RTED 0.6.1

RTED (RT Editor) is a freeware program. Read file “licence.wri”.

If you are interested on this program, you have created some good images or you have found some bugs please send a mail to “sergio.per@iol.it”, or you can write to:

Sergio Perani
Via Filippo Turati, 27
05100 Terni (TR) , Italy

RT email newsletter

You can subscribe to RT email newsletter.
Subscribe to RT: The best way to hear about new content and preview upcoming events on RT is to subscribe to RT email newsletter.
You can subscribe sending the message "SUBSCRIBE" to "sergio.per@iol.it".
You can unsubscribe if you send the message "NO SUBSCRIBE".

Download

The home page for RT program is :

http://www.geocities.com/SiliconValley/3526

if the page is moved you can ask me the new url.

You can download the last version of the program at SimTel:

http://www.coast.net/SimTel/win95/graphics.html

OR

ftp://ftp.coast.net

OR in any SimTel mirror site.

The program can work with:
processor :386
OS: Windows 3.1 + Win32s
RAM: 4 Meg

But it works better with:
processor: 486 or better
RAM: 8 Meg or better
Video: true color display

Introduction

The program RT is a ray tracer with the following features:

Ray Casting and Ray Tracing
Antialiasing
Depth bitmap for stereogram
VRML 1.0 output
Texture mapping (from bitmap Windows .bmp with 24 bit color)

Models of illumination

Phong

Sources of light (Lamp)

Point
Cone
Area

Geometric Models

Box
Sphere
Cylinder
CSG General
Boundary Representation (with Phong interpolation)
Bezier mesh
Polyline

The program reads a text file,which describes the scene ed returns a bitmap file.

How the program works

The program RT gets the scene description in a file .rt and creates the 24 bit image in a file .bmp. RT uses geometric models and textures.
The scene description is a text file divided in some sections which describe the 3d view, the texture used, the object materials, the objects and the light sources.

Limits of the raytracer

The program works with any number of object and of light source. There are some constraints to follow:

you must use only closed objects. A closed object is for any line in the space, the intersection with the object consists or in no intersection point or in a even number of intersection points. It is possible to create objects non closed (es. with the brep). This object are wrong. The program doesn' t control if an object is closed.

Compenetrations between objects are forbidden. Two objects cannot share the same point of the space. The program doesn' t control this. If you want the same effect of a compenetration use a csg node of type union.

User Manual

RTED (RT Editor) 0.6.0

Features

- trivial but complete undo
- mixed editing (editor and text)
- read RT file format
- on screen selection for objects and view
- editing tools

REMEMBER: Don' t work with unnamed file, always save it. If you don' t have a named file, you haven' t a directory where to get model and to save undo.

In the title bar, you will see:
RT Scene Editor - c:\Dir\foo.rt [actuallay]

actuallay is the name of the layer that is now selected. When you start to create a scene, you will start from layer default.
The layers are usefull when you work with a great number of objects. No problem, create a new layer and put in this all the object you need to edit. The layers are usefull also to create a group of objects. Follow this:
1) Create a new layer with the same name of the group
2) Add in this layer the objects that you want in the group
3) Use command Layer|Group. The program will create the group and will put it in default layer.
You can open a group with command Layer|Open Group. The program will recreate the layer.
If you open a group you will not loose the trasformation (rotation, scale, translation) applied to group. The program will keep an empty group to remember of them when you will want to rebuild the group.

Read first the file “Edtut.wri”, the editor tutorial, where is explained how to build the first scene.

COMMANDS:

<u>Left button of the mouse</u>

1) Select a view. You can see e little triangle in the top on the left in the selected view.
2) Select an object. If you click with left button on a object in the selected view you select the object.
You can edit some types of object if you select them.

<u>Right button of the mouse</u>

Active only in 3D views. click on objects and view (VUP, VPN, VRP, PRP).
If you select multiple object, a dialog box pops up and ask you an item.
If it starts with “$” it is an object, with an “*” is a lamp, with a “&” is a view value.
If you select an object or a lamp then will appear a submenu with action for objects or lamps:
translate, rotate, scale. If you select translate then click for new object position.

<u>File|Load ...</u>

You can load a .rt file.

<u>File|Save</u>

You can save actual .rt file. If the file has no name, no action will be made.

File|Save as...

You can save the actual file with a new name.
Remeber: save the new file in the same directory. If you must save in another, you must copy all the referenced file *.brp, *.bzr.

File|Exit

Exit the program.

Edit|Duplicate

Duplicate an object.

Edit|Object|Create

you must choose an object type. Then a specific dialog box will pop up. You must always give the name to the object.

- sphere
- brep
- box
- cylinder
- intersection
- subtraction
- union
- bezier
- polyline
- anim
- deformgraph

Look after to see how to modify polyline, anim and deformgraph.

Edit|Object|Modify

You must choose the object name from the list. A dialog box will pop up, where you can edit the object characteristics.

Edit|Object|Modify Comm

You must choose the object name from the list.
A dialog box will pop up. You can change the object name, its material and texture and apply the current mapping, if you have created one.

Edit|Object|Delete

You must choose the object name from the list.
The program will delete the object from the scene.

Edit|Object|Traslate

You must choose the object name from the list.
Choose the translation values for x,y,z axes.

Edit|Object|Rotate

You must choose the object name from the list.
Then you choose the rotation angle for x, y, and z axes.

Edit|Object|Scale

You must choose the object name from the list.
Then you choose the scale factor for x, y, and z axes.
Choose negative values for mirroring.

Edit|Object|Special

Special menu for object. Not all the objects have a special menu.

Edit|Object|Extra Info

You can change some extra caracteristic for objects.
style is the type of visualization: normal, fastview and box.
fastview is usefull for object with a great number of lines. With fastview the program show only a line every n times, where n is a value in the fastview attribute.
With box you will see only the box of the object.

Edit|Lamp|Create

you must choose an lamp type (only a lamp type for now).
Then a specific dialog box will pop up. You must always give the name to the lamp.
Now the program don' t check if two lamps have the same name!

- pointlamp
  give center and color (intensity).

- conelamp

Edit|Lamp|Modify

You must choose the lamp name from the list.
A dialog box similar to the "Edit|Lamp|Create" will appear.
There is no other way to change the lamp position for now.

Edit|Lamp|Modify Comm

You must choose the lamp name from the list.
A dialog box will pop up. You can change the object name.

Edit|Lamp|Delete

You must choose the lamp name from the list.
The lamp will be deleted from the scene.

Edit|Lamp|Special

Special menu for lamp. Not all the lamps have a special menu.

Edit|Material|Create

A dialog box will appear. Give the name of the new material, then the material editor will appear.
Look in manual or in tutorial for attribute meaning.

Edit|Material|Modify

You must choose the material name from the list.
The material editor dialog box will appare.
Look in manual or in tutorial for attribute meaning.

Edit|Material|Delete

You must choose the material name from the list.
The program will delete the material from the scene.

Edit|Mapping|Create

You must choose the mapping type from the list. A new mapping will be created. The mapping created will be the selected mapping.You can apply it to objects using the command “Edit|Object|Modify Comm”.

Edit|Mapping|Get from ...

Get the selected mapping from an object. You must choose from the object list.

Edit|Mapping|Delete

Delete the selected mapping.

Edit|Mapping|Show

Show the mapping on view windows, so you can edit it. You can edit only the mapping, to return to normal edit mode use command “Edit|Mapping|Hide”.

Edit|Mapping|Hide

Hide the mapping and return to normal mapping mode.

Edit|Texture|Create

You must choose a texture from the list. In the list there are the texture found in the texture path.

Edit|Texture|Delete

You must choose the texture name from the list. This command will not delete the texture from you disk but only from the scene.

Edit|View|General ...

