.

3.8.3 Transformations

The geometrical transformations can be absolute or relative depending on whether the name of the function finishes with `_abs` or not. Several absolute transformations on the same object only make real the last one, while several relative transformations on the same object compose themselves.

3.8.4 Attributes and Textures

An attribute must be initially created and can be then modified with the specific functions. Using the `set_attribute` functions will be possible to apply it to an object. Since an object can have only an attribute, if more than one is applied, only the last application will be real.

Chapter 4

Error codes

In Table 4.1 is shown a list of possible errors from calls to functions in the *descriptor* library, together with their mnemonic names, which are defined in `descriptor.h`.

Value	Meaning
NOMEM	memory not avalible
ZERPAR	parameter value 0
CONSTR	constrain violated
DOMAIN	parameter value out of range
TYPE	item/attribute does not belong to the right class
IO	I/O file error

Table 4.1: error codes

Chapter 5

A programming example

The following is the sequence of instructions used to produce a marble chess tower, obtained with a texture mapping, a light blue reflecting background, a white ambient light and three light sources. Note the creation of the tower, which is not a whole surface, but a slice of a surface, inserted 4 times into the hierarchy and rotated each time. The loaded NURBS surface is the slice. The following is the commented code:

```
/* chess_tower.c */

#include "descriptor/descriptor.h"

#define DIM_CHESS (0.5)

int main(void) {

  ITEM objectB;
  ITEM Qtower,tower;

  /* background setting */
  set_background(0.6,0.7,0.8,TRUE);
  /* ambient light setting */
  set_ambient_light(1,1,1,0.7);
  /* distant light creation */
  create_distant_light(0,0,1, 1,1,1, 0.7, "distant_light");
  /* point light creation */
  create_point_light(35.68,5.21,7.49, 1,1,1,
                     0.7, 0, "point_light");
  /* spot light creation */
  create_warn_light(1,0,0, 0,1,0, 1,1,1, 0.8,
                    40, 5, 70, 1,0,0,1,5, "warn_light");
  /* attribute definition */
  objectB=create_attribute("black_chess");
  set_color(objectB,1,0.062,0.031,0.5,0.5);
  set_reflectivity(objectB,0.4,2);

  /* hierarchy initialization and construction */
  Qtower=read_nurbs("Qtower.db");
  tower=create_list(create_copy(Qtower,"slice_1"), "tower");
  set_z_rotate(Qtower,90,1);
  tower=insert_in_list(tower, create_copy(Qtower,"slice_2"));
  set_z_rotate(Qtower,90,1);
  tower=insert_in_list(tower, create_copy(Qtower,"slice_3"));
  set_z_rotate(Qtower,90,1);
  tower=insert_in_list(tower, create_copy(Qtower,"slice_4"));

  /* tower positioning to immerge into texture space */
  set_scale(tower,1.0/0.26,1.0/0.26,1);

  /* textute mapping */
  set_projection_texture(tower,objectB,
                         "marble_5_black.hr",
                         FALSE,FALSE,0.6,0.8);

  /* tower repositioning */
  set_scale(tower,0.26,0.26,1);
  /* tower positioning in the scene */
  set_translate(tower,DIM_CHESS/2,DIM_CHESS/2,0);
  set_translate(tower,0,DIM_CHESS*3,0);

  save_scene("chess_tower.md", "tower", tower);
  return(0);
}
```

The first part of the code is devoted to the description of the objects in the scene. They are all of the type ITEM, and therefore generic. They will later gain specificity, during the creation of the attributes:

```
  objectB=create_attribute("black_chess");
```

or in the creation of the hierarchy:

```
  Qtower=read_nurbs("Qtower.db");
  tower=create_list(create_copy(Qtower,"slice_1"), "tower");
```

This mode of scene description has the disadvantage of not immediately being able to visualise the scene itself, but only after the process of realistic rendering, that is very expensive in terms of time. The scene adaptation phase is therefore a very slow process. Another disadvantage is that C language must be used.

