

Raytrace Workbench Help Index

[Introduction](#)

[Using the Mouse](#)

[The Toolbar](#)

[Menu Commands](#)

[POV-Ray Quick Reference](#)

[RTAG Quick Reference](#)

[Contacting the author](#)

Introduction

Raytrace Workbench a copyrighted-but-free program that is designed to facilitate the creation and/or modification of POV-Ray and RTAG script files. Some of the hilights includes :

- * Multiple document interface (work with more than one file at a time)
- * Extensive context-sensitive on-line help
- * Quick reference to POV-Ray objects, textures and more

Using the Mouse

Selecting text

word	double-click on the word
anything else	mark the starting point and drag

Context-sensitive help

Raytrace Workbench allows you to instantly look up a POV-Ray or RTAG command by placing the cursor on it and pressing the **RIGHT** mouse button.

Menu Commands

File

<u>N</u> <u>e</u> <u>w</u>	Create a new file.
<u>O</u> <u>p</u> <u>e</u> <u>n</u> <u>...</u>	Open an existing file.
<u>S</u> <u>a</u> <u>v</u> <u>e</u>	Save the current file.
<u>S</u> <u>a</u> <u>v</u> <u>e</u> <u>a</u> <u>s</u> <u>...</u>	Save the current file under a new name.
<u>E</u> <u>x</u> <u>i</u> <u>t</u>	Exit Raytrace Workbench.

Edit

<u>U</u> <u>n</u> <u>d</u> <u>o</u>	Undo the latest edit
<u>C</u> <u>u</u> <u>t</u>	Cut the marked text to the clipboard.
<u>C</u> <u>o</u> <u>p</u> <u>y</u>	Copy the marked text to the clipboard.
<u>P</u> <u>a</u> <u>s</u> <u>t</u> <u>e</u>	Paste the clipboard contents into the current file.
<u>F</u> <u>i</u> <u>n</u> <u>d</u> <u>...</u>	Find a text string.
<u>R</u> <u>e</u> <u>p</u> <u>l</u> <u>a</u> <u>c</u> <u>e</u> <u>...</u>	Find and replace a text string.
<u>F</u> <u>i</u> <u>n</u> <u>d</u> <u>N</u> <u>e</u> <u>x</u> <u>t</u> <u>...</u>	Repeat the last Find or Replace.

Render

<u>R</u> <u>u</u> <u>n</u> <u>P</u> <u>O</u> <u>V</u> <u>-</u> <u>R</u> <u>a</u> <u>y</u>	Save the current file and start POV-Ray.
---	--

Options

<u>P</u> <u>O</u> <u>V</u> <u>-</u> <u>R</u> <u>a</u> <u>y</u>	Set POV-Ray options.
<u>R</u> <u>T</u> <u>A</u> <u>G</u>	Set RTAG options
<u>D</u> <u>i</u> <u>r</u> <u>e</u> <u>c</u> <u>t</u> <u>o</u> <u>r</u> <u>i</u> <u>e</u> <u>s</u>	Set POV-Ray directories.
<u>F</u> <u>o</u> <u>n</u> <u>t</u>	Set the font used in edit windows.

Tools

<u>R</u> <u>T</u> <u>A</u> <u>G</u>	Run RTAG.
<u>D</u> <u>i</u> <u>s</u> <u>p</u> <u>l</u> <u>a</u> <u>y</u> <u>T</u> <u>a</u> <u>r</u> <u>g</u> <u>a</u>	Displays the associated Targa file.

Window

<u>T</u> <u>i</u> <u>l</u> <u>e</u>	Tile the windows.
<u>C</u> <u>a</u> <u>s</u> <u>c</u> <u>a</u> <u>d</u> <u>e</u>	Cascade the windows.
<u>A</u> <u>r</u> <u>r</u> <u>a</u> <u>n</u> <u>g</u> <u>e</u> <u>I</u> <u>c</u> <u>o</u> <u>n</u> <u>s</u>	Arrange the icons.
<u>C</u> <u>l</u> <u>o</u> <u>s</u> <u>e</u> <u>A</u> <u>l</u> <u>l</u>	Close all windows.

Help

<u>I</u> <u>n</u> <u>d</u> <u>e</u> <u>x</u>	Show Raytrace Workbench help.
<u>U</u> <u>s</u> <u>i</u> <u>n</u> <u>g</u> <u>H</u> <u>e</u> <u>l</u> <u>p</u>	Show Help on Help.
<u>A</u> <u>b</u> <u>o</u> <u>u</u> <u>t</u> <u>...</u>	Info about Raytrace Workbench.

File | New

Use this command to create a new file.

How to create a new file

1. Select **File** from the menu bar.
2. Select **New**.
3. A new window will be created.

Related topics :

[Opening an existing file](#)

[Menu commands](#)

File | Open

Use this command to open an existing file.

How to open an existing file

1. Select **File** from the menu bar.
2. Select **Open**.
3. A dialog box will pop up. Use the directory list and the file list to select the file you want to open.
4. Press **OK**.

Related topics :

[Creating a new file](#)

[Menu commands](#)

[The Toolbar](#)

File | Save

Use this command to save the file you are currently working on.

How to save a file

1. Select **File** from the menu bar.
2. Select **Save**.

Related topics :

[Saving a file under a new name](#)

[Menu commands](#)

[The Toolbar](#)

File | Save As

Use this command to save the file you are currently working on under a new name.

How to save a file under a new name

1. Select **File** from the menu bar.
2. Select **Save As**.
3. A dialog box will pop up. Select the new directory.
4. Type in the new name.
5. Press **OK**.

Related topics :

[Saving a file](#)

[Menu commands](#)

[The Toolbar](#)

Edit | Undo

Use this command to undo the latest edit.

How to undo the latest edit

1. Select **Edit** from the main menu.
2. Select **Undo**.

Alternatively,

1. Press Shift+Backspace

Related topics :

[Menu commands](#)

[The Toolbar](#)

Edit | Cut

Use this command to cut out selected text to the clipboard.

How to cut out text to the clipboard

1. Mark the text you want to cut out.
2. Select **Edit** from the main menu.
3. Select **Cut**.

Alternatively,

1. Mark the text you want to cut out.
2. Press Shift-Delete

Related topics :

[Copying text to the clipboard](#)

[Pasting text from the clipboard](#)

[Menu commands](#)

[The Toolbar](#)

Edit | Copy

Use this command to copy the selected text to the clipboard.

How to copy text to the clipboard

1. Mark the text you want to copy.
2. Select **Edit** from the main menu.
3. Select **Copy**.

Alternatively,

1. Mark the text you want to cut out.
2. Press **Ctrl-Insert**

NOTE: The text will be moved to the clipboard and will be erased from the current document.

Related topics :

[Cutting text to the clipboard](#)

[Pasting text from the clipboard](#)

[Menu commands](#)

[The Toolbar](#)

Edit | Paste

Use this command to paste text from the clipboard into the active document.

How to paste text from the clipboard

1. Select **Edit** from the main menu.
2. Select **Paste**.

Alternatively,

1. Press **Shift-Insert**

Related topics :

[Cutting text to the clipboard](#)

[Copying text to the clipboard](#)

[Menu commands](#)

[The Toolbar](#)

Edit | Find...

Use this command to search for a specific word, phrase or sentence in the file.

How to find a word, phrase or sentence

1. Select **Edit** from the main menu.
2. Select **Find**.
3. Type in the text you want to find.
4. Press **OK**.

Eventually, press **F3** to find the next occurrence of the text.

Related topics :

[Replacing text](#)

[Menu commands](#)

[The Toolbar](#)

Edit | Replace

Use this command to find a text and replace it with another text.

How to use Replace

1. Select **Edit** from the main menu.
2. Select **Replace**.
3. Type in the text you want to replace.
4. Type in the text you want to replace with.
5. Press **OK**.

Eventually, press **F3** to find the next occurrence of the text.

Related topics :

[Finding text](#)

[Menu commands](#)

[The Toolbar](#)

Render | Run POV-Ray

Use this command to start POV-Ray and begin rendering your scene. **Raytrace Workbench will save your file before starting POV-Ray.**

How to render a scene

1. First, check the POV-Ray options.
2. Select **Render** from the main menu.
3. Select **Run POV-Ray**.

NOTE: Depending on the selected output file format, the output file will be Filename[Ext];

If you have specified an output filename the output will be directed to that file. If not, it will be the name of the .POV file.

Where [Ext] is....	If you selected....
.TGA	Targa file Format
.DIS	Dump file format
.R8,.G8 and .B8	Raw file format.

The output file will be found in the directory specified in the [Directory options dialog](#).

See [POV-Ray options dialog](#) for more information about file formats.

If you want, you can press <Alt-Tab> to get back to Raytrace Workbench. When you are back, you can open and render another scene while the first is being rendered.

Troubleshooting :

If POV-Ray doesn't start, check the following :

1. Are the entries in **POV.PIF** correct ?
2. Did you follow the steps in **SETUP.TXT** ?

If the problem remains, contact the author.

Related topics :

[Setting POV-Ray options](#)

[Menu commands](#)

[Contacting the author](#)

[The Toolbar](#)

Options | POV-Ray

Use this command to set different POV-Ray options.

How to set POV-Ray options

1. Select **Options** from the main menu.
2. Select **POV-Ray**.

A dialog box will pop up.

Related topics :

[POV-Ray options dialog](#)

[Menu commands](#)

Options | RTAG

This command bring up a dialog box where you can specify RTAG options.

1. Select **Options** from the main menu.
2. Select **RTAG**.

A dialog box will pop up.

Related topics :

[RTAG options dialog](#)

[Menu commands](#)

Options | Directories

Use this command to specify which directories POV-Ray should search when looking for a file and where the finished image should be placed.

How to set POV-Ray directories

1. Select **Options** from the main menu.
2. Select **Directories**.

A dialog box will pop up.

Related topics :

[Directory options dialog](#)

[Menu commands](#)

Options | Font

Here you set the font Raytrace Workbench should use when displaying your POV-Ray scripts.

How to change the font

1. Select **Options** from the main menu.
2. Select **Font...**

A dialog box will pop up

3. Select the font you want.
4. Press **OK**.

Related topics :

[Menu commands](#)

Show color palette

Use this command to view the **color palette**. This palette is supposed to act as a tool whenever you need to select a color.

How to activate the color palette

1. Press .

The palette will disappear when you press **Insert** or when you switch to another window.