Allow to change the view in a simpler way, and this will preserve the aspect ratio.
Dimx, Dimy are like in Edit|Param, but this change the window dimension too.
Focal is the focal distance (The distance from PRP to VRP).
High is the semi high of the view window.

Edit|View|VRP

Change the VRP position. You can change this with right button of the mouse.
Look in the tutorial for more information.

Edit|View|VUP

Change the VUP position. You can change this with right button of the mouse.
Look in the tutorial for more information.

Edit|View|VPN

Change the VPN position. You can change this with right button of the mouse.
Look in the tutorial for more information.

Edit|View|PRP

Change the PRP position.
Look in the tutorial for more information.

Edit|View|Window

Modify the window in the view plane.
Look in the tutorial for more information.

Edit|Param

You can edit varius parameters from the param section.
Look in the tutorial for more information.

Command|Redraw 3D

Redraw the front, top, left views. You can press “F5” key.

Command|Preview

Redraw the preview. You can press “F9” key.

Command|Zoom In

Zoom in the selected view window. This command does not work with the preview window. You can press the “i” key.

Command|Zoom Out

Zoom out the selected view window. This command does not work with the preview window. You can press the “o” key.

Command|Center

Allow to change the center of the front, top, left views.
Click with the left mouse in the new center of the window.

Command|Axes

Show or hide the axes.

Window|Single window

Show only the selected window.

Window|All windows

Allow to return to the edit windows.

Window|Show image

Show the last rendered image.

Window|Front view

Show only the front view.

Window|Top view

Show only the top view.

Window|Side view

Show only the side view.

Window|Preview view

Show only the preview view.

Action|Create Image

Create and display the image on the view.

Action|Create all frames

Create all the frames of a animation.

Animate|Go to frame ...

Select a frame of the animation, and enter in the animation mode.

Animate|NFrame

Set the number of frame in the animation.

Animate|End

Exit from the animation mode.

Help|About

Information about the program.

Layer|Create

Create a layer. You must enter the new name.

Layer|Delete

Delete a layer

Layer|Add to ...

Add an object to a layer. An object can be in more than a layer.

Layer|Remove from ...

Remove an object from a layer.

Layer|Group

Create a group from the selected layer.

Layer|Open group ...

Trasform a group in a layer.

Layer|Select

Select a layer. You will see only the object in the layer.

Insert|File RT

Ask for a .rt file. It will insert the .rt file in the actual scene.

Insert|File DXF

Ask for a .dxf file. It will insert the .dxf file in the actual scene. It generate .brp files and a .rt file in the scene directory.

Insert|File RAW

Ask for a .raw file. It will create a .brp file in the scene directory.

Option|General

You can change some editor values:

KeepEdge	speedup redraw, but needs more memory.
UseColor	use colors to draw object in wireframe mode.
DirectBox	show all the objects like boxs.
Background	the color of the background.
SelectedColor	the color of the selected object.
UseGrid	use the grid for polyline editing.
GridDim	dimension of the grid.
InvertY	invert the y axes on Top view.

Option|Interface

You can change some interface values:

DimPixel	pixel dimension, during image costruction.
	0 no draw
	1 normal
	2..6 bigger image
CBitmap	hold image in memory, after image creation.
MultiTask31	multitasking for Windows 3.1.

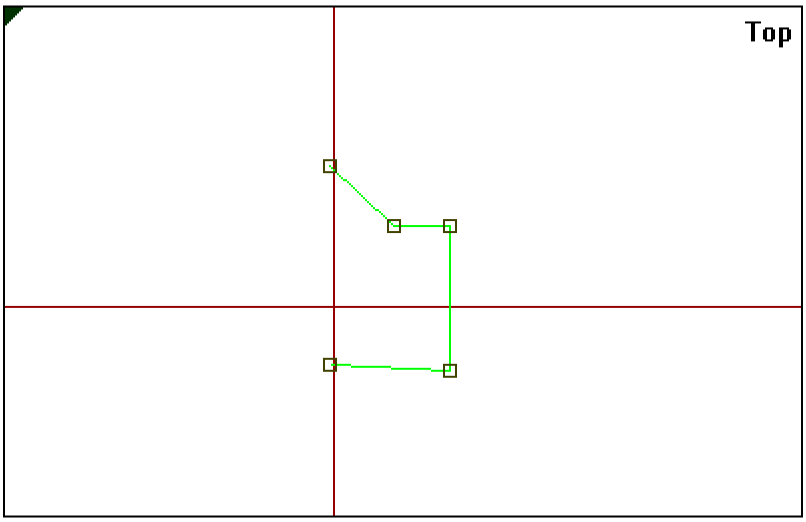
Editing Curves (polyline)

If you click with left button of the mouse on a curve in a selected view, you select the curve. The curve will change color, and its control points will appear on display (you will see a little square for any control point). If you click again far form the curve, you deselect it.
If on a selected curve you press the left button of the mouse in a control point, the square will become red., you have selected the control point. There are some actions you can do in this situation:
press key d’. This delete the control point.
press key a’. A new point control will be added in the cursor position.
press key m’. You move the control point position. The new position will be the same of the cursor.
You can use the grid (Option|General...) to limit the positions of the point.

The only curve you can edit is the polyline.
You can use the polyline in many way. You can use it to build brep objects. Remember to work always in the Top view!
It is possible to build rotation brep or sweep brep.

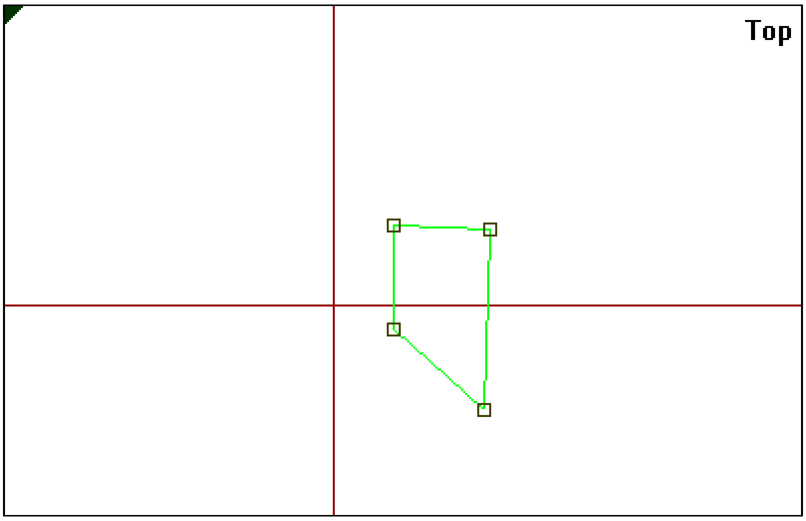
Rotation brep

To build a brep you need to rotate a polyline around an axis. The rotation axis will be the Y axis. To see the axis press the F12 key or use Command|Axis. The polyline can be open or close. (Use the special menu for open or close a polyline).
If the polyline is open you need that the first and the last point of the curve are on the Y axis, like in the picture.



The first and the last point can not be exactly on the axis but near; the program will move them for you.

The curve can be closed:



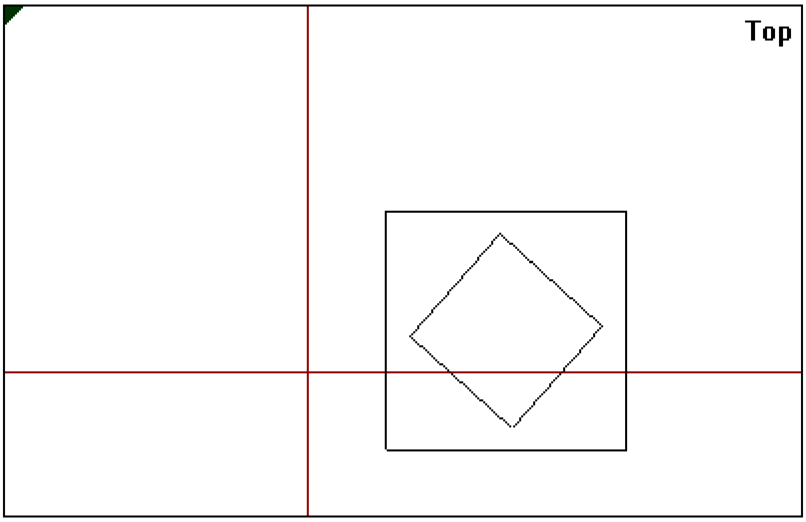
All the points must be all positive or all negative.

To make a rotation of a polyline you must choose “rotation” from the special menu of the polyline.
The program will ask for the name of the new object, the number of subdivisions and the angle of rotation.

Sweep brep

You may build a sweep brep only from a closed polyline. Choose “sweep” from the special menu of the polyline.
The program will ask for the name and the height of the new object.

It is possible to build more complex sweep object, with hole, following a path or with deformations.
Before you have to create a shape to sweep. This must be a group of polylines. Follow this procedure:
1) Create a new layer with the name of the shape (es. test_shape).
2) Select the new layer
3) Select the top view
4) Draw the external and the internal polyline (you must draw one external polyline and one or more internal polyline or no internal polyline).



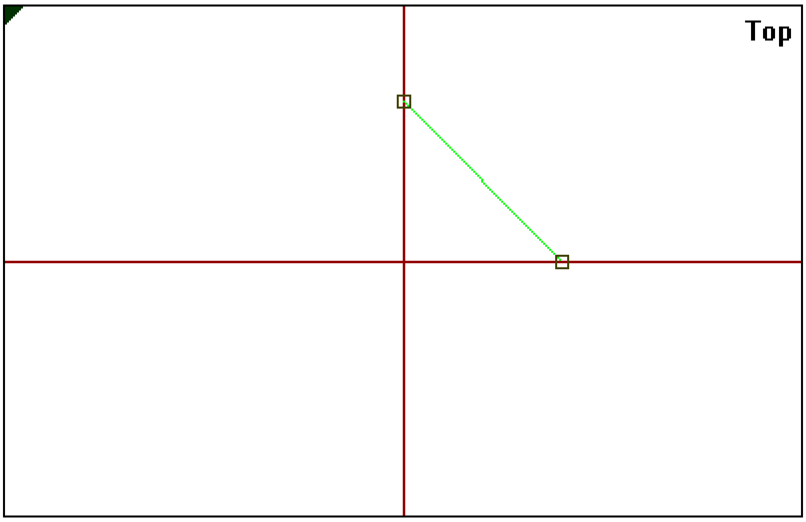
5) Use command Layer|Group to create group “test_shape”.
6) Now in the default layer, draw a polyline that will be the sweep path (in the front or side view) with name

"path1".
7) Select the group "test_shape" and call the special menu of the object.
8) Choose "Sweep".
9) In the dialog box give a name to the brep object, and choose "path1" in the path listbox.

At the end you will have a sweep brep.
You can also deform the shape when it follow the path. You need a deformation graph ("deformgraph").

Creating a deformation graph

A deformation graph is used for a general sweep operation, to deform the shape of the sweep.
You can work only in the Top view and only in the range 0 <= x <= 1. Draw a curve in the range like in the picture.



This curve start from point (0,1) to point (1,0). You can see it like a function f(x). The 1-value means an unmodified shape, greater than 1 a bigger shape and smaller than 1 a reduced shape. When the program creates the sweep , it divides the interval from 0 to 1 in as many parts as the times it uses the shape.
To create a deformation graph from this curve use command Edit|Object|Create and choose deformgraph.

A deformation graph can be in any layer, when you create a general sweep you can select it also if in a different layer.

UNDO FEATURE:

If you work with a .rt file which name is "name.rt", the program will create a directory "name.rtd", and will put some files in it. Inside there is the directory "editor" which contains files used by the scene editor. Inside there is a directory "UNDO". After every action RTED will put there a copy of the scene. This because the program can crash in every time so you will have you work save.

Definition of configuration file

The position of the configuration file of the program RTED is definited in file win.ini in the section [Rtw] in record Ini. Es.

[Rtw]
Ini = c:\rt\rtw.ini

Section view
    Definitions of the metods to display image during creation.

Dimpixel.
    Dimension of pixel of image showed during creation. 3 means that the image will occupate a grid of 3x3 pixel. You can use it when you create a small image.

Use Editor|Interface... to change this value.

CBitmap

If the value is 1 RTED keep a copy of the bitmap in memory.
Use Editor|Interface... to change this value.

Section Dir

Definition of directory used by the program.

TextureDir.

Directory where the program looks for the textures.

ModelDir.

Directory where the program looks for geometric models.

TUTORIAL (for RT 0.1.2 file format)

Introduction

This tutorial explains step by step how to create a scene that you can render with RT program. There are explained some advanced features too.
The scene is described by a text file; the extension used is “rt”.
You must write a .rt file using a text editor, like the dos edit, windows notepad or the editor which you prefer. You must be able to use one of it, to read a file and to save it.
That is not the only a way to create a .rt file. You may use the RT Scene Editor. It creates a .rt file in the same format explained here.

For all paragraphs in the form :

title (Tutxx.rt)

there is a .rt file which uses a specific characteristic. You can find that file in the “examples\tutorial” directory; so you can print this and not the examples.

Basic arguments

How data are organised in a file

The .rt file is a text file.
This is an example which describes the way data are subdivided:

```
{ section1
        { subsection1
                attribute1 value;
        }
        attribute2 value;
}
{ section2

}
.
```

This file cannot be read by RT, it is only a template to show the highlight of the structure of a .rt file. The real identifiers used in a real .rt file are described after, with their meanings.
The parts of file separated by { and } are called sections. The file ends with a dot (.) .
The word after { is the type of the section.
A section can contain other sections (called subsections) or attribute.
A section contains some attributes. Every attribute has a value. The value depends on the attribute, name. The attribute values can be also subsection.
The attribute that can be in a section depends on the type of the section.
The sections describe things in the scene. The attributes tell how things are made. If a section needs the data of another one, the first one must follow the second.

Don’ t worry if you don’ t understand exactly, you will understand when you see the examples.

A simple scene (Tut01.RT)

Look at file TUT01.RT.
Tut01.rt is the simplest scene you can render. All sections are needed.
This scene is a sphere with a light source (lamp) .
Here a short description of the sections used:

vcr
view coordinate reference, it describes the 3d view of the scene.

lamp
is a light source.

material
describes the physical characteristics of an object, if it is red or white, if it is opaque or transparent.

object
is an object of the scene.

They can be declared in any order, except in the case of materials which must precede objects, because, the object sections reference to the material section.
For lamps, materials, and objects vary types of model exist. For example an object can be a sphere (like in this case) or a box or other. All the objects have some attributes that are common to all objects, (the part before model) while others that are specific to the particular type of object. The common attributes in the section object that must always be declared are name and material.

Give a look to the section object. The common attribute can be:

name
is the name of the object. All objects in the scene must have an unique name; if not the new object will override the old.

material
is the name of the material which describes the physical characteristics of an object. It must exist a material section which declares a material which name will be used in a object section.
For some type of object, called metaobjects, you must not include the attribute material, but these are not real object. They are explained later.

model
its value is a subsection which name is the type of the object: in this case sphere. It must be the last attribute in the section.

The sphere is the most simple object. It has only two attributes: center and radius.

center
is the center of the sphere. It is described in the point format "(x , y , z)".

radius
is a real, the radius of the sphere.

For material section , it is the same. There is a common part, but you can use only the common attribute name. Now is available only a type of material: phong, so you must always use phong. It will be used by one or more object section (a material never used is not an error). The name must be unique, if not it will delete old definition.
We can divide its attribute into three groups: the group which start with letter K (coefficient) , which start with O (colors) , the others.
The K attributes are:

Ka
is the ambient-reflection coefficient.

Kd
is the diffusive-reflection coefficient.

Ks
is the specular-reflection coefficient.

Kt

the transmission coefficient.

They determinates the object color and the direction of the ray when it intersects an object with this characteristics. Ka allows to show object without a lamp, using a special ambient lamp, but you don’ t get good results with Ka attribute.

Some examples:

Ka=0, Kd=0.7, Ks=0, Kt=0	an opaque object
Ka=0, Kd=0.6, Ks=0.1, Kt=0	a reflexive object
Ka=0, Kd=0.1, Ks=0.7, Kt=0	a mirror
Ka=0, Kd=0.3, Ks=0, Kt=0.8	a transparent object

The O attributes are:

Od
is the diffuse color. If you want change the color of an object change this, it is what you intend for color of an object.

Os
is the specular color. If you want to change the color of the highlight, change Os.

They are specificated in color format "(r , g , b)" where r, g, b are the red, green, blue component, which must be real numbers between 0 and 1 and 1 means all color , 0 no color.
The other parameters are:

n
the specular-reflection exponent. Defines how large must be the reflex of light sources on object.

ni
the index of refraction. Determinates the deviation of the ray when it crosses a transparent object.

The same is true for lamp. The common attribute is only the name (must be unique).
A model is pointlamp, which can transmit the same energy at any distance. It is all in a point. pointlamp has attributes pos and color.

pos
the position of the lamp. The lamp is all in a point. It is in the point format "(x , y , z)".

color
color of the lamp. A rgb color in the format "(r , g , b)".

Remember that sometimes you can' t see the object because there are not light sources.

A complex thing is to put the view in a way that you can observe the object you want.
The view described is more complex, but you can make some images without understanding it. If you want to learn it go to paragraph "Understanding the 3D view". If you don' t want for now, you can use an empirical way, but for a simple use. You can copy every time the vcr section of the file tut01.rt which we will call the standard view (for program RT).
It is sufficient you follow a simple rule : your object must have all its points with x coordinates > 0 and the distance from x axis * 2 < x coordinate.
For example:

point (-3 , 0 , 2) is out the view
because x < 0 , for the first condition

point (7 , 3 , 10) is out the view
because sqrt(3*3+10*10) * 2 = 20.8 < 7, for the second condition

point (17 , 3 , 3) is ok

You must not calculate it for all the points of an object: you can compute only for the center or another point so you will see that part. Then , if you want, you can move it empirically to see it better.

Some hints

A .rt file is a little verbose. A good technique is to start form some already written files and to modify them, adding sections using copy and paste in a text editor.

A text file is slower to read than a binary file, but it is simple to edit and understand. It is more flexible.
Next versions or RT will improve the file format, adding new features or modifying olds, but old formats will be always compatible, so all you learn you can always use: the program will auto-translate old characteristics into new. Another bonus of text file format is that you can easily write a program which outputs a scene description in text file format, while it is complex to do in binary mode.

Adding a new sphere (Tut02.rt)

You must only add two section. One material, one object.
You can use copy and paste for file Tut01.rt. Copy the last two section and paste then in the end of file, before the dot (“.”). Change the name of material and the object, and change the sphere position. Change the name attribute of the last two sections; subst “name mat1;” with “name mat2;” and “name sph1;” with “name sph2;” so RT can distinguish them. Then you must move the last object, because it occupies the same space of object sp1 and you will see only one. Move it up, changing the coordinates of the center. You can see this transformation in file tut02.rt.

Changing the color of an object (Tut03.rt)

In previous example you could use the same material for the two sphere. You use two different material if you want change the characteristic of one. If you want to change the color change Od in the material section with name “m2”. Look at file tut03.rt.

Modifying the parameters (Tut04.rt)

Now you may want to change some image characteristics, like the image dimension: in fact a 100 x 100 image may be too small. To do this, you must add a new section.
Look at file tut04.rt . There is a new param section.
Param section allows to change the parameters which determinate the way to build the image. Below are described the principal attribute of param section:

dimx, dimy

represent the image dimension in pixel (default is 100x100).
If you change the image dimension, and horizontal dimension becames different from vertical, you can affect the aspect ratio, so circles became ellipses and squares rectangles. To avoid this you must change the view window in the view section (look at paragraph “Understanding the view”), or , simply, always use squared image of any dimension, with dimx = dimy.

shade

is the way the color will be computed. The allowed values are “phong” and “noshadowsphong”. The default is “phong”. “phong” and “noshadowphong” are similar, but “noshadowphong” does not compute the shadows. It is faster, but the image could be lighter.

mode

is the number of deviation allowed for a ray. If you set =0 you will not see transparency and reflection. Don’ t specificate big value. A value of 5 or 7 is good for transparence.

anti_alias

is a technique for creating a smoothed image. Set to 1 if you want, o if not. The program will use adaptive antialiasing.

backcolor

is the color of the background. its value is in rgb format.
buildhier if set to 1, create an automatic hierarchy to speedup the creation of the image. If set to 0 no hierarchy is used. This will be explained later.

Adding a box (Tut05.rt)

A box is parallel to the x, y, z axes. It is simple like a sphere.
You must only specificate the max and the min value for x, y and z, like in file tut05.rt.

The brep object

Brep stands for boundary representation. You define a solid describing its boundary surface.
You use polygon to make the surface. Every object with a boundary surface is contained in a .brp file.
The .brp file is a text editor, but this can be very complex and boring to write. You can use other program like 3D Studio for creating a model, save it like a .dxf file and convert it with program dxf2rt. Remember : dxf2rt convert only object formed by polyline not from 3dface.

Look at his declaration in a .rt file :

```
{ object
        name breptest;
        ...
        model {
                name mm1;
                constr 1.2;
                interp 0;
        }
}
```

This includes a brep object with name “breptest”. It looks for model in the file “mm1.brp”. First it look in the current directory, then, if RT haven’ t find it, the program will look in the common model directory.

A particular subset are the brep formed only from triangles. They have particular possibility not allowed for general brep (for now).

The attribute interp create a smooted brep object, but you need that all the faces of the object are triangles.

Cylinder

It is possible to create a cylinder.
The syntax is:

```
        { cylinder
                high 1;
                radius 0.5;
        }
```

The cylinder axes is the z axes. The default values are 1 for high , 0.5 for radius.

Cone Lamp

You may create a lamp which lights only in a conic area.

```
        { conelamp
                pos ( 0 , 0 , 0 );
                color (1,1,1);
                dir ( 0 , 0 , -1 );
                ang 0;
        }
```

pos and color are like for the point lamp.

dir
a vector which gives the illumination direction.

ang

an angle which tells how large is the conic area of the lamp.

Creating a coloured background (Tut06.rt)

Using a box you can create your own background. It is sufficient you make a big box which cover all the image window and is back to all the object. You can give it the color you want. Set its material with Kt = 0 and also Ks = 0 if you don' t want reflection on it. Look at tut06.rt.

Applying a texture mapping to an object (Tut07.rt)

You can apply a bitmap to an object. If you use a box as a background you can use a bitmap as background. But , if you want, you can apply a texture mapping to any object.
Before you need a bitmap. It must be a .bmp file, in Windows 3.0 bitmap file format and only at 24bit format (16 million of colors). If , for example, its name is "bitm.bmp" you must put it in the same directory where is the file or in the default texture directory. Put the bitmap in the same directory with your .rt file if it is the only file will use that bitmap, while put it in the common directory if more files use it.
You can preload in memory the bitmap with a new section:

```
{ texture name bitm; }
```

After in any object section you can use the attribute texture to apply the bitmap: the colors of bitmap will override the Od parameter.
You must write like this:

```
{ object
	name ...;
	texture bitm;

	....
}
```

You have not finished. You have told what texture to apply, not how to apply. In fact you must add a new common attribute "mapping".
A mapping associate every point in a 3d system (in this case in the object 3d system) with a point of the bitmap, and then with a color.
There are many types of mapping, one of this is the planar.

This is an example of the definition of a planar mapping:

```
mapping { mapping
   model { planar
 center ( 0 , 0 , 0 );
 assx ( 0 , 1 , 0 );
 assy ( 0 , 0 , -1 );
 moltx 0.130;
 molty 0.130;
}
```

mapping is a common attribute of an object section. The value is a subsection of mapping type. The mapping section follows the same rule of the already seen sections: a common part for all mapping types (null in this case), a model attribute, always present and always the last in the section. For now the allowed values are: planar, spherical and cylinder.
In this case is planar.

The planar model put the bitmapped texture on a plane in the space, fill all the plane with periodic replies of the same and then it projects the points in the plane in all the space; in this way planar mapping associates every point in the space with a color. Obvious, this will affect only the object in which the mapping is declared.
center, assx, assy defines the plane and the position of the bitmap. The plane contains the center point and it is parallel to vector assx and assy. Also they defines a reference system on the plane: center is the origin and assx and assy the axes. The primary bitmap will lies in the space defined by disequations : 0 <= x <= 1 and 0 <= y <= 1.
You can modify the dimension of the image with moltiplicative factor moltx and molty, which enlarge the image in

horizontal and vertical.

Advanced argument

Limits of the raytracer

The program works with any number of object and of light source. There are some constraints to follow:

you must use only closed objects. A closed object is for any line in the space, the intersection with the object consists or in no intersection point or in a even number of intersection points. It is possible to create objects non closed (es. with the brep). This object are wrong. The program does not control if an object is closed.

Compenetrations between objects are forbidden. Two objects cannot share the same point of the space. The program does not control this. If you want the same effect of a compenetration use a csg node of type union.

This limits are true if you want transparent objects. An object with material characteristic with Kt equal to 0, the object is not transparent and you can use open objects.

The program never checks if things are good, it thinks you know what are you doing.

A transformation of an object (Tut08.rt , Tut09.rt) or a lamp

transf describes an object rotated, translated, scalated.

```
  { object
     name transfname;
     model { transf

           s x x x;
           r x x x;
           s x x x;
           t x x x;
           ....

           {object
              ....
           }
     }
  }
```

the object transformation is described by a sequence of r(rotation),
s(scale), t(translation). The object in the transf. Section can be any type of object, also another transf. transf is a metaobject.

In Tut08.rt there is a box rotated. The box, centered in the origin, is rotated around z axes.
In Tut09.rt a sphere becomes an ellipsoid. The sphere, centred in the origin, is scaled in y axes.

If you want to apply a transformation to an object, you must check if the object is centred in the origin, because the rotation and the scale are made on this assumption. If not, the object will be rotate and scaled as you want, but also it will be moved in a unwanted position.
If you know the center of an object, you should first translate it to the origin of the axis, then you can rotate or scale it how you want. The object will be scaled and rotated, but the center will not move from the origin. Now you can remove the center in the original position.

You can now apply a transformation to a lamp too. The syntax is similar.

```
{ lamp
     name transfname;
     model { transf

           s x x x;
```

```
            r x x x;
            s x x x;
            t x x x;
            ....

            {lamp
                ....
            }
        }
    }
```

This is usefull for some special effects, like to scale a cone lamp.

Group of objects

group describes a group of objects. The syntax is:

```
    { object
        name groupname;
        model { group

            { object
                ...
            }

            { object
                ...
            }

            ...
        }
    }
```

The object in the group can be anyone, also other groups. group is a metaobject.
What can you do with this ?
1) you can group more objects in a logical way (for example, an computer is a complex object, a group of monitor, keyword, desktop and so on).
2) you can translate or rotate or scale a group of objects only with a transf section.

Building a csg node (Tut10.rt)

CSG stands for constructive solid geometry.
In tut10.rt you can see, from left to right the subtraction, the union and the intersection between two spheres.
You must give the two objects, like attributes :first and second. This attribute values can be also other csg node.
The material is not needed, csg node use the material of the associate objects.
Remember : objects in csg node must be always closed.
Look at file tut10.rt for an example of all tree case.

Bezier Mesh

I suppose you know what a bezier patch is. The bezier mesh is a series of path which form an object.
To add a bezier mesh, you must add a new subsection to attribute “model” in a section “object”.

```
            { bezier
                    name xxx;
                    cache x;
                    div x;
            }
```

Only name is needed.
“name” is the name of the model. The model is in a .bzr file in the model path (the same for brep model).

“cache” tells is rt must record the data needed to render the object. The value for “cache” must be 1 if you want to record, 0 if not. Default value is 1.
RT will convert bezier mesh in a brep object. “div” tells RT how accurate must be the convertion.

There is a new .bzr format. A bezier mesh is a series of bezier patch. Here there is a template :

```
{ bezier

  { patch
      (1.40000, 0.00000, 2.40000)
      (1.40000, -0.78400, 2.40000)
      (0.78400, -1.40000, 2.40000)
      (0.00000, -1.40000, 2.40000)

      (1.33750, 0.00000, 2.53125)
      (1.33750, -0.74900, 2.53125)
      (0.74900, -1.33750, 2.53125)
      (0.00000, -1.33750, 2.53125)

      (1.43750, 0.00000, 2.53125)
      (1.43750, -0.80500, 2.53125)
      (0.80500, -1.43750, 2.53125)
      (0.00000, -1.43750, 2.53125)

      (1.50000, 0.00000, 2.40000)
      (1.50000, -0.84000, 2.40000)
      (0.84000, -1.50000, 2.40000)
      (0.00000, -1.50000, 2.40000)
  }

  { patch
      (0.00000, -1.40000, 2.40000)

          ....

  }

          ......
}
.
```

Understanding the view

The complete format (not all are used by RT) is:

```
{ vrc
     vrp ( 0 , 0 , 0 );
     prp ( 0 , 0 , -10 );
     vup ( 0 , 0 , 1 );
     vpn ( 1 , 0 , 0 );
     umax 3;
     umin 0;
     vmax 3;
     vmin 0;
     type 1;
     f 1;
     b 0;
     fon 0;
     bon 0;
}
```

All the scene is described in an absolute system.
With param vcr, vup, vpn, you describe a second system, the observer system. So, if you want to modify the view, you must modify only some parameter in vcr section. The tree axes are called u,v,n.
So if you want to modify the view, you must modify only some parameter in vcr system.

vrc
is the origin of the new coordinate system.

vpn
defines the n axis.

vup
defines the v axis.

Fixed this three parameter, the new system is defined. The image will be on plane uv. We can define which part of plane we want to represent the image, with:

Umin, Umax, Vmin, Vmax
the min and max value on the u,v axes, which represent the image that we want to see.

The others parameters are :

prp
describes the position of the observer. IMPORTANT: its coordinates are in the vcr system and not in the absolute system like other parameters. The rays will start from prp point and will go through the window and in the scene.

type
is the way of representation: o means parallel projection, 1 means perspective projection.

Example:
If you want to center an object at position (10,20,5) you must follow this procedure.

1) fix parameter prp, Umin, Umax, Vmin, Vmax. They will not change if you move the system.
2) put vcr in a point quiet near the object. Es. (0,10,0).
3) the n axis is the direction where you see. We can set vpn equal to (10,10,5) so it is in the direction exactly (vcr + vpn = object position). Don't worry if vpn is not a versor.
4) You can use vup to set the top of the image. You can set it to (0,0,1). Don't worry if vup is not a versor or it is not vertical to vpn, but it is important that vup and vpn are not parallel.

If you want an aspect ratio = 1 you must follow the rule :

dimx / dimy = (Umax - Umin) / (Vmax - Vmin)

where dimx and dimy are attributes of param section, while Umax, Umin, Vmax, Vmin are attributes of vcr section.

Other attribute on param section

Other attributes that you may encounter in the param section are:

render
is the type of output. The allowed values are : "raytracing" (standard), "depth", "vrml", "t_analysis".
"raytracing" is the standard output.
"vrml" outputs a .wrl file. It is in VRML 1.0 file format. You can see it with a VRML browser.
"t_analysis" outputs a .ana file. It describes some characteristics of the objects in the scene.

toll
is a value used in intersection calculus. Intersection is true if the distance is less then toll. This avoid the noise. If in your image you get some noise you may change this.

anti_al_toll

if it is low RT makes a more in depth analysis for antialiasing.

ambientcolor
is the ambient lamp color. Default is (1,1,1). You don’ t need to change this.

backcolor
color of the background. The default is (0,0,0).

ambient_ni
value for refraction coefficient of the vacuum. Default is 1.

Adding comment

You can add your comment to .rt files.
They use the C++ style:
1) you can put you comments from "/*" to "*/"
2) you can start comments from "//" until the end of line
For example:

```
/*   RT example

     more line, this is a comment

*/

{object
     ...
     model { sphere // comment until the end of line

          ...
     }
}
.
```

Remember that if you use with some automatic programs which work on .rt file you will lose these comments!

Creating stereogram

There are two phases to create stereogram:
1) create a depth bitmap with program rtw
2) create the stereo bitmap from depth bitmap with program rtst or other program.

1) from file .rt you want to create a stereogram, modify or add the attribute render equal to “depth”. Add if you want the attribute depthmax, depthmin, depthback.
depthmax is the max value allowed in the depth image, depthmin is the min. depthback is the value for dot without objects (the background).
The default values are: depthmax 64, depthmin 48, depthback 64. Remember that the output is a grey scale image, but in the 24bit format. For some use you must convert it.

2) if you use rtst you must only execute from file manager

```
rtst  <depth image> <output bitmap>
```

Using hierarchy

If in the param section of a .rt file exist attribute buildhier with value 1, then the program will use an hierarchy for speedup the creation of the image.
If it is the first time that it loads the .rt file (for example demo.rt) then the program creates an hierarchy and save it in a file with extension .rt_ and the same name (it will be demo.rt_ in the example). The next time you load the file to create the image, it will find the saved file and load the hierarchy without creation.
If you have changed some object, or added or deleted, you may encounter an error or a wrong image; in this case you should delete the .rt_ file.
You should not delete the file if you change a section different from object section.

Appendix

If something goes wrong

If you get an error or an unknown exception, perhaps you have find a bug in the program. You could contact the authors to inform about it. We like to know the file which make the problem and your computer configuration. We will correct the bug or will explain you which was the cause.
If you get an error when you load a scene, perhaps there is an error in the .rt file. Error code 1 means generic error, the program cannot explain you what type of error is. But you obtain the position in which the error was encountered, so you can read the file and modify it.
Sometimes the position could be a little incorrect, a little moved forward,
but at least a line.

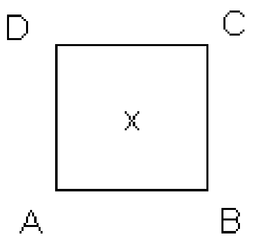
About .brp file format

The description of a simple file .brp:

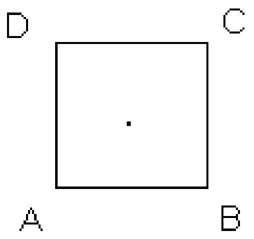
```
{ pbrep
        vertices
        3
        0    0    0
        1    0    0
        1    1    0
        0    1    0

        faces
        1
        { 1 { 4 0 1 2 3 } }
}
```

This is a simple object. It is a square in the plane x-y with edge of 1.
Remember this is an open brep, you cannot assume it will be a transparent object.
There are definite four vertices, which we call A for (0,0,0) , B for (1,0,0) , C for (1,1,0) , D for (0,1,0) . It is definite one face with one contour. A face can have more than a contour; in this case the first contour is the external contour, while the others are holes in the face.
Every face has a normal versor, which describes the external side of a face. We can tell this to the program with the order we put the vertices in the contour description:

x exit
. enter



ABCD   anticlockwise



ADCB clockwise

If you have a hole, the order of the hole contour must be opposite to the external.

File format

The file format is defined in a formal way for sintax, in a informal for the semantic.

Definition of the metalanguage:

Definition

A = B   means A is defined like B

Sequence

A + B   means A is tied with B

Selection

A | B   means A or B

Repetition

{A } means that it is possible repeat A a inprecisate number or time (also zero).

Optional

[A]   means A or nothing

Non rapprentable simble

description'   in this way you can indicate a simbol the you can't print

Also:
An identificator in bold means a terminal simbol, a normal identificator means a non terminal simbol.

Numeric definition

plus = the simbol +'
dot = the simbol .'
digit = [**0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**]
number_integer = [**-** | plus |] + digit + { digit }
number_float = number_integer + [**.** + { digit }]
number_real = number_float + [**E** + number_integer]

number_integer
    is an integer number (es. 18554).

number_float
    is a number with comma (es. 34.87653).

number_real
    is a number in scientific format (es. 13.47E-2).

definition of identificator

alfama = [**A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** | **L** | **M** | **N** | **O** | **P** | **Q** | **R** | **S** | **T** | **U** | **V** | **X** | **Y** | **Z**]
alfami = [**a** | **b** | **c** | **d** | **e** | **f** | **g** | **h** | **i** | **j** | **k** | **l** | **m** | **n** | **o** | **p** | **q** | **r** | **s** | **t** | **u** | **v** | **x** | **y** | **z**]
alfa = [alfama | alfami]

ident_first = [alfa | _]
ident_other = [alfa | digit | _]
ident = ident_first + [{ ident_other }]

ident_first
    is the first character of an identifer.

ident_other
    the type of all characters of an identifier but the first.

ident
    declaretion of an identifier.

definition of a separator
(between identifiers or numbers or other)

ret_carriage = 'carriage return (ASCII = 13) '
space = ' '
tab = ' tabulation character (ASCII = 6) '
sep = { [space | ret_carriage | tab] }

The separator is used to separe two identifiers.
There is no difference between space, carriage return and tabulation.

Now we will use a new meta-operator dot (.), with the meaning of " + sep +".
For example: **a** . **b** is the same of "a b" or of "a b"

Other common definition

vector = (. number_real . { , . number_real } .)
matrix = [. number_real . { [, | ;] . number_real } .]
point = (. number_real . , . number_real . , . number_real .)
color = point

matrix
    this definition is meaningfull if the groups which are separated from ; have the same number of elements separated from , .
    For example [2 , 3 ; 4 1.2] is correct, [3, 1; 7] is not correct.

Section

For the text type definition the common way to represent the structured data is to create a section which contains the data with similar definition.
The definition prototype is :

section = **{** . sectionname . defsection . **}**

sectionname = ident

section
    definition of whole section.

sectionname
    definition of the type of the section.

defsection
    declaration of the data in the section. It is impossible to define this non terminal; it must be defined every time.

Definition of the file format

File .bmp

It is the standard file format for bitmap defined for Windows 3.0.
You can use only bitmap with 24bit.

File .brp

Definition of a geometric model of type Boundary Representation.

A face is defined by some contours. The first contour is the extern contour, which must be always included; the other contours describe the holes in the face.
A contour of a face is defined by the values of vertex indexes which limit it.
You must verificate the condition that all vertices of all contours in a face are complanar.

filebrp = **{** . **pbrep** . section_vertex . section_face . **}** . dot
section_vertex = **{** . **vertices** . defvertex . **}**
section_face = **{** . **faces** . defface . **}**
defvertex = number_vertex . { vertex }
number_vertex = number_integer
vertex = number_real . number_real . number_real
defface = number_face . { defcontour }
defcontour = **{** . number_contour . **{** . contour . **}** . **}**
contour = number_index . { index }
number_index = number_integer
index = number_integer

filebrp
: definition of file .brp.

section_vertex
: definition of all vertex. All the vertex are definited by a sequence of vertex definition. It is possible to associate an index to every vertex, 0 to the first, 1 to the next and so on.

defvertex
: definition of the verteces.

number_vertex
: definition of the number of the verteces in a brep model.

vertex
: definition of a single vertex. A vertex is a point in the space.

sections_face
: definition of all faces.

defface
: definition of the faces.

defcontour
: definition of the contours in a face.

contour
: definition of a contour.

number_index
: number of verteces in a contour.

index

index of a vertex of a contour.

Example of file .brp:

```
{ pbrep

vertices
10

16 0 54
16 10 54
8 16 54
0 10 54
0 0 54
16 0 30
16 10 30
8 16 30
0 10 30
0 0 30

faces
7

{ 1
        { 5   0 1 2 3 4 }
}
{ 1
        { 4   1 2 7 6 }
}
{ 1
        { 4   2 3 8 7 }
}
{ 1
        { 4   3 4 9 8 }
}
{ 1
        { 4   4 0 5 9 }
}
{ 1
        { 4   1 0 5 6 }
}
{ 1
        { 5   5 6 7 8 9 }
}

}
```

File .bzr

Definition of a geometric model of type Bezier mesh.

File .rt

The file describes a raytracer scene.
It is so defined :

filebrp = section_rt . dot
section_rt = { [section_view | section_param | section_texture | section_material |
section_lamp | section_object] }

filebrp
definition of the entire file .rt .

Section View

This section defines the 3D view of a raytracer scene with the sistem VRC (3D viewing-reference coordinate).
To define exactly all the attributes you can see in [1].

section_view = **{** . **vcr** . { [def_vrp | def_prp | def_vup | def_vpn | def_umax | def_umin | def_vmax |
def_vmin | def_type | def_f | def_b | def_fon | def_bon] } . **}**
def_vrp = **vrp** . point . ;
def_prp = **prp** . point . ;
def_vup = **vup** . point . ;
def_vpn = **vpn** . point . ;
def_umax = **umax** . number_real . ;
def_umin = **umin** . number_real . ;
def_vmax = **vmax** . number_real . ;
def_vmin = **vmin** . number_real . ;
def_type = **type** . number_integer . ;
def_f = **f** . number_real . ;
def_b = **b** . number_real . ;
def_fon = **fon** . number_integer . ;
def_bon = **bon** . number_integer . ;

section_view
complete definition of the view 3D.

def_vrp
definition of the view reference point (VRP) , the origin of VCR system.
With the VPN defines the plane of the image.

def_prp
definition of the projection reference point (PRP).

def_vup
definition of the view up vector (VUP), which limit the v axis on the image plane.

def_vpn
definition of the view-plane normal (VPN), is the n axis of the VRC system.
With the VPN defines the plane of the image.

def_umax
def_umin
def_vmax
def_vmin
definition of limitis of the image window on the view plane.

def_type
Type of projection.
1 means perspective projection , while 0 means parallel projection.

def_f
the front distance (F). This parameter will be ignored.

def_b
    the la back distance (B). This parameter will be ignored.

def_fon
    This parameter will be ignored.

def_bon
    This parameter will be ignored.

Section parameters

This section defines the parameters which change the execution of the raytracer.

section_param = **{** . **param** . { [def_toll | def_mode | def_shade | def_anti_alias |
    def_anti_al_toll | def_dimx | def_dimy | def_ambientcolor | def_backcolor |
    def_buildhier | def_modifydef | def_depthmax | def_depthmin |
    def_depthback] } . **}**
def_toll = **toll** . number_real . ;
def_mode = **mode** . number_integer . ;
def_shade = **shade** . ident . ;
def_anti_alias = **anti_alias** . number_integer . ;
def_anti_al_toll = **anti_al_toll** . number_real . ;
def_dimx = **dimx** . number_integer . ;
def_dimy = **dimy** . number_integer . ;
def_ambientcolor = **ambientcolor** . color . ;
def_backcolor = **backcolor** . color . ;
def_buildhier = **buildhier** . number_integer . ;
def_modifydef = **modifydef** . number_integer . ;
def_depthmax = **depthmax** . number_integer . ;
def_depthmin = **depthmin** . number_integer . ;
def_depthback = **depthback** . number_integer . ;

def_toll
    definition of the standard tollerance value.

def_mode
    max value of ray deviations. 0 == raycasting.

def_shade
    type of shading, an identifier recognize from the system. The standard value is “phong”.
    “phong” uses the phong shading metod.

def_anti_alias
    1 if you want to use the adaptive antialiasing, 0 if you don’t want it.

def_anti_al_toll
    value used in antialiasing.

def_dimx
    horizontal dimension (pixel) of the image.

def_dimy
    vertical dimension (in pixel) of the image.

def_ambientcolor
    this is the value of ambiental illumination.

def_backcolor
    back color, the color used for a pixel in the image when the ray doesn’ t match an object.

def_buildhier
    1 if you want to build a hierarchy to speed up the intersection calculus, 0 if you don’ t want it.

def_modifydef
    1 if you want that the program changes the file definition, 0 if you don’t want.

def_depthmax
    value used to build the depth graph.
    It is the maximum value of depth allowed.

def_depthmin
    value used to build the depth graph.
    It is the minimum value of depth allowed.

def_depthback
    value used to build the depth graph.
    It is the depth value for pixel of the back.

Section texture

This section is used to declare a texture

section_texture = **{** . **texture** . name_texture . ; . **}**
name_texture = ident

name_texture
    the name of file bitmap which contains the texture. The texture file has name name_texture + “.bmp” ed it will be in the standard directory with all other textures.

Section material

This section defines a material of a specific type and characteristic. In the section object you can use the material defined in this section.

section_material = **{** . **material** . **name** . name_material . ; . **model** . **{** . defmaterial . **}** . **}**
name_material = ident
defmaterial = [defmat_phong]
defmat_phong = **phong** . { [def_Ks | def_Kt | def_ni | def_Ka | def_Kd | def_n | def_Od | def_Os] }
def_Ks = **Ks** . number_real . ;
def_Kt = **Kt** . number_real . ;
def_ni = **ni** . number_real . ;
def_Ka = **Ka** . number_real . ;
def_Kd = **Kd** . number_real . ;
def_n = **n** . number_real . ;
def_Od = **Od** . color . ;
def_Os = **Os** . color . ;

name_material
    name of the material. It must be unique to distinguish a definition from another.

defmat_phong
    definition of material of type phong. See [1].

def_Ks
    Specular-reflection coefficient.

def_Kt
Refraction coefficient.

def_ni
index of refraction.

def_Ka
Ambient-reflection coefficient.

def_Kd
Diffuse-reflection coefficient.

def_n
Specular-reflection exponent.

def_Od
Diffuse color.

def_Os
Specular color.

Section lamp

This section defines the lamps (or light sources) in the scene.

section_lamp = **{** . **lamp** . **name** . name_lamp . **model** . **{** . deflamp . **}** . **}**
name_lamp = ident
deflamp = [deflamp_point | deflamp_cone]
deflamp_point = **pointlamp** . { [defpoint_pos | defpoint_color] }
defpoint_pos = **pos** . point . ;
defpoint_color = **color** . color . ;
deflamp_cone = **conelamp** . { [defcone_pos | defcone_ang | defcone_dir | defcone_color] }
defcone_pos = **pos** . point . ;
defcone_ang = **ang** . number_real . ;
defcone_dir = **dir** . number_real . ;
defcone_color = **color** . color . ;

name_lamp
name of the lamp. It must be unique.

deflamp_point
definition of a uniform type lamp.

defpoint_pos
definition of the lamp position.

defpoint_color
definition of the lamp color.

deflamp_cone
definition of cone lamp.

defcone_pos
vertex of cone lamp.

defcone_ang
opening angle.

defcone_dir
    direction of cone lamp.

defcone_color
    definition of the color lamp.

