PRM Toolcase Documentation

Contenu

Shading

Version 0.02

It's possible to shade PRM using PRM tool, this feature has been introduced with version 0.02 and it combines "rvshade" and "autoshade" (CarLighting) tools.

Shade using one color

This is quite similar to rvshade, it's able to take either 3 values of colors, 4 or even 1.
In normal case, it takes 3 values of color intensity as represented in 8-bit RGB color system.

The color intensity values of "Red" (R), "Green"(G) and "Blue"(B), can each represented with a byte value, that's an integer value ranging from 0 to 255. When value of R,G,B are all 0 it's pitch black and it's pure white when they're all 255.

syntax

```
prm shade file.prm R G B A
prm shade file.prm R G B
prm shade file.prm gray
```

The Alpha value(A) represent the opacity and so when A=255 it's opaque and when it's 0 it's fully transparent. It goes without saying that A=128 means it's half-transparent. In case **alpha was not set**, it's assumed to be **A=255**.

Using a "gray" means setting R=G=B. In other words, when gray=128 this is exactly said as R=G=B=128.

Examples:

`Prm shade body.prm 230 230 255` *This should make body.prm blueish white*

`Prm shade body.prm 192` *Set body.prm as R=G=B=192*

`Prm shade body.prm 192 192 192 160` *Shade body to gray-192, and with opacity of 160*100/255 = 62.75%*

AutoShade

Maybe RIKO, CarLighting and Car Studio can perform better results than this command, this is then another alternative and is slightly faster than all of above + 3Ds max and Blender.

Autoshade tries to shade the PRM file using Y-normal values stored inside PRM files. If you're not sure about what it does mean, let's say it's better than rvshade's result and capable of outputing a more realistic result than rvshade and prm shade.

AutoShading can be performed either using vertex (by default used when it's precised) or polygon. AutoShade Vertex will be able to give a smooth shading while polygon shading is more to give shading in tiles.

syntax

```
prm autoshade file.prm vertex
prm autoshade file.prm polygon
prm autoshade file.prm
prm autoshadeinv file.prm vertex
prm autoshadeinv file.prm polygon
prm autoshadeinv file.prm
```

Some normals may be inverted, and if that's the case then it's wise to use "**autoshadeinv**" instead of "autoshade" to invert the shade. This should be applied when "autoshade" gives dark results on top of the PRM.

Examples:

```
Prm autoshade body.prm
Prm autoshade body.prm  polygon
Prm autoshadeinv body.prm polygon
```

Opacity

Version 0.01

Opacity, trans

Setting opacity without changing the shade colors is also possible using “opacity” and “trans”.

syntax

```
prm opacity file.prm 0.8
prm trans file.prm 0.2
```

As of version 0.02 “opacity” and “trans” are the EXACT opposite (used to be the same in version 0.01). for instance 80% opacity is exactly 20% transparency.

ATTENTION: unlike shade, prm use float numbers from 0.000 to 1.0000 where “opacity : 0.00” (transparency: 1.00) means object totally transparent (or invisible), as for the exact opposite “opacity: 1.0” (transparency: 0.00) means the object is fully opaque. It goes without saying that “opacity 0.5” (transparency 0.5) means that the object is 50% transparency, 50% opaque)

Examples:

`Prm opacity body.prm 0.3`	*object is 30% opaque, 70% transparent*
`Prm trans body.prm 0.3`	*object is 30% transparent, 70% opaque*

Environment “reflection”

Version 0.02

Setting EnvMap (reflections) **on/off**

Envmap is that « reflection » effect that can be seen on car or PRM and has a feeling of « moving by moving the perspective »

syntax

```
prm envon file.prm
prm envoff file.prm
```

Examples:

`Prm envon body.prm`	*Set object reflective “envmap” on*
`Prm envoff body.prm`	*remove “envmap” from a PRM file*

Version 0.01 Version 0.02

Geometry manipulation

It was possible to resize a PRM using version 0.01 of PRM tool, and with version 0.02 being out, we have added more features.

Version 0.02

Translate

This is the fastest way to move a PRM file.

syntax

```
prm translate file.prm X Y Z
```

This will allow PRM to be moved by X units to right, Y units to down, Z units to forward.