Related topics :

[The color palette](#)

[The Toolbar](#)

Tools | RTAG

This command executes RTAG with the current file. To set RTAG options, please see the [Options | RTAG](#) menu command.

Troubleshooting :

If RTAG doesn't start, check the following :

1. Are the entries in **RTAG.PIF** correct ?
2. Did you follow the steps in **SETUP.TXT** ?

If the problem remains, contact the author.

Related topics :

[Setting RTAG options](#)

[Menu commands](#)

[Conacting the author](#)

Tools | Display Targa















Use this command to launch your favorite targa viewer. Use [Options](#) | [Directories](#) to select viewer.

Related topics :

[Menu commands](#)

The Toolbar

Use the toolbar for fast and convenient access to RWB features.

Tool	Menu Command
	Help Index
	File <u>O</u> pen
	File <u>S</u> ave
	Edit <u>U</u> ndo
	Edit <u>C</u> ut
	Edit <u>C</u> opy
	Edit <u>P</u> aste
	Edit <u>F</u> ind
	Edit <u>F</u> ind
	Edit Find next
	Render <u>R</u> un POV-Ray
	<u>S</u> how color palette
	<u>I</u> nsert camera definition
	<u>I</u> nsert light source

POV-RAY Quick Reference

NOTE: Most (99%) of the information in this Quick reference was taken from QUICKREF.DOC, written by Drew Wells and Larry Tweed. The reference for **Textures.Inc** was written by Dan Farmer. This is **not part of the official release from the POV-Team** and may not be up-to-date.

This is only intended as a reference for users who are familiar with POV-Ray. For more in-depth coverage of a topic, please see the POV-Ray manual.

POV-Ray Scene description language

Shapes and Objects

Texture Modifiers

Textures.Inc

Colors.Inc

Stones.Inc

POV-Ray Scene description language

#declare
#include

#declare

The parameters used to describe the scene elements can be tedious to use at times. Some parameters are often repeated and it seems wasteful to have to type them over and over again. To make this task easier, the program allows users to create synonyms for a pre-defined set of parameters and use them anywhere the parameters would normally be used. For example, the color white is defined in the POV-Ray language as:

```
color red 1 green 1 blue 1
```

This can be pre-defined in the scene as:

```
#declare White = color red 1 green 1 blue 1
```

and then substituted for the full description in the scene file, for example:

```
object {  
  sphere { <0 0 0> 1 }  
  texture { color red 1 green 1 blue 1 }  
}
```

becomes:

```
#declare White = color red 1 green 1 blue 1
```

```
object {  
  sphere { <0 0 0> 1 }  
  texture { color White }  
}
```

This is much easier to type and to read. The pre-defined element may be used many times in a scene.

You use the keyword "declare" to pre-define a scene element and give it a one-word synonym. This pre-defined scene element is not used in the scene until you invoke its synonym. Shapes, textures, objects, colors, numbers and more can be predefined.

Pre-defined elements may be modified when they are used, for example:

```
#declare Mickey = // Pre-define a union called Mickey  
union {  
  sphere { <0 0 0> 2 }  
  sphere { <-2 2 0> 1 }  
  sphere { <2 2 0> 1 }  
}  
  
// Use Mickey  
object {  
  union {  
    Mickey  
    scale <3 3 3>  
    rotate <0 20 0>  
    translate <0 8 10>  
  }  
  texture {  
    color red 1  
    phong .7  
  }  
}
```

}

This scene will only have one "Mickey", the Mickey that is described doesn't appear in the scene. Notice that Mickey is scaled, rotated, translated, and a texture is added to it. The Mickey synonym could be used many times in a scene file, and each could have a different size, position, orientation, and texture.

Declare is especially powerful when used to create a complex shape or object. Each part of the shape or object is defined separately using declare. These parts can be tested, rotated, sized, positioned, and textured separately then combined in one shape or object for the final sizing, positioning, etc.

Declare can be used anywhere in a scene or include file.

NOTE: Declare is not the same as the C language's define. Declare creates an internal object of the type specified that POV-Ray can recognize when it is prefaced by the object type, while define substitutes like a macro. The only declared object type that doesn't need to be prefaced with a keyword is numeric, which can be used wherever a number is expected.

#include

The language allows include files to be specified by placing the line:

```
#include "filename.inc"
```

at any point in the input file . The filename must be enclosed in double quotes and may be up to 40 characters long (or your computer's limit), including the two double-quote (") characters.

The include file is read in as if it were inserted at that point in the file. Using include is the same as actually cutting and pasting the entire contents of this file into your scene.

Include files may be nested. You may have at most 10 include'd files per scene trace, whether nested or not.

Generally, include files have data for scenes, but are not scenes in themselves. Scene files should end in .pov.

Quick Reference : Shapes and Objects

beizer / bicubic patches

blob

bounded_by

box

camera

clipped_by

composite

difference

direction

height_field

intersection

inverse

light_source

look_at

no_shadow

object

plane

quadric

quartic

right

rotate

scale

sky

smooth_triangle

sphere

spotlight

texture

translate

triangle

union

up

beizer / bicubic patches

```
Syntax: bicubic_patch {  
    patch_type_#  
    [flatness #]  
    u_steps  
    v_steps  
    < Control point 1 > <CP2> <CP3> <CP4>  
    <CP5> <CP6> <CP7> <CP8>  
    <CP9> <CP10> <CP11> <CP12>  
    <CP13> <CP14> <CP15> <CP16>  
}
```

Description: A beizer or bicubic patch is a 3D surface created from a mesh of triangles. Every control point "pulls" the surface towards it. The number of triangles is $u_steps * v_steps$.

blob

Syntax: blob {
 threshold #
 component (strength_val) (radius_val) <component center>
 component (strength_val) (radius_val) <component center>
 [any number of components]
 [sturm]
}

Description: A blobby shape. Components radii should overlap.

bounded_by

Syntax: object {
 ...
 bounded_by {
 shape {...}
 }
}

Description: A bounding shape helps speed rendering time in many cases. The bounding shape is first tested by the raytracer. If the ray does not strike the bounding shape, the raytracer does not need to test or calculate any of the bounded objects.

box

Syntax: box { <x1 y1 z1> <x2 y2 z2> }

Description: A box shape is defined by specifying 2 corners. The first corner (<x1 y1 z1> in the example above) must be smaller than the second corner.

camera

```
Syntax: camera {  
    location <0 0 0>  
    direction <0 0 1>  
    up <0 1 0>  
    right <1.33 0 0>  
    look_at <0 0 0>  
    sky <0 1 0>  
}
```

Description: The camera defines the orientation and location in space of the viewer. The values shown above are the default values. If these items are not specified, the defaults will be used.

clipped_by

```
Syntax: object {  
    ...  
    clipped_by {  
        object {...}  
    }  
}
```

Description: `clipped_by` will "cut off" any part of the object that is outside the clipping shape. This should not be confused with `bounded_by`.

composite

Syntax: composite {
 object1 {...}
 object2 {...}
}

Description: composite will "glue together" 2 or more objects. Essentially, the objects can then be considered as single unit.

difference

```
Syntax: object {  
    difference {  
        shape1 {...}  
        shape2 {...} // This will be "cut out" of shape1  
        shape3 {...} // This will be "cut out" of shape1  
        shape4 {...} // This will be "cut out" of shape1  
        (...)  
    }  
}
```

Description: A difference is a CSG shape. All points in shape1 that are not in shape2 will be included in the final shape. A difference essentially subtracts the shapes following shape1 from shape1. Any number of shapes may be used.

direction

Syntax: direction <# # #>

Description: The direction vector is used in the camera block and specifies the direction the camera is pointing with a vector. Normally, this vector points straight ahead (<0 0 1>) and look_at is used to point the camera. The direction vector should be used to set the "length" of the camera lens. Small values are similar to a wide-angle lens, large values are like a tele-photo lens.

height_field

Syntax: height_field { gif "file.gif" water_level # }

Description: A height field is a rectangular mesh of triangles where the height of a triangle at a certain X,Z point is controlled by the number in a file at that same index. GIF, TGA and POT files may be used as height fields. The water_level is the height below which the untransformed height field is cut off. The untransformed height field is similar in size to:

box { <0 0 0> <1 1 1> }

intersection

Syntax: object {
 intersection {
 shape1 {...}
 shape2 {...}
 }
}

Description: An intersection is a CSG shape composed of 2 or more shapes. All points that are contained in all the included shapes are included in the final shape. In essence, an intersection is the space where all the shapes meet.

inverse

```
Syntax: object {  
    csg_shape_type {  
        shape1 {...}  
        shape2 {... inverse }  
    }  
}
```

Description: Inverse only has effect when using CSG. When inverse is put into the definition of a shape, it turns the shape "inside-out". Essentially, every point that was "inside" the shape is now "outside" and vice versa.

light_source

Syntax: object {
 light_source { <x y z> color red # green # blue #}
}

Description: To shed a little light on a scene, you must provide a light_source. The center of the light is at the vector x, y and z. The red, green, and blue color values define the color of the light which is cast. The light_source itself is invisible. A light source is treated as a shape even though it cannot be seen. It may be used in CSG.

look_at

Syntax: camera {
 ...
 look_at <x y z>
}

Description: look_at defines the point in space at which the camera is pointing, or focused on as defined by the x, y and z parameters.

no_shadow

Syntax: object {
 ...
 no_shadow
}

Description: no_shadow causes an object to be transparent to all light sources. The object will not cast a shadow. This feature is especially useful for enclosing a light source to give the illusion that the light source is actually visible with a shape.

object

Syntax: object {
 shape_type { ... }
 texture { ... }
}

Description: objects are the basic building blocks. An object defines a shape and associated textures. Objects may not be used in CSG, but they may be used in composite objects.

plane

Syntax: plane { <x y z> d }

Description: A plane is a flat surface which is infinite in all directions. The surface normal (or orientation) of the plane is determined by the x, y and z arguments. The d parameter specifies the distance of the plane from the origin in the direction of the surface normal.

plane { <0 1 0> 0 } // XZ plane, a floor

plane { <0 0 1> 10 } // XY plane, a wall

quadric

Syntax: quadric {
 <A B C>
 <D E F>
 <G H I>
 J
}

Description: A quadric is a surface that satisfies the following equation:

Some of the predefined quadrics are :

Shapes.Inc : Ellipsoid
Cylinder_X,Cylinder_Y,Cylinder_Z
QCone_X,QCone_Y,QCone_Z
Paraboloid_X,Paraboloid_Y,Paraboloid_Z

quartic

Syntax : quartic {
 <a00 a01 a02 a03 a04 a05 a06 a07 a08 a09 a10 a11 a12 a13 a14 a15 a16 a17 a18 a19 a20 a21 a22
 a23 a24 a25 a26 a27 a28 a29 a30 a31 a32 a33 a34>
 [sturm]
}