Section object

This section defines the objects in the scene.

section_object = **{** . **object** . **name** . name_object . attrib_std . **model** . **{** . defobject . **}** . **}**
name_object = ident
attrib_std = { [attrobj_material | attrobj_toll | attrobj_texture | attrobj_mapping] }
attrobj_toll = **toll** . number_real . ;
attrobj_texture = **texture** . ident . ;
attrobj_mapping = **mapping** . { **mapping** . **model** . { name_mapping . defmapping } . }
name_mapping = ident
defmapping = { [attrmap_center | attrmap_assx | attrmap_assy | attrmap_moltx | attrmap_molty] }
attrmap_center = **center** . point . ;
attrmap_assx = **assx** . vector . ;
attrmap_assy = **assy** . vector . ;
attrmap_moltx = **moltx** . number_real . ;
attrmap_molty = **molty** . number_real . ;
defobject = [defobj_sphere | defobj_box | defobj_brep | defobj_quad | defobj_csg]

defobj_brep = **brep** . { [defbrep_name | defbrep_costr] }
defbrep_name = **name** . ident . ;
defbrep_costr = **costr** . number_real . ;

defobj_sphere = **sphere** . { [defsphere_center | defsphere_radius] }
defsphere_center = **center** . point . ;
defsphere_radius = **radius** . number_real . ;

defobj_box = **box** . { [defbox_xmin | defbox_xmax | defbox_ymin | defbox_ymax |
    defbox_zmin | defbox_zmax] }
defbox_xmin = **xmin** . number_real . ;
defbox_xmax = **xmax** . number_real . ;
defbox_ymin = **ymin** . number_real . ;
defbox_ymax = **ymax** . number_real . ;
defbox_zmin = **zmin** . number_real . ;
defbox_zmax = **zmax** . number_real . ;

defobj_quad = **quad** . { [defquad_a | defquad_b | defquad_c | defquad_d | defquad_e
    | defquad_f | defquad_g | defquad_h | defquad_i | defquad_j] }
defquad_a = **a** . number_real . ;
defquad_b = **b** . number_real . ;
defquad_c = **c** . number_real . ;
defquad_d = **d** . number_real . ;
defquad_e = **e** . number_real . ;
defquad_f = **f** . number_real . ;
defquad_g = **g** . number_real . ;
defquad_h = **h** . number_real . ;
defquad_i = **i** . number_real . ;
defquad_j = **j** . number_real . ;

defobj_csg = [**intersect** | **union** | **subtract**] . { [defcsg_first | defcsg_second] }
defcsg_first = **first** section_object [**;** |]
defcsg_second = **second** section_object [**;** |]

section_object
definition of an object.

name_object
name of the object. It must be unique.

attrobj_toll
value of tollerance for the intersection with the object.

attrobj_texture
name of the texture. The texture must be preload in a previous section texture. The object can ignore this value.

attrobj_mapping
mapping of the object. The object can ignore this value.

name_mapping
type of the mapping. The standard values are planar, cilindric, spherical.

defmapping
mapping caracteristic, it define a reference system in the space.

attrmap_center
center of the reference system for mapping.

attrmap_assx
versor parallel to the x axis of the reference system of the mapping.

attrmap_assy
versor parallel to the y axis of the reference system of the mapping.

attrmap_moltx
scale factor for x axis of the mapping reference system.

attrmap_molty
scale factor for y axis of the mapping reference system.

defobject
definition of the object model.

defobj_brep
definition of a brep model.

defbrep_name
name of the file .brp which describes the geometric model of the object. The file .brp will have name defbrep_name + ".brp" and will be in the directory where are all the models.

defbrep_costr
value for the hierarchy of the object. If the value is equal to zero, no hierarchy will be used. Se the value is different from zero and a previus hierarchy of the object already exist, this will be used (also if it has been created with a different value). If the hierarchy doesn' t exist, the program will create a new with compactness value equal to the value.

defobj_sphere
definition of a sphere.

defsphere_center
definition of the sphere center.

defsphere_radius
    definition of the sphere ray.

defobj_box
    definition of a box.

defbox_xmin
defbox_xmax
defbox_ymin
defbox_ymax
defbox_zmin
defbox_zmax
    extreme values of a box oriented to the axes .

defobj_quad
    definition of a quadric.

defquad_a
defquad_b
defquad_c
defquad_d
defquad_e
defquad_f
defquad_g
defquad_h
defquad_i
defquad_j
    definition of the form of the quadric. The equation is
    A*x^2+B*y^2+C*z^2+D*x*y+E*x*z+F*y*z+G*x+H*y+I*z+J=0

defobj_csg
    definition of a csg node.

defcsg_first
    first object which is in the csg node.

defcsg_second
    second object which is in the csg node.

Example of a file .rt :

```
{ vrc
      vrp ( 63.7424 , -77.0513 , 0 );
      prp ( 0 , 0 , -100 );
      vup ( 0 , 0 , 1 );
      vpn ( -0.637424 , 0.770513 , 0 );
      umax 6;
      umin -6;
      vmax 4.5;
      vmin -4.5;
      type 1;
      f 1;
      b 0;
      fon 0;
      bon 0;
}

{ param
```

```
        toll 0.01;
        mode 5;
        shade phong;
        anti_alias 1;
        anti_al_toll 0.05;
        dimx 1024;
        dimy 768;
        ambientcolor (1,1,1);
        backcolor (0,0,0);
        ambient_ni 1;
        buildhier 1;
        modifydef 1;
        depthmax 64;
        depthmin 48;
        depthback 64;
}

{ texture
        name wood ;
}

{ material
        name mvetro;
        model { phong
                Ks 0.6;
                Kt 0.8;
                ni 1.2;
                Ka 0.0;
                Kd 0.4;
                n 10;
                Od (0.24,0.19,0.88);
                Os (0.24,0.19,0.88);
        }
}

{lamp
        name lamp1;
        model { pointlamp
                pos ( 5 , 0 , 3 );
                color (0.7,0.7,0.7);
        }
}

{object
name watch1;
material mvetro;
toll 0.01;
model { brep
                name watch1;
                costr 1.1;
        }
}

{object
        name sph3;
        material m3;
        texture marple;
        mapping { mapping
                model { spherical
```

```
                    center ( 10 , 2 , 0 );
                    assx ( 1 , 0 , 0 );
                    assy ( 0 , 1 , 0 );
                    moltx 0.1;
                    molty 0.1;
                }
        }
        toll 0.01;
        model   { sphere
                center ( 10 , 2 , 0 );
                radius 0.95;
        }
}

{ object
        name tcsg;
        material m1;
        model { intersect
                first {object
                        name sph1;
                        material m1;
                        model   { sphere
                                center ( 5 , 0.5 , 0 );
                                radius 0.95;
                        }
                };
                second {object
                        name sph1;
                        material m1;
                        model   { sphere
                                center ( 5 , -0.5 , 0 );
                                radius 0.95;
                        }
                };

        }
}

{object
        name box1;
        material m1;
        model { box
                xmin 10;
                xmax 12;
                ymin -1;
                ymax 1;
                zmin -1;
                zmax 1;
        }
}

{object
        name sph1;
        material m1;
        model   { quad
                a 10;
                b -1;
                c 2;
                d 0;
```

```
            e 0;
            f 0;
            g 0;
            h 0;
            i 0;
            j -1;
        }
}
.
```

Bibliography

[1] Foley, Van Dam, Feiner, Hughes (1992). Computer Graphics. Addison Wesley.

[2] Microsoft Corporation (1992-1993). Microsoft Windows 16bit/32bit Api Reference.