Examples:

`Prm translate body.prm 0 -100 0`	*move body.prm upwards by 100 units*
`Prm translate body.prm 10 20 30`	*move body.prm 10 to right, 20 down and 30 forward.*

Also, by using those two next commands in row (one after another) you'll get a neutralized effect (car moves 10 up then 10 down, this is like not moving at all)

```
Prm translate body.prm 0 -10 0
Prm translate body.prm 0 10 0
```

Version 0.02

Rotation

This is the fastest way to rotate a PRM file using a specific rotation axis point.

syntax

```
prm rotate file.prm origin Rx Ry Rz
prm rotate file.prm center Rx Ry Rz
prm rotate file.prm centroid Rx Ry Rz
prm rotate file.prm Cx Cy Cz Rx Ry Rz
```

There is no default, the user has to specify which point should be the center of rotation:

- origin: a virtual point defined as position X=Y=Z=0.0
- center: another virtual point defined as the center of Bounding Box (BBOX)
- centroid: Virtual point defined as the Center of Mass of the PRM file.
- A custom point specified by its values Cx, Cy, Cz

After specifying the center of rotation, the value of rotation Rx, Ry, Rz in degrees (0 to 360). As a matter of fact a rotation of 360, -360, 720 etc is same as having rotation of 0 degrees, that's not rotating anything at all. The Values Rx, Ry and Rz respectively represent the degrees of rotations revolving along X-axis (right), Y-axis (down) and Z-axis (forward), that's said as pitch, yaw and roll values.

It's recommended to use either center or centroid as center of rotation and have only one rotation at one time (only one of Rx, Ry or Rz is set while others are 0).

Examples:

`Prm rotate body.prm origin 0 90 0`	*have a yaw of 90° with rotation center (0,0,0)*
`Prm rotate body.prm center 90 0 0`	*have a pitch of 90°around the center of object (or the Boundingbox to be exact)*
`Prm rotate body.prm 10 -10 0 90 0 0`	*Revolve a 90° pitch rotation, using point (10,-10,0) as center of rotation*

Version 0.01

Scale / resize

This is the fastest way to scale a PRM file .

syntax

```
prm scale file.prm 0.33
prm scale file.prm 0.3 0.2 0.5
prm resize file.prm 0.33
prm scale file.prm -nfs4
```

Examples:

`Prm scale  body.prm 0.33`	*scale body.prm on 3 dimensions 33% (33% X; 33% Y and 33% Z)*
`Prm scale body.prm 0.3 2.5 1`	*Scale body.prm 0.3 on sides (X-axis), 2.5x (250%) on height (Y-Axis), and keep depth same (Z-axis)*
`Prm scale body.prm -nfs4`	*scale in all directions 80%*
`Prm scale body.prm -double`	*scale in all directions 200% (double)*
`Prm scale body.prm -half`	*scale in all directions 50% (half)*
`Prm scale body.prm -tiny`	*scale in all directions 1% (tiny)*
`Prm scale body.prm -zmod`	*scale in all directions 333% (about 3 times)*

Also, it's possible to write "prm resize" instead of "prm scale", the results are the same

Version 0.01 Version 0.02

Centering objects

PRM's center is a main command.

syntax

```
prm center file.prm
prm center file.prm XYZ
prm centroid file.prm
```

Center (version 0.01)

Centering an object can be by centering its X (left-right), Y (up-down) or Z (forward-backward) axis.. ***`prm center file.prm XYZ`*** or ***`prm center file.prm`*** will center the objects in 3d dimensions. So in case we want to center the objects up_down then we may write ***`prm center file.prm Y.`*** Again to save time, if we're to center every axis EXCEPT Z axis, we can write ***`prm center file.prm XY.`***

Centeroid or centroid (version 0.02)

"Centroid" uses another way to calculate the center, this time using CoM `prm centroid file.prm`.

Examples:

`prm center file.prm`	*centers file.prm in 3-axis using conventional method*
`Prm center body.prm YX`	*centers body.prm in 2 axis (Y and Y).*
`Prm centroid body.prm`	*centers body.prm around CoM (in 3 axis XYZ)*

Version 0.02

Flips

syntax

```
prm flip file.prm
prm vflip file.prm
prm hflip file.prm
```

While they're all categorized as "flips", each of the 3 commands is totally different from the other.

- Flip: have a mirror flip (to be combined with vflip or hflip)
- Vflip: vertical flip, same as `prm resize file.prm 0 -1 0`
- Hflip: horizantal flip, same as `prm resize file.prm -1 0 0`

After a vertical or horizontal flip, it's recommended to use a doubleside command (`prm dblsd file.prm`)

Example: horizontally mirror one car body (needs two commands, one run after the other)

```
Prm hflip body.prm
Prm flip body.prm
```

Version 0.01

Doublesiding and singlesiding objects

syntax

```
prm dblsd file.prm
prm double file.prm
prm single file.prm
```

In some objects, it happens that one side is "visible" while going to the other side (like back of the plane) will reveal transparent (not visible).

While it's possible to flip the normal, it's also possible to let the two sides visible this is so called "doublesiding", as opposite "singlesiding"

Center (version 0.01)

Centering an object can be by centering its X (left-right), Y (up-down) or Z (forward-backward) axis.. ***`prm center file.prm XYZ`*** or ***`prm center file.prm`*** will center