Description : Quartics are complex surfaces that are defined by the following formula :

$$a_{00}x^4 + a_{01}x^3y + a_{02}x^3z + a_{03}x^3 + a_{04}x^2y^2 + a_{05}x^2yz + a_{06}x^2y + a_{07}x^2z^2 + a_{08}x^2z + a_{09}x^2 + a_{10}xy^3 + a_{11}xy^2z + a_{12}xy^2 + a_{13}xyz^2 + a_{14}xyz + a_{15}xy + a_{16}xz^3 + a_{17}xz^2 + a_{18}xz + a_{19}x + a_{20}y^4 + a_{21}y^3z + a_{22}y^3 + a_{23}y^2z^2 + a_{24}y^2z + a_{25}y^2 + a_{26}yz^3 + a_{27}yz^2 + a_{28}yz + a_{29}y + a_{30}z^4 + a_{31}z^3 + a_{32}z^2 + a_{33}z + a_{34}$$

To declare a quartic surface requires that each of the coefficients (a0 -> a34) be placed in order into a single long vector of 35 terms.

right

Syntax: right <x y z>

Description: Used in the camera description, it specifies which direction in the ray tracing universe is the right hand side of the image being generated. Usually, right <1.33 0 0>.

rotate

Syntax: object { ... rotate <x y z> }
 shape { ... rotate <x y z> }
 texture{ ... rotate <x y z> }

Description: rotate will move any element about the origin in x, y and z degrees. It is important to note that if the object is not centered at the origin, it will "orbit" the origin rather than its current center.

scale

Syntax: object { ... scale <x y z> }
shape { ... scale <x y z> }
texture{ ... scale <x y z> }

Description: scale will enlarge or reduce the size of any element. If the values for x, y or z are greater than 1.0, the object is enlarged. If the values are between 0.0 and 1.0, the object is shrunk. Scale may also be use on textures.

NOTE: Scaling by zero will cause an error.

sky

Syntax: camera {
 ...
 sky <x y z>
}

Description: sky describes the orientation of the sky, which is not necessarily the same as the UP direction. If sky is defined, it must be defined before the look_at parameter.

smooth_triangle

Syntax: smooth_triangle {
 <x1 y1 z1> <sn1 sn2 sn3>
 <x2 y2 z2> <sn4 sn5 sn6>
 <x3 y3 z3> <sn7 sn8 sn9>
}

Description: The smooth shaded triangles use a formula called Phong normal interpolation to calculate the surface normal for the triangle. This makes the triangle appear to be a smooth curved surface. In order to define a smooth triangle, however, you must supply not only the vertices, but also the surface normals at those vertices.

For example:

```
smooth_triangle {  
    < 0 30 0 >   <0 .7071 -.7071>  
    < 40 -20 0 >   <0 -.8664 -.5 >  
    < -50 -30 0 >   <0 -.5   -.8664>  
}
```

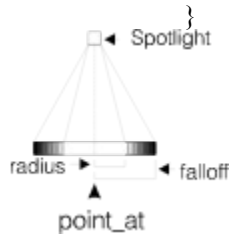
sphere

Syntax: sphere { <x y z> r }

Description: A sphere is a perfectly round shape. Its location in space is defined by the x, y, and z arguments. The radius is determined by the r argument. The width of a sphere will be 2 x r. It cannot be scaled unevenly.

spotlight

```
Syntax: object {  
    light_source {  
        <x y z> // center of light source  
        color red # green # blue #  
        spotlight  
        point_at <x y z>  
        radius #  
        falloff #  
        tightness #  
    }  
}
```



Description: A spotlight `light_source` emulates the behavior of a real spotlight, projecting a cone of light. `point_at` specifies the point in space that the light is aimed at. `radius` is the radius in degrees of the circular "hotspot" at the center of the spotlight's area of effect. `falloff` is the radius in degrees that defines the area where the brightness falls off to zero. Both values may range between 1 and 180. `Tightness` controls how fast the brightness falls off at the edges. Low values cause softer edges, high values create sharper edges.

texture

Syntax: object {
 ...
 texture {
 (texture modifiers)
 }
}

shape { ... texture {...} }

Description: The texture keyword begins a block which describes the appearance of an object, but not the size. See the section on texture modifiers for a list of available modifiers and their defaults. A texture may be used inside a shape or object, but not in a composite object.

translate

Syntax: object { ... translate <x y z> }
 shape { ... translate <x y z> }
 texture{ ... translate <x y z> }

Description: translate moves the element in space by the number of units specified by the x, y and z parameters. Translate is relative to the element's current location. If the element is at <3 4 5> and is translated by <1 -1 1>, the element is moved to <4 3 6>. Normally, translate is used after scale because the scale will "scale" the translate.

triangle

Syntax: triangle { <x1 y1 z1> <x2 y2 z2> <x3 y3 z3> }

Description: A triangle is specified by the coordinates of the 3 vertices. Triangles have no inside or outside, so cannot be used correctly in CSG shapes.

union

```
Syntax: object {  
    union {  
        shape1 {...}  
        shape2 {...}  
    }  
}
```

Description: Union is a CSG shape. A union essentially superimposes two or more shapes to create a single object. All points in the shapes included in a union are included in the final object.

up

Syntax: camera {
 ...
 up <x y z>
}

Description: The up parameter describes the surface normal of the "up" direction. up <0 1 0>, for example would have a "up" direction in the positive y direction.

Quick Reference : Texture Modifiers

agate
alpha
ambient
bozo
brilliance
bump_map
bumps
checker
color
color_map
default texture
#default
dents
diffuse
fog
gradient
granite
image_map
interpolate
ior
leopard
map_type
marble
material_map
max_trace_level
metallic
octaves
onion
phong
phong_size
reflection
refraction
ripples
roughness
specular
spotted
tiles
turbulence
waves
wood
wrinkles

agate

Syntax: agate color_map {...}

Description: agate is a pattern similar to marble. It is always turbulent and ignores the turbulence keyword. agate is used within a texture block.

alpha

Syntax: color red # green # blue # alpha #

Description: The alpha property of a color determines how transparent the color is. Values range from 0.0 (opaque) to 1.0 (totally transparent). Transparency is a filter. Black is always opaque. The color red with alpha 1 will only allow red light through, and so on.

ambient

```
Syntax: texture {  
    ...  
    ambient #  
}
```

Description: ambient determines the amount of light an object receives even if it is in complete shadow. This emulates the light that is just "bouncing around" the room. The default value for ambient is 0.1. Values range from 0.0 to 1.0.

bozo

Syntax: bozo color_map {...}

Description: A splotchy color pattern. Turbulence works on it. It's often used to create clouds.

brilliance

```
Syntax: texture {  
    ...  
    brilliance #  
}
```

Description: brilliance controls the tightness of diffuse illumination on an object and adjusts the appearance of surface shininess. Values from 3.0 to 10.0 can give a shiny or metallic appearance. The default value is 1.0. There is no limit on the brilliance value.

bump_map

Syntax: bump_map { file_type "filename"
map_type # interpolate # bump_size #
(use_color) (use_index) }

Ex: bump_map { gif "rough.gif" map_type 1 interpolate 2 bump_size 3 }

Description: Use a graphic image to simulate bumps on a shape.

bumps

Syntax: texture {
 bumps #
}

Description: bumps gives the surface of an object a bumpy appearance. Values for bumps range from 0.0 (no bumps) to 1.0 (very bumpy).

checker

Syntax: checker color red # green # blue #
color red # green # blue #

Description: the checker pattern gives an object a checkerboard appearance. Two colors must be specified after the checker keyword. These are the colors that will alternate in the checker pattern. The checker option is used within the texture block and works best on planes.

color

Syntax: color red # green # blue # alpha #

Description: colors are specified using the red, green, blue and (optionally) alpha components. The values for each component range from 0.0 to 1.0. If a component is not specified, it is assumed to be 0 (none of that component color). Alpha specifies the transparency of the color.

color_map

```
Syntax: color_map {  
    [start_value end_value color1 color2]  
    [start_value end_value color1 color2]  
    ...  
}
```

Description: A color_map provides a palette for color patterns. A point on the surface is located and it is determined which start_value end_value pair the points falls within. The color is then determined by smoothly blending the colors associated with the start_value end_value pair. Values for start_value and end_value range from 0.0 to 1.0.

Example:

```
color_map {  
    [0 .25 color red 1 color red 1]  
    [.25 .5 color red 1 color green 1]  
    [.5 .75 color green 1 color blue 1]  
    [.75 .76 color Yellow color Orange]  
    [.76 1 color Black color blue 1]  
}
```

default texture

Description: When a texture is first created, POV-Ray initializes it with default values for all options. The default values are:

```
color red 0 green 0 blue 0 alpha 0  
ambient .1  
diffuse .6  
phong 0  
phong_size 40  
specular 0  
roughness .05  
brilliance 1  
metallic FALSE  
reflection 0  
refraction 0  
ior 1  
turbulence 0  
octaves 6  
texture randomness (dither) 0  
phase 0  
frequency 1  
color map NONE
```


#default

Syntax: #default { texture { (modifications to default texture) }

Description: The default textures can be modified by using the #default option. Any textures created after this option has been used will use the new defaults as specified in #default. All other defaults not specified will remain the unchanged.

dents

Syntax: texture {
 ...
 dents #
}

Description: dents will give the object the appearance of being dented. Values for dents range between 0.0 (no dents) and 1.0 (the dentiest) that determines how dented the surface should be.

diffuse

Syntax: texture {
 ...
 diffuse #
}

Description: The diffuse value specifies how the colors in a texture react to light directly shining on it. Higher values make the colors very bright, lower values make the colors more subdued. Values for diffuse range from 0.0 to 1.0. The default value is 0.6.

fog

Syntax: fog { color red # green # blue # distance_val }

Description: Simulate a uniform haze over the entire scene. Fog should be described outside of all other descriptions. Ie. don't put it inside a texture, object, camera, or shape block.

Example: fog { color red 1 green 1 blue 1 200.0 }

gradient

Syntax: gradient <axis vector> color_map {...}

Description: This is a specialized pattern that uses approximate local coordinates of an object to control color map gradients. This texture does not have a default color_map, so one must be specified.

It has a special <X Y Z> triple called the axis vector given after the gradient keyword, which specifies any (or all) axes to perform the gradient action on.

Example: a Y gradient <0.0 1.0 0.0> will give an "altitude color map", along the Y axis. In the axis vector, values given other than 0.0 are taken as 1.0.

For smooth repeating gradients, you should use a "circular" color map, that is, one in which the first color value (0.0) is the same as the last one (1) so that it "wraps around" and will cause smooth repeating gradient patterns.

Scaling the texture is normally required to achieve the number of repeating shade cycles you want.

Transformation of the texture is useful to prevent a "mirroring" effect from either side of the central 0 axes.

Here is an example of a gradient texture which uses a sharp "circular" color mapped gradient rather than a smooth one, and uses both X and Y gradients to get a diagonally-oriented gradient. It produces a dandy candy cane texture!

```
texture {
  gradient < 1.0 1.0 0.0 >
  color_map {
    [ 0 .25 color red 1 green 0 blue 0
      color red 1 green 0 blue 0]
    [.25 .75 color red 1 green 1 blue 1
      color red 1 green 1 blue 1]
    [.75 1 color red 1 green 0 blue 0
      color red 1 green 0 blue 0]
  }
  scale <30 30 30>
  translate <30 -30 0>
}
```

You may also specify a turbulence value with the gradient to give a more irregular color gradient. This may help to simulate things like fire or corona effects. By default, gradient has no turbulence.

granite

Syntax: granite color_map {...}

Description: This will create a granite pattern based on the supplied color_map. granite will respond to the turbulence keyword, but the default is no turbulence. granite is typically used with small scaling values (2.0 to 5.0).

image_map

Syntax: image_map { file_type "filename" alpha (index # or all) #
map_type # interpolate # (once) }

Description: Place a graphic image on a shape as surface coloring.

interpolate

Syntax: image_map { gif "file.gif" interpolate # }

```
interpolate 1 // Norm dist interpolation
interpolate 2 // Bilinear interpolation (best)
```

Description: Smooths the jaggies on image_maps and bump_maps.

ior

Syntax: texture {
 ...
 refraction 1
 ior #
}

Description: The ior is the Index of Refraction. This value determines how far light will bend as it passes through a texture. To be effective, refraction should be set to 1 and the texture must have some transparent colors that use alpha. A value of 1.0 will not bend the light. Some typical ior values are 1.0 for air, 1.33 for water, 1.5 for glass and 2.4 for diamond.

The following ior values are declared in Ior.Inc :

```
Flint_Glass_Ior = 1.71  
Crown_Glass_Ior = 1.51  
Diamond_Ior = 2.47  
Water_Ior = 1.33  
Air_Ior = 1.000292
```

leopard

Syntax: leopard color_map {...}

Description: Uniform spotted color pattern. Turbulence works.

map_type

Syntax: map_type #

Description: Changes the mapping type used with image_map, bump_map, and material map.

0 = Planar

1 = Spherical

2 = Cylindrical

3 = Toroidal (donut)

marble

Syntax: marble color_map {...}

Description: marble creates parallel bands of colors based on the color_map. Adding turbulence will give the appearance of true marble or other types of stones. The default is no turbulence.

material_map

Syntax: material_map { file_type "filename" map_type # [once]
 texture {...} texture {...} (...)
}

Description: Changes the texture on a surface based on the colors in the mapped image.

max_trace_level

Syntax: max_trace_level #

Description: This option will set the number of levels that a ray will be traced. If a ray is reflected or refracted, it creates another ray. This is 1 level. The default value is 5.

metallic

```
Syntax: texture {  
    ...  
    metallic  
}
```

Description: This keyword specifies that the color of specular and phong highlights will be the surface color rather than the color of the light source. This creates a metallic appearance.

octaves

Syntax: octaves #

Description: Affects turbulenc. Default value is 6. Values range from 1 to 16.

onion

Syntax: onion color_map {...}

Description: onion creates a pattern of concentric circles based on the supplied color_map. By default, onion has no turbulence.

phong

```
Syntax: texture {  
    ...  
    phong #  
}
```

Description: The phong keyword causes a bright shiny spot on the object that is the same color as the light source. Values for phong range from 0.0 (none) to 1.0 (very bright at the center of the highlight). There is no phong highlighting by default.

phong_size

Syntax: texture {
 ...
 phong_size #
 phong #
}

Description: The value for phong_size determines the size of the phong highlight of the object. The larger the value, the smaller (tighter) the highlight. The smaller the value, the larger (looser) the highlight. Values range from 1.0 (very dull) to 250 (highly polished). The default phong_size is 40.

reflection

Syntax: texture {
 ...
 reflection #
}

Description: The value of reflection determines how much of the light coming from an object is reflected from other objects in the scene. Values range from 0.0 (no reflection) to 1.0 (a perfect mirror).

refraction

Syntax: texture {
 ...
 refraction #
}

Description: The value for refraction will affect how light passing through transparent textures is treated. Values range from 0 to 1. Lower values will make the transparent portions less transparent. This value will usually be set to 1 with the transparency amounts controlled by alpha. By default, there is no refraction.

ripples

Syntax: texture {
 ...
 ripples #
}

Description: Simulates ripples on a shape's surface.

roughness

Syntax: texture {
 ...
 roughness #
}

Description: The roughness value for a surface determines the size of the specular highlight of that object. Typical values range from 1.0 (Very Rough, large highlight) to 0.0005 (Very Smooth, small highlight). The default, if no roughness is specified, is 0.05.

specular

Syntax: texture {
 ...
 specular #
}

Description: A specular highlight is similar to a phong highlight, but provides a more credible spreading of the highlights near the object horizons. Values for specular range from 0.0 (no highlighting) to 1.0 (bright highlighting). The size of the highlight is determined by the roughness value.

spotted

Syntax: texture {
 spotted color_map {...}
}

Description: spotted pattern is a sort of swirled random spotting of the colors of the object. If you've ever seen a metal organ pipe up close you know about what it looks like (a galvanized garbage can is close...). Play with this one, it might render a decent cloudscape during a very stormy day. No extra keywords are required. With small scaling values, looks like masonry or concrete.

tiles

```
Syntax: texture {  
    tiles {  
        texture {...}  
    }  
    tile2  
        texture {...}  
    }  
}
```

Description: tiles gives an effect similar to checker, but with textures rather than just colors. The textures for tile1 and tile2 may also be layered, but only the first layer will be seen.

turbulence

Syntax: texture {
 ...
 turbulence #
}

Description: turbulence will distort a pattern so that it is not so "perfect". Typical values for turbulence range between 0.0 and 1.0, but any value can be used.

waves

Syntax: waves #

Description: Simulates bumpy waves on a shape's surface.

wood

Syntax: wood color_map {...}

Description: wood used the supplied color map to create concentric cylindrical bands of color centered on the Z axis. Small amounts of turbulence will make the texture look more like real wood. There is no turbulence by default.

wrinkles

Syntax: texture {
 ...
 wrinkles #
}

Description: wrinkles is a bump pattern that will give the appearance of a wrinkled surface. Values for wrinkles range from 0.0 (no wrinkles) to 1.0 (very wrinkled)

Quick Reference : Textures.inc

Syntax : texture { Jade | Red_Marble | White_Marble | etc.... }

Stone Textures

Jade	Swirled jade.
Red_Marble	Pink and gray.
White_Marble	White marble with large "veins".
Blood_Marble	Bloody marble.
Blue_Agate	Dark purple.
Sapphire_Agate	Delicate dark blue stone.
Brown_Agate	Brown on white.
Pink_Granite	Black, light purple, and orange granite. But no pink !
PinkAlabaster	Soft gray/pink alabaster. Subtle !

Sky Textures - Manipulate the scale and the turbulence for best results.

Blue_Sky	Blue sky, with white clouds.
Blue_Sky2	Variation on Blue_Sky.
Blue_Sky3	Small, puffy clouds.
Bright_Blue_Sky	Bright blue sky with bright clouds.
Blood_Sky	Red sky with stormy yellow clouds. Surrealistic.
Apocalypse	Black sky with red and purple clouds. (Another CdW dreamscape.)
Clouds	Clouds with a clear sky. Only clouds. Transparent.

Wooden Textures - Again, try manipulating scale, translation, and turbulence.

Cherry_Wood	A light reddish wood.
Pine_Wood	Light tan wood with greenish growth rings.
Dark_Wood	Dark wood with a greenish hue to it.
Tan_Wood	Light tan background with brown rings.
White_Wood	A very pale wood with tan rings -- kind of balsa-ish.
Tom_Wood	Brown wood - looks stained.

Dan Farmer woods

DMFWood1	Kind of like cedar, maybe?
DMFWood2	Light colored wood.
DMFWood3	Rosewood, very pretty red/black.
DMFWood4	Another light, piney wood.
DMFWood5	Grayish-tan.
DMFLightOak	A very realistic looking fresh white oak surface.
DMFDarkOak	Similar to DMFLightOak, but with a typical oak stain.
Cork	A very realistic looking cork texture

Doug Otwell woods

Yellow_Pine	A beautiful yellow pine.
Rosewood	Deep red/black woodgrain.
Sandalwood	Highly turbulated creamy-gray. Great burlled maple, too!

Surface Textures

Dull	We all know somebody like this guy, don't we? Uses specular.
Shiny	Small, tight highlights. No reflection. Uses specular.
Phong_Dull	Like a soft rubber ball or flat paint.
Phong_Shiny	Some say that phong is less worthy than specular. I use this.
Glossy	Shinier than shiny. Has some reflection included.
Phong_Glossy	Similar to Glossy, but uses phong. Very tight highlight.

Luminous	Good for sky-spheres, no shadows will be cast on it.
Mirror	A perfect mirror. The raytraced sphere classic.
Glass	Clear glass. May want to add Gloss.
Glass2	Probably more like acrylic plastic. Softer gloss than glass.
Glass3	An excellent grayish lead crystal. New with POV-Ray 1.0.
Green_Glass	Glass3 with a pale green tint to it.

Metal Textures

Metal	Add your own color before using this one.
Chrome_Metal	Chrome.
Brass_Metal	Brass.
Gold_Metal	Gold. 24k.
Bronze_Metal	Bronze.
Copper_Metal	Copper.
Silver_Metal	Silver.
Brass_Valley	Hmmmm... something like "Black Hills Gold".
Rusty_Iron	Oxidated iron.
Rust	Oxidated iron.

Special Effect Dept

Candy_Cane	Red and white barber pole.
Peel	Orange and transparent spiral stripes. Use it to emulate the artist M.C. Escher.
X_Gradient	Here as an example more than as a useful texture
Y_Gradient	Here as an example more than as a useful texture
Z_Gradient	Here as an example more than as a useful texture
Water	Requires a sub-surface. Has transparency and ripples.

Quick Reference : Colors.Inc

Syntax : color Gray05 | Blue | Red | etc...

Gray colors

Gray05	color red 0.05 green 0.05 blue 0.05
Gray10	color red 0.10 green 0.10 blue 0.10
Gray15	color red 0.15 green 0.15 blue 0.15
Gray20	color red 0.20 green 0.20 blue 0.20
Gray25	color red 0.25 green 0.25 blue 0.25
Gray30	color red 0.30 green 0.30 blue 0.30
Gray35	color red 0.35 green 0.35 blue 0.35
Gray40	color red 0.40 green 0.40 blue 0.40
Gray45	color red 0.45 green 0.45 blue 0.45
Gray50	color red 0.50 green 0.50 blue 0.50
Gray55	color red 0.55 green 0.55 blue 0.55
Gray60	color red 0.60 green 0.60 blue 0.60
Gray65	color red 0.65 green 0.65 blue 0.65
Gray70	color red 0.70 green 0.70 blue 0.70
Gray75	color red 0.75 green 0.75 blue 0.75
Gray80	color red 0.80 green 0.80 blue 0.80
Gray85	color red 0.85 green 0.85 blue 0.85
Gray90	color red 0.90 green 0.90 blue 0.90
Gray95	color red 0.95 green 0.95 blue 0.95

DimGray	colour red 0.329412 green 0.329412 blue 0.329412
DimGrey	colour red 0.329412 green 0.329412 blue 0.329412
Gray	colour red 0.752941 green 0.752941 blue 0.752941
Grey	colour red 0.752941 green 0.752941 blue 0.752941
LightGray	colour red 0.658824 green 0.658824 blue 0.658824
LightGrey	colour red 0.658824 green 0.658824 blue 0.658824
VLightGrey	color red 0.80 green 0.80 blue 0.80

Pure colors

Clear	colour red 1.0 green 1.0 blue 1.0 alpha 1.0
White	colour red 1.0 green 1.0 blue 1.0
Red	colour red 1.0
Green	colour green 1.0
Blue	colour blue 1.0
Yellow	colour red 1.0 green 1.0
Cyan	colour blue 1.0 green 1.0
Magenta	colour red 1.0 blue 1.0
Black	colour red 0.0 green 0.0 blue 0.0

The rest...

Aquamarine	colour red 0.439216 green 0.858824 blue 0.576471
BakersChoc	color red 0.36 green 0.20 blue 0.09
BlueViolet	colour red 0.62352 green 0.372549 blue 0.623529
Brass	colour red 0.71 green 0.65 blue 0.26
BrightGold	color red 0.85 green 0.85 blue 0.10
Bronze	colour red 0.55 green 0.47 blue 0.14

Bronze2	colour red 0.65 green 0.49 blue 0.24
Brown	colour red 0.647059 green 0.164706 blue 0.164706
CadetBlue	colour red 0.372549 green 0.623529 blue 0.623529
CoolCopper	color red 0.85 green 0.53 blue 0.10
Copper	colour red 0.72 green 0.45 blue 0.20
Coral	colour red 1.0 green 0.498039 blue 0.0
CornflowerBlue	colour red 0.258824 green 0.258824 blue 0.435294
DarkBrown	color red 0.36 green 0.25 blue 0.20
DarkGreen	colour red 0.184314 green 0.309804 blue 0.184314
DarkOliveGreen	colour red 0.309804 green 0.309804 blue 0.184314
DarkOrchid	colour red 0.6 green 0.196078 blue 0.8
DarkPurple	color red 0.53 green 0.12 blue 0.47
DarkSlateBlue	colour red 0.419608 green 0.137255 blue 0.556863
DarkSlateGray	colour red 0.184314 green 0.309804 blue 0.309804
DarkSlateGrey	colour red 0.184314 green 0.309804 blue 0.309804
DarkTan	color red 0.59 green 0.41 blue 0.31
DarkTurquoise	colour red 0.439216 green 0.576471 blue 0.858824
DarkWood	color red 0.52 green 0.37 blue 0.26
DkGreenCopper	color red 0.29 green 0.46 blue 0.43
DustyRose	color red 0.52 green 0.39 blue 0.39
Feldspar	colour red 0.82 green 0.57 blue 0.46
Firebrick	colour red 0.556863 green 0.137255 blue 0.137255
Flesh	color red 0.96 green 0.80 blue 0.69
ForestGreen	colour red 0.137255 green 0.556863 blue 0.137255
Gold	colour red 0.8 green 0.498039 blue 0.196078
Goldenrod	colour red 0.858824 green 0.858824 blue 0.439216
GreenCopper	color red 0.32 green 0.49 blue 0.46
GreenYellow	colour red 0.576471 green 0.858824 blue 0.439216
HuntersGreen	color red 0.13 green 0.37 blue 0.31
IndianRed	colour red 0.309804 green 0.184314 blue 0.184314
Khaki	colour red 0.623529 green 0.623529 blue 0.372549
LightBlue	colour red 0.74902 green 0.847059 blue 0.847059
LightSteelBlue	colour red 0.560784 green 0.560784 blue 0.737255
LightWood	color red 0.91 green 0.76 blue 0.65
LimeGreen	colour red 0.196078 green 0.8 blue 0.196078
MandarinOrange	color red 0.89 green 0.47 blue 0.20
Maroon	colour red 0.556863 green 0.137255 blue 0.419608
MediumAquamarine	colour red 0.196078 green 0.8 blue 0.6
MediumBlue	colour red 0.196078 green 0.196078 blue 0.8
MediumForestGreen	colour red 0.419608 green 0.556863 blue 0.137255
MediumGoldenrod	colour red 0.917647 green 0.917647 blue 0.678431
MediumOrchid	colour red 0.576471 green 0.439216 blue 0.858824
MediumSeaGreen	colour red 0.258824 green 0.435294 blue 0.258824
MediumSlateBlue	colour red 0.498039 blue 1.0
MediumSpringGreen	colour red 0.498039 green 1.0
MediumTurquoise	colour red 0.439216 green 0.858824 blue 0.858824
MediumVioletRed	colour red 0.858824 green 0.439216 blue 0.576471
MediumWood	color red 0.65 green 0.50 blue 0.39
Mica	color Black needed in textures.inc
MidnightBlue	colour red 0.184314 green 0.184314 blue 0.309804
Navy	colour red 0.137255 green 0.137255 blue 0.556863
NavyBlue	colour red 0.137255 green 0.137255 blue 0.556863
NeonBlue	color red 0.30 green 0.30 blue 1.00
NeonPink	color red 1.00 green 0.43 blue 0.78
NewMidnightBlue	color red 0.00 green 0.00 blue 0.61
NewTan	color red 0.92 green 0.78 blue 0.62

OldGold	colour red 0.81 green 0.71 blue 0.23
Orange	colour red 1 green 0.5 blue 0.0
OrangeRed	colour red 1.0 blue 0.498039
Orchid	colour red 0.858824 green 0.439216 blue 0.858824
PaleGreen	colour red 0.560784 green 0.737255 blue 0.560784
Pink	colour red 0.737255 green 0.560784 blue 0.560784
Plum	colour red 0.917647 green 0.678431 blue 0.917647
Quartz	color red 0.85 green 0.85 blue 0.95
RichBlue	color red 0.35 green 0.35 blue 0.67
Salmon	colour red 0.435294 green 0.258824 blue 0.258824
Scarlet	color red 0.55 green 0.09 blue 0.09
SeaGreen	colour red 0.137255 green 0.556863 blue 0.419608
SemiSweetChoc	color red 0.42 green 0.26 blue 0.15
Sienna	colour red 0.556863 green 0.419608 blue 0.137255
Silver	colour red 0.90 green 0.91 blue 0.98
SkyBlue	colour red 0.196078 green 0.6 blue 0.8
SlateBlue	colour green 0.498039 blue 1.0
SpicyPink	color red 1.00 green 0.11 blue 0.68
SpringGreen	colour green 1.0 blue 0.498039
SteelBlue	colour red 0.137255 green 0.419608 blue 0.556863
SummerSky	color red 0.22 green 0.69 blue 0.87
Tan	colour red 0.858824 green 0.576471 blue 0.439216
Thistle	colour red 0.847059 green 0.74902 blue 0.847059
Turquoise	colour red 0.678431 green 0.917647 blue 0.917647
VeryDarkBrown	color red 0.35 green 0.16 blue 0.14
Violet	colour red 0.309804 green 0.184314 blue 0.309804
VioletRed	colour red 0.8 green 0.196078 blue 0.6
Wheat	colour red 0.847059 green 0.847059 blue 0.74902
YellowGreen	colour red 0.6 green 0.8 blue 0.196078

Quick Reference : Stones.Inc

Syntax : texture { Stone1 | Stone2 | Stone3 | etc... }

The following was taken from STONES.INC by Mike Miller.

Mike Miller says : Grnt0 - Grnt29 color maps (generally) contain no alpha values

Grnt0	Gray/Tan with Rose.
Grnt1	Creamy Whites with Yellow & Light Gray.
Grnt2	Deep Cream with Light Rose, Yellow, Orchid, & Tan.
Grnt3	Warm tans olive & light rose with cream.
Grnt4	Orchid, Sand & Mauve.
Grnt5	Medium Mauve Med.Rose & Deep Cream.
Grnt6	Med. Orchid, Olive & Dark Tan "mud pie".
Grnt7	Dark Orchid, Olive & Dark Putty.
Grnt8	Rose & Light Cream Yellows
Grnt9	Light Steely Grays
Grnt10	Gray Creams & Lavender Tans
Grnt11	Creams & Grays Kahki
Grnt12	Tan Cream & Red Rose
Grnt13	Cream Rose Orange
Grnt14	Cream Rose & Light Moss w/Light Violet
Grnt15	Black with subtle chroma
Grnt16	White Cream & Peach
Grnt17	Bug Juice & Green
Grnt18	Rose & Creamy Yellow
Grnt19	Gray Marble with White feather Viens
Grnt20	White Marble with Gray feather Viens
Grnt21	Green Jade
Grnt22	Clear with White feather Viens (has some transparency)
Grnt23	Light Tan to Mauve
Grnt24	Light Grays
Grnt25	Moss Greens & Tan
Grnt26	Salmon with thin Green Viens
Grnt27	Dark Green & Browns
Grnt28	Red Swirl
Grnt29	White, Tan, w/ thin Red Viens

Mike Miller says : Grnt0A - Grnt24A color maps containing alpha

Grnt0a	Translucent Grnt0
Grnt1a	Translucent Grnt1
Grnt2a	Translucent Grnt2
Grnt3a	Translucent Grnt3
Grnt4a	Translucent Grnt4
Grnt5a	Translucent Grnt5
Grnt6a	Translucent Grnt6
Grnt7a	Translucent Grnt7
Grnt8a	Aqua Tints
Grnt9a	Alpha Creams With Cracks
Grnt10a	Alpha Cream Rose & light yellow
Grnt11a	Alpha Light Grays
Grnt12a	Alpha Creams & Tans

Grnt13a	Alpha Creams & Grays
Grnt14a	Cream Rose & light moss
Grnt15a	Alpha Sand & light Orange
Grnt16a	Cream Rose & light moss (again?)
Grnt17a	???
Grnt18a	???
Grnt19a	Gray Marble with White feather Viens with Alpha
Grnt20a	White Feather Viens
Grnt21a	Thin White Feather Viens
Grnt22a	???
Grnt23a	Transparent Green Moss
Grnt24a	???

Mike Miller says : complete texture statements - edit to your scene & lighting situations.

Stone1	Deep Rose & Green Marble with large White Swirls
Stone2	Light Greenish Tan Marble with Agate style veining
Stone3	Rose & Yellow Marble with fog white veining
Stone4	Tan Marble with Rose patches
Stone5	White Cream Marble with Pink veining
Stone6	Rose & Yellow Cream Marble
Stone7	Light Coffee Marble with darker patches
Stone8	Gray Granite with white patches
Stone9	White & Light Blue Marble with light violets
Stone10	Dark Brown & Tan swirl Granite with gray undertones
Stone11	Rose & White Marble with dark tan swirl
Stone12	White & Pinkish Tan Marble
Stone13	Medium Gray Blue Marble
Stone14	Tan & Olive Marble with gray white veins
Stone15	Deep Gray Marble with white veining
Stone16	Peach & Yellow Marble with white veining
Stone17	White Marble with gray veining
Stone18	Green Jade with white veining
Stone19	Peach Granite with white patches & green trim
Stone20	Brown & Olive Marble with white veining
Stone21	Red Marble with gray & white veining
Stone22	Dark Tan Marble with gray & white veining
Stone23	Peach & Cream Marble with orange veining
Stone24	Green & Tan Moss Marble

Quick Reference : RTAG Script Language

NOTE: This reference was taken from RTAG.DOC. This is NOT an oficial release from Phillip H. Sherrod. All information is COPYRIGHT PHILLIP SHERROD.

Assignment Statement

Arithmetic And Logical Expressions

Auxiliary Data Files

Break Statement

Comments

Continuation Of Lines

Declaration Of Variables (The VAR Statement)

Epilog Statement

For Statement

If Statement

Nextframe Statement

Output File Declarations

Stop Statement

Subchar Statement

While Statement

Write Statements

ASSIGNMENT STATEMENT

The assignment statement is an executable statement that evaluates an expression and assigns its value to a variable. The syntax for an assignment statement is:

`variable = expression`

where "variable" is a previously declared variable and "expression" is a valid arithmetic or logical expression following the rules explained earlier. If the expression involves a relational comparison operator (e.g., `<`, `>`, `>=`, etc.) or a logical operation (`&&`, `||`, `!`), the value 1 is used for true and 0 for false.

ARITHMETIC AND LOGICAL EXPRESSIONS

Much of the power of RTAG comes from its ability to perform mathematical calculations. This is important for any type of animation generation, but is especially critical for physical system simulations. This section explains the arithmetic operators and built in functions that may be used in arithmetic expressions.

Arithmetic Operators

The following arithmetic operators may be used in expressions:

+	addition
-	subtraction or unary minus
*	multiplication
/	division
%	modulo
** or ^	exponentiation

The following operators compare two values and produce a value of 1 if the comparison is true, or 0 if the comparison is false:

==	Equal
!=	Not equal
<=	Less than or equal
>=	Greater than or equal
<	Less than
>	Greater than

The following logical operators may be used:

!	Logical NOT (negates true and false)
&&	AND
	OR

Operator precedence, in decreasing order, is as follows: unary minus, logical NOT, exponentiation, multiplication, division and modulo, addition and subtraction, relational (comparison), logical (AND and OR). Parentheses may be used to group terms.

Numeric Constants

Numeric constants may be written in their natural form (1, 0, 1.5, .0003, etc.) or in exponential form, n.nnnEppp, where n.nnn is the base value and ppp is the power of ten by which the base is multiplied. For example, the number 1.5E4 is equivalent to 15000. All numbers are treated as "floating point" values, regardless of whether a decimal point is specified or not. As a convenience for entering time values, if a value contains one or more colons, the portion to the left of the colon is multiplied by 60. For example, 1:00 is equivalent to 60; 1:00:00 is equivalent to 3600.

Built-in Constant

The symbolic name "PI" is equivalent to the value of pi, 3.14159... You may write PI using either upper or lower case.

Built in Functions

The following functions are built into RTAG and may be used in

expressions:

ABS
ACOS
ASIN
ATAN
CEIL
COS
COSH
COT
CREATE
CSC
DEG
EXP
FAC
FLOOR
GAMMA
INT
LINEAR
LOG
LOG10
MAX
MIN
MOD
MOREDATA
NPD
OPEN
PAREA
PULSE
RAD
RANDOM
ROUND
SEC
SEL
SIN
SINH
SPLINE
SQRT
STEP
TAN
TANH

$\text{ABS}(x)$ -- Absolute value of x .

ACOS(x) -- Arc cosine of x. Angles are measured in degrees.

ASIN(x) -- Arc sine of x. Angles are measured in degrees.

ATAN(x) -- Arc tangent of x. Angles are measured in degrees.

CEIL(x) -- Ceiling of x (an equivalent name for this function is INT). Returns the smallest integer that is at least as large as x. For example, CEIL(1.5)=2; CEIL(4)=4; CEIL(-2.6)=-2.

$\text{COS}(x)$ -- Cosine of x . Angles are measured in degrees.

$\text{COSH}(x)$ -- Hyperbolic cosine of x .

COT(x) -- Cotangent of x. (COT(x) = 1/TAN(x)). Angle in degrees.

CREATE(file) -- Create a new auxiliary data file. The value returned by this function is a file number that can be used with subsequent WRITE statements. See the description of auxiliary file I/O for additional information.

CSC(X) -- Cosecant of x. ($\text{CSC}(x) = 1/\text{SIN}(x)$). Angle in degrees.

DEG(x) -- Converts an angle, x , measured in radians to the equivalent number of degrees.

EXP(x) -- e (base of natural logarithms) raised to the x power.

FAC(x) -- x factorial (x!). The FAC function is computed using the GAMMA function
(FAC(x)=GAMMA(x+1)) so non-integer argument values may be computed.

FLOOR(x) -- Floor of x. Returns the largest integer that is less than or equal to x. For example,
FLOOR(2.5)=2; FLOOR(4)=4; FLOOR(-3.6)=-4.

GAMMA(x) -- Gamma function. Note, GAMMA(x+1) = x! (x factorial).

$\text{INT}(x)$ -- Ceiling of x (an equivalent name for this function is CEIL). Returns the smallest integer that is at least as large as x . For example, $\text{INT}(1.5)=2$; $\text{INT}(4)=4$; $\text{INT}(-2.6)=-2$.

`LINEAR(t, t1,x1, t2,x2, ... tn,xn)` -- Perform linear interpolation between a set of points. If the value of a function is x_1 at some point t_1 , x_2 at t_2 , etc., this function determines the value at some arbitrary point, t , where t falls in the range (t_1, t_n) . The first argument to the `LINEAR` function is the value t at which the interpolation is to be performed. The remaining arguments are (t_i, x_i) data pairs. For example, consider the function `LINEAR(t, 0,0, 1,6, 2,4)`. This specifies that when t is 0, the value of the function is 0, when t is 1, the value is 6, and when t is 2 the value is 4. If this function is called with a t value of 0.5, the returned value of the function will be 3. If the function is called with a t value of 1.5, the returned value will be 5. You can use expressions as arguments to this function. For example, the following function invocation is valid: `LINEAR(t, starttime,basex, starttime+3,midx, endtime,midx*2)`. If the function is called with a t value that is outside the range (t_1, t_n) (i.e., beyond either end point), then the nearest end point and the second closest point are used to extrapolate to the specified t value. See also the description of the `SPLINE` function.

$\text{LOG}(x)$ -- Natural logarithm of x .

$\text{LOG}_{10}(x)$ -- Base 10 logarithm of x .

$\text{MAX}(x_1, x_2)$ -- Maximum value of x_1 or x_2 .

$\text{MIN}(x_1, x_2)$ -- Minimum value of x_1 or x_2 .

MOD(x1,x2) -- x1 modulo x2. The MOD function calculates the floating-point remainder, f, of $x1/(i*x2)$ such that $x1=i*x2+f$, where i is an integer; f has the same sign as x1, and the absolute value of f is less than the absolute value of x2.

`MOREDATA(file)` -- This function checks to see if there is another data record available in the external file whose file number is specified as the argument. If there is a record available, this function returns the value true (1), if not, it returns false (0). This function does not consume any data in the file, you must use the `READ` statement to actually read the next record. You can call `MOREDATA` any number of times without disturbing the next record in the file. See the section on auxiliary file I/O for additional information.

$NPD(x, \text{mean}, \text{std})$ -- Normal probability distribution of x with specified mean and standard deviation. X is in units of standard deviations from the mean.

OPEN("file name") -- Open an existing file for reading. The value returned by this function is a file number that can be used with subsequent READ statements. See the description of auxiliary file I/O for additional information.

PAREA(x) -- Area under the normal probability distribution curve from $-\infty$ to x. (i.e., integral from $-\infty$ to x of NORMAL(x)).

$\text{PULSE}(a,x,b)$ -- Pulse function. If the value of x is less than a or greater than b , the value of the function is 0. If x is greater than or equal to a and less than or equal to b , the value of the function is 1. In other words, it is 1 for the domain (a,b) and zero elsewhere. If you need a function that is zero in the domain (a,b) and 1 elsewhere, use the expression $(1-\text{PULSE}(a,x,b))$.

$\text{RAD}(x)$ -- Converts an angle measured in degrees to the equivalent number of radians.

RANDOM() -- Returns a random value uniformly distributed in the range 0 to 1. You can use the srand system variable to specify a starting seed.

ROUND(x) -- Rounds x to the nearest integer. For example, ROUND(1.1)=1; ROUND(1.8)=2;
ROUND(-2.8)=-3;

SEC(x) -- Secant of x. ($\text{SEC}(x) = 1/\text{COS}(x)$). Angle in degrees.

$\text{SEL}(a_1, a_2, v_1, v_2)$ -- If a_1 is less than a_2 then the value of the function is v_1 . If a_1 is greater than or equal to a_2 , then the value of the function is v_2 .

SIN(x) -- Sine of x. Angles are measured in degrees.

$\text{SINH}(x)$ -- Hyperbolic sine of x .

`SPLINE(t, t1,x1, t2,x2, ... tn,xn)` -- Perform a cubic spline interpolation between a set of points. A spline is a smooth path (continuous first and second derivatives) that passes through a set of points. The SPLINE function is very useful in animations because it allows you to easily construct a smooth path for the motion of some object (or the camera). If the value of a function is x_1 at some point t_1 , x_2 at t_2 , etc., this function determines the value at some arbitrary point, t , where t falls in the range (t_1, t_n) . The first argument to the SPLINE function is the value t at which the interpolation is to be performed. The remaining arguments are (t_i, x_i) data pairs. You can use expressions as arguments to this function. For example, the following function invocation is valid: `SPLINE(t, starttime, basex, starttime+3, midx, endtime, midx*2)`. If the function is called with a t value that is outside the range (t_1, t_n) (i.e., beyond either end point), then the value of the function at the nearest end point and the slope of the function at that point are used to extrapolate in a linear fashion to the specified t value. See also the description of the LINEAR function.

SQRT(x) -- Square root of x.

$\text{STEP}(a,x)$ -- Step function. If x is less than a , the value of the function is 0. If x is greater than or equal to a , the value of the function is 1. If you need a function which is 1 up to a certain value and then 0 beyond that value, use the expression $\text{STEP}(x,a)$.

TAN(x) -- Tangent of x. Angles are measured in degrees.

TANH(x) -- Hyperbolic tangent of x.

AUXILIARY DATA FILES

RTAG provides statements to open, close, read from, and write to auxiliary data files. This is useful if some other program has computed object positions or other data that affects the animation.

Up to 30 auxiliary data files can be open at once. The files are identified by numbers (also called file "handles") that are assigned by RTAG at the time that the file is opened. The file numbers can be reused by closing and reopening files. You must use the VAR statement to declare variables to hold the file numbers.

Opening a File for Reading

The form of the statement used to open an existing file for reading is:

```
variable = OPEN("file name")
```

where 'variable' is a variable you have previously declared with a VAR statement. When RTAG opens the file it generates a unique file number and assigns it to the variable on the left of the equal sign. This file number can be used subsequently in READ statements. An example of this statement is

```
infile = open("spiral.dat")
```

Creating an Output File

The form of the statement used to create an auxiliary output file is

```
variable = CREATE("file name")
```

When RTAG creates the file it assigns a unique file number to the specified variable. This file number can be used subsequently in WRITE statements. An example of this statement is:

```
outfile = create("trace.dat")
```

Closing an Auxiliary File

The statement used to close an auxiliary file is:

```
CLOSE(file)
```

where 'file' is a variable that has a file number. For example: close(outfile)

Any open files are closed automatically when your program terminates.

Reading from an Auxiliary File

You may read data from an open auxiliary file by using a READ statement of the following form:

```
READ (file) variable1,variable2,...
```

Each READ statement reads the next line of data from the file and assigns numeric values to the list of variables you specify. There must be at least as many data values on the line as the number of variables specified with the READ statement (additional values are ignored). The data values may be separated by spaces, tabs, and/or commas. Numeric values may be specified in the same form as numbers within an RTAG animation control file (natural, exponential, or dd:dd:dd).

The MOREDATA function can be used to control how many records are read from the file. The value of MOREDATA(file) is true (1) if there is another, unread, record available in the file; its value is false (0) if there is no more data available in the file. MOREDATA does not consume any data, it just checks for the availability of data. The following statements would read all data in a file and generate a frame file for each record:

```
var f,xpos,ypos
  f = open("indata.dat")
  while (moredata(f)) do nextframe
    read (f) xpos,ypos
    owrite "#declare xpos=`xpos`" owrite "#declare ypos=`ypos`"
  endwhile
```

Writing to an Auxiliary File

The form of the statement used to write to an auxiliary file is: WRITE (file) "string"

[,expression1,expression2,...]

where 'file' is a variable containing a file number assigned by a previously executed CREATE function.

"string" is a quoted string of the same form as described for the PRINT, OWRITE and BWRITE statements.

The string may include variable and expression replacement operators. Expression1, expression2, etc. are

optional expressions whose values are to be substituted into the string where "%w.dlf" sequences occur. The following statements illustrate the use of WRITE statements:

```
var f,x,y
f = create("sine.out")
for y = 0 while y < 2*pi step .1 do
x = sin(y)
write (f) "x=`x`, y=`y`"
endfor
close (f)
```

BREAK STATEMENT

The BREAK statement can be used in FOR and WHILE loops to terminate the loop and cause control to transfer to the statement beyond the end of the loop. The following is an example of a BREAK statement:

```
for time=0 while time<60 do
    x = x + delta * xspeed
    if (x > 10) then
        break
    endif
endfor
```

COMMENTS

The beginning of a comment is denoted with `"/"` (two consecutive slash characters). Everything from the `"/"` sequence to the end of the line is treated as a comment. Comments may be on lines by themselves or on the ends of other statements. The following lines illustrate both types of comments:

```
// Begin main simulation loop  
y = y + timestep * yvelocity // Advance y position
```

CONTINUATION OF LINES

Unlike programming languages such as C, the RTAG language is line oriented. That is, statements are normally terminated by the end of the line rather than using a terminating character such as a semicolon. If you need to enter a command that is too long to fit on a line (i.e., more than 80 characters), you can continue it on subsequent lines by placing "\\" (two consecutive left leaning slashes) at the end of each line that is to be continued. You can also use tabs to indent lines or align portions of statements. The following is an example of a continued line:

```
y = spline(t, 0,0, 1,2, 5,8, 10,9, 12,10, \  
          15,20, 17,25, 20,30)
```

DECLARATION OF VARIABLES (The VAR Statement)

All variables used in your animation control file must be declared before they are used. This is done using the VAR statement which may optionally assign an initial value to the variable. If no initial value is specified, 0 is used by default. VAR is a declarative statement: that is, it declares the existence of the variable and assigns an initial value, it does not reassign the value to the variable if the VAR statement appears in a loop.

Variable names may be up to 30 characters long. They ARE case sensitive. Keywords (FOR, WHILE, VAR, etc.) and library function names (SIN, COS, ABS, etc.) are NOT case sensitive.

The syntax of the VAR statement is:

```
VAR variable1[=value], variable2[=value], ...
```

You may use as many VAR statements as you need. The following are examples of this statement:

```
var x, y, speed=5
```

```
var Acceleration=2.5
```

In addition to your own variables, RTAG provides several system variables. If you do not provide VAR statements for these variables, they are defined automatically at the start of a run and RTAG assigns their values.

Variable	Default	Meaning
curframe	0	The current frame number. Set by NEXTFRAME.
firstframe	1	The first frame that will generate output.
lastframe	999999	The last frame that will generate output.
stepinc	1	The step increment between frames.
srand	1	Seed for random number generator.

These variables may be used in your animation control file just like ordinary variables. If you want to set different default values for firstframe, lastframe, stepinc or srand, use the VAR statement and specify an initial value. For example, the following commands would cause frames 20 through 40 to be output:

```
var firstframe = 20
```

```
var lastframe = 40
```

The default values, or the values specified by VAR statements, can be overridden by qualifiers on the command line. /F sets firstframe, /L sets lastframe, and /S sets stepinc.

The srand system variable is used to set a starting "seed" value for the RANDOM() function. The default value for srand is 1. For a given starting seed value, the pseudorandom sequence is always the same.

EPILOG STATEMENT

The EPILOG statement (which may also be spelled EPILOGUE) informs RTAG that the last frame has been completed and that subsequent BWRITE statements are unconditionally to generate output regardless of the setting of the firstframe, lastframe, and stepinc values. Otherwise BWRITE statements do not generate output if they occur after the last frame, or between frames if stepinc is not 1. Commonly the BWRITE statement is used after EPILOG to write some "end-of-all-frames" termination statements to the batch file, such as commands to run DTA to build the animation sequence.

FOR STATEMENT

The FOR statement is a looping control statement similar to the WHILE statement; however, the FOR statement assigns an initial value to a variable at the top of the loop and increments the variable by a specified amount at the bottom of the loop. The form of the FOR statement is:

```
FOR variable=expression1 WHILE expression2 [STEP expression3] DO << controlled statements >>
ENDFOR
```

At the beginning of the loop expression1 is evaluated and its value is assigned to the variable, then the loop begins. At the top of the loop expression2 is evaluated. If its value is false (0) control transfers to the statement beyond the end of the loop. If its value is true (non zero) the controlled statements are executed.

The "STEP expression3" clause is optional. If specified, expression3 is evaluated at the end of the loop and its value is algebraically added to the value of the loop variable (the step value may be negative). If the STEP clause is not specified, 1 is added to the loop variable. The loop

continues to be executed until expression2 is false or a BREAK statement is executed within the loop.

The following is an example of a FOR statement:

```
for time=starttime while time<endtime step timestep do
    << controlled statements >>
endfor
```


NEXTFRAME STATEMENT

The NEXTFRAME statement performs the following actions: (1) close the currently open output file, if any; (2) increment the current frame number (and curframe variable); (3) if an OFILE statement was used, open a new output file with the current frame number substituted for the "###" characters in the file name. You must execute a NEXTFRAME statement before you can use an OWRITE statement. PRINT, BWRITE and WRITE statements can be executed before the first NEXTFRAME command. The last output file is closed when your program stops.

OUTPUT FILE DECLARATIONS

RTAG is capable of producing multiple files during each run. Most of these files are optional and will only be produced if you include the appropriate statements in your control file. The listing file is always produced. It has the same name as the animation control file but with an extension of ".LST".

Most RTAG runs will produce a batch file to drive the ray tracing program. The batch file may contain commands to generate the appropriate include file for each frame or you may choose to have RTAG generate the full set of include files separate from the batch file. If you wish to generate a batch file you must place a command of the following form in your control file:

BFILE name

where 'name' is the name of the batch file. The default extension is ".BAT".

The batch file normally contains a set of commands to render each frame. Typically, the commands for each frame perform the following actions:

1. Generate an include file with object position information.
2. Run the ray tracer using the current include file.

The BWRITE command is used to write to the batch file. The batch file is optional and is generated only if you use a BFILE statement.

If you wish to have RTAG generate a set of include files, you must use the following command to declare the names of the files:

OFILE name

If the OFILE statement is used, one output file is generated for each frame. The output file name must include a string of one to seven '#' characters. These characters are replaced by the current frame number when the file is opened. For example, consider the following command:

ofile bounc###.inc

Then for frame 27 the name of the created file will be BOUNC027.INC. The default extension is ".INC".

The NEXTFRAME

command closes the current output file, increments the frame number, and opens the next output file (assuming the OFILE statement was specified). The last output file is closed when the program stops. The OWRITE statement writes lines to the currently open output file.

RTAG can also create or read from auxiliary data files. This is described in a subsequent section.

STOP STATEMENT

The STOP statement terminates the execution of RTAG and closes all output files. An implicit STOP occurs if you "fall through" the bottom of your animation control file. The following is an example of the STOP statement:

```
while (curframe < 60) do
    << controlled statements >>
    if (xposition1 >= xposition2) then
        stop
    endif
endwhile
```

SUBCHAR STATEMENT

The SUBCHAR statement specifies which character is to be used in write strings to enclose the names of variables whose values are to be substituted in the string. The default character is "'" -the accent character which is usually on the same key as the tilde character. The form of the statement is:

SUBCHAR char

where "char" is a single character.

WHILE STATEMENT

The WHILE statement loops until the controlling expression becomes false (0) or a BREAK statement is executed within the loop. The form of the WHILE statement is:

```
WHILE (expression) DO  
    << controlled statements >>
```

```
ENDWHILE
```

Each time around the loop the expression is evaluated. If it is true (non zero) the controlled statements are executed up to the ENDWHILE statement and then the process repeats until the expression becomes false. If a BREAK statement is executed within the loop, control transfers to the statement that follows the ENDWHILE statement. The following is an example of a WHILE statement:

```
while (x < 5) do  
    x = x + xmove  
    y = y + ymove  
endwhile
```

WRITE STATEMENTS

RTAG has three output statements. These statements and the files they write to are as follows:

PRINT -- Listing file (also displayed on screen).

OWRITE -- Output file (OFILE).

BWRITE -- Batch file (BFILE).

Since the form for these statements is the same (except for the keyword), the PRINT statement will be used in the illustrations. The form of the statements is:

```
PRINT "string"[,expression1, expression2,...]
```

where "string" is a quoted text string that may include variable and expression replacement operators. You can cause the current value of a variable to be substituted into a string by inserting the variable name in the string enclosed by "~" characters. Note, this is not the apostrophe, it is the accent character that is on the same key as the tilde on most keyboards. You can use the SUBCHAR statement (see below) to change the character used to mark a variable value substitution. For example, the following statement prints the values of the xpos and ypos variables:

```
print "X position is `xpos`, and Y position is `ypos`"
```

When this statement is executed `xpos` and `ypos` are replaced by the current values of the xpos and ypos variables.

If the sequence `###` (1 to 7 '#' characters) appears in the string, the pound signs are replaced by the current frame number with leading zeros. In addition, the strings `ofile` and `bfile` are replaced by the names of the output file, and batch file respectively. The string `file` is replaced by the name of the input animation command file with any device, directory, and extension removed.

You can also substitute the values of expressions. To do this, use the C programming notation "%w.dlf" where 'w' is the overall field width and 'd' is the number of decimal places. For example, the following statement prints the value of an expression:

```
print "The diagonal distance is %5.1lf", sqrt(xd^2 + yd^2)
```

Each WRITE statement generates a line of output terminated by carriage-return, line-feed.

POV-Ray Options

These options tell POV-Ray how to render the scene. The command-line equivalents are shown after the option name.

Width (+w####)

Specify the width of the image in pixels.

Height (+h####)

Specify the height of the image in pixels.

Quality (+q#)

Specifies the quality of the image. Lower quality speeds up rendering, but produces uglier images.

Anti-Aliasing Threshold (+a####)

Specifies the anti-aliasing threshold. During rendering, POV-Ray will calculate the difference between the rendered pixel and its neighbors. If the difference is higher than the threshold, POV-Ray will attempt to smooth the pixel by super-sampling it. A threshold of 0.0 means that every pixel will be super-sampled, and a threshold of 1.0 means that no pixel will be super-sampled. According to the POV-Ray manual, good values are around 0.2 to 0.4.

Disable Anti-Aliasing (-a)

Check this checkbox if you want to disable anti-aliasing.

Display image stats while rendering (+v)

If on, POV-Ray will display image statistics while rendering.

Display image graphically while rendering (+d)

If on, POV-Ray will display the image on-screen as it is being traced.

Allow abort with keypress (+x)

Check this checkbox if you want to be able to abort POV-Ray.

Pause and wait for keypress after tracing image (+p)

If this checkbox is checked, POV-Ray will pause and wait for you to press a key when it has traced the image.

Partial Rendering

Activated (+c)

This option selects partial rendering. When activated, POV-Ray will read and display the unfinished image and continue rendering. This is useful if your system hangs and forces you to reboot or restart, since you can continue rendering where you left off.

Start at line (+s####)

Specifies the line number POV-Ray should start on when using **Partial Rendering**.

Stop at line (+e####)

Specifies the line number POV-Ray should stop at when using **Partial Rendering**.

Output file format

Targa, Raw and Dump radiobuttons (+ft,+fr,+fd)

Use these radiobuttons to select the output file format. Targa creates a standard Targa 24bit uncompressed

image file. Dump creates a QRT-style file. **Raw creates three files**; one for red, one for blue and one for green. These files have the extensions .r8 , .g8 and .b8 respectively. Raw file format is used with **PICLAB**.

Disable output to file (-f)

If you check this checkbox, POV-Ray will **not** write an output file. This option should be used in conjunction with **Display image graphically while rendering** and **Pause and wait for keypress after tracing image**.

Output Buffering

Buffer Size (+b####)

The **buffer size** option allows you to assign large buffers to the output file. This reduces the amount of time spent writing to the disk and prevents unnecessary wear. If this parameter is zero, then as each scanline is finished, the line is written to the file and the file is flushed. On most systems, this operation insures that the file is written to the disk so that in the event of a system crash or other catastrophic event, at least part of the picture has been stored properly and retrievably on disk.(see also the **Partial Rendering** option.) 30 is a good value to use in order to speed up small renderings.

Directory Options Dialog

Separate the directories with a semicolon (;).

For example :

C:\POVRAY;C:\POVRAY\SAMPLE;C:\POVRAY\INCLUDE

POV-Ray dirs

Here you specify which directories POV-Ray should search when looking for a file.

Output dir

Use this option to specify where the output file should be placed. **You may only specify one (1) directory !**

Output File name

This is equivalent to the **+o** option. If you specify a filename, all output from POV-Ray will be redirected to this file.

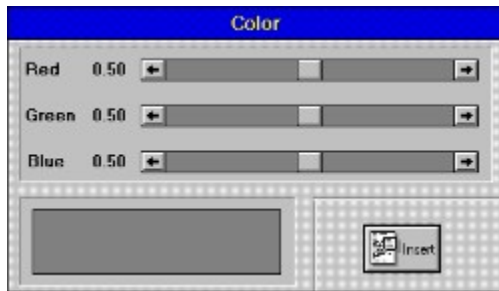
Targa Viewer

The path to your favorite Targa viewer.

The Color Palette

This is used as a tool to specify color. Use the scroll bars to select the color. To the left of the scroll bars, you can see the RGB value for the color. In the lower right of the window, you'll see Windows' approximation of the selected color. If you press the **Insert** button, the current color will be inserted into the scene description file.

The palette will disappear when you press **Insert** or when you switch to another window.



Insert camera definition

Can't remember the syntax for the camera definition ? **Look no further !**

Just fill in the values and press insert!

location

The position of the camera.

direction

The "lens length" of the camera.

up

Tells POV-Ray the "up" or "top of image" direction .

right

Used to specify the image's aspect ratio. Usually set to 1.333.

look_at

Where the camera should be pointed at. This point should be the "center of attention" of your image.

sky

The sky direction.

Insert light_source

Location

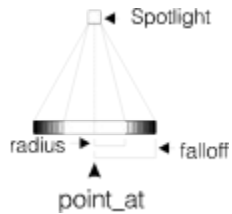
The position of the light source.

Color

The color of the light.

Spotlight

Check this checkbox if you want to insert a spotlight light source. If you do then you must also specify the following parameters :



Point at

The spot where the spotlight is aimed at.

Radius

The size of the white "hotspot" in the center in degrees.

Falloff

The distance from the center to the point where the light intensity is 0, measured in degrees.

Tightness

The tightness value specifies how fast the light falls off at the edges. Higher values makes the edges sharper.

The RTAG Options dialog

First frame

The first frame to generate output. Output is discarded for frames prior to this number.

Last Frame

The last frame to generate output. Output is discarded for frames following this number.

Steps between frames

Specifies the number of steps between frames. Default is 1.

Include file names

The names of the include files. The name **must** include 1 to 5 '#' characters.

Batch File name

Specifies the name of the batch file.

Disable Output

Specifies that no output is to be written to the batch and include files. This is useful when you are debugging an animation control file and want to test it without generating the output files. Output generated by PRINT and WRITE commands always is written, even if this option is specified.

List control file

Causes the entire control file to be listed on the screen and written to the listing file. Normally only error messages and output generated by PRINT statements are displayed.

Display each statement before executing

Specifies that each source statement is to be displayed before it is executed. This is useful for debugging purposes.

The DISABLE checkboxes

Since you can specify some of the options listed here in your .RTA file, and since these options will **override** all the settings in the RTA file, you have the option to disable any one of them. In order to make life easier, you should specify all options in your .RTA file, since it makes the RTA file easier to share with others.

Contacting the author

Raytrace Workbench is constantly evolving. If **you** would like to have a special feature included, **write!**

Send all your questions, suggestions and opinions about **Raytrace Workbench** to :

Leo Sutic
CIS ID: 100040,600