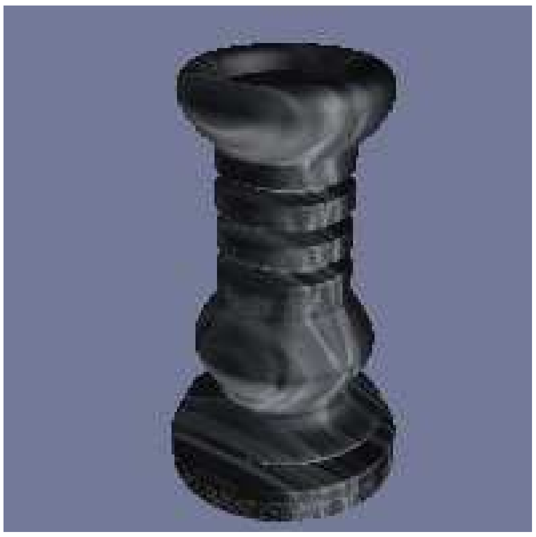

Figure 5.1: rendering obtained by the scene model produced with the presented C code

Chapter 6

Another programming example

The following is the sequence of instructions used to produce a marble chess tower animation. The scene is similar to the previous one; there is a base plane and a light source move around the tower producing an animated shadow on the base plane. The following is the commented code:

```
/* chess_tower_anim.c */

#include "descriptor/descriptor.h"

#define DIM_CHESS (0.5)

int main(void) {

  ITEM objectB,attr_plane;
  ITEM Qtower,tower,plane,scene;
  ITEM  DistantL,WarnL,PointL;
  float teta,xi,yi,xip1,yip1,ct,st;
  int i;

  /* background setting */
  set_background(0.6,0.7,0.8,TRUE);
  /* ambient light setting */
  set_ambient_light(1,1,1,0.7);
  /* distant light creation */
  DistantL=create_distant_light(0,0,1, 1,1,1, 0.7, "distant_light");
  /* point light creation */
  PointL=create_point_light(35.68,5.21,7.49, 1,1,1,
                   0.7, 0, "point_light");
  /* spot light creation */
```

```
WarnL=create_warn_light(1,0,0, 0,1,0, 1,1,1, 0.8,
                  40, 5, 70, 1,0,0,1,5, "warn_light");

/* attribute definition */
objectB=create_attribute("black_chess");
set_color(objectB,0.1,0.062,0.031,0.5,0.5);
set_reflectivity(objectB,0.4,2);

attr_plane=create_attribute("attr_plane");
set_color(attr_plane,0.1,0.75,0.1,0.5,0.5);

/* hierarchy initialization and construction */
tower=read_nurbs("tower.db");
set_attribute(tower,objectB);

set_scale(tower,1.0/0.26,1.0/0.26,1);

/* texture mapping */
set_projection_texture(tower,objectB,
                       "marble_5_black.hr",
                       0,FALSE,0.6,0.8);

/* tower repositioning */
set_scale(tower,0.26,0.26,1);

plane=read_nurbs("plane.db");
set_attribute(plane,attr_plane);

scene=create_list(tower, "scene");
scene=insert_in_list(scene, plane);

/* parameters initialization for anumation */
teta=6.28/15;
ct=cosf(teta);
st=sinf(teta);
xi=8.0;
yi=0.0;
/* loop: the point_light rotates around the object */
/* render with shadow on */
for (i=0; i<16; i++)
{
  xip1=xi*ct-yi*st;
  yip1=yi*ct+xi*st;
  set_point_light(xip1,yip1,7.49, 1,1,1,
                     2.0, 0, PointL);
    /* save the i-th of 16 frames */
    save_scene_ani(16,"chess_tower_anim.ani",
                   "chess_tower_anim", "scene", scene);
    xi=xip1;
    yi=yip1;
  }
  return(0);
}
```

CHAPTER 7

How to compile

This section describes how to compile, link and execute a program in C language with Unix Operating System. A program, to call the *descriptor* library functions, should include the file `descriptor.h`, that is, it should have the line

```
#include "descriptor/descriptor.h"
```

An Imakefile is provided to build a specific Makefile on your computer and operating system. Note that the C code file name must be update manually in the Imakefile (chess_tower for the above example). Give the following commands:

```
xmkmf
```

now it is possible to compile and execute the program with:

```
make
```

What happens is that from a C code (chess_tower.c), an exe-file is produced (chess_tower), whose execution creates an .md description file of the scene (chess_tower.md).

If errors occur during execution, these will be marked with the descriptor_errno value.

The use of a Makefile.ok present in every models subdirectory is advised for that who do not have imake available.

List of Figures

Bibliography

[XCRAYT07] G.Casciola *XCRayt*: the scene descriptor; User's Guide - Version 2.0, 2007
http://www.dm.unibo.it/∼casciola/html/xcmodel.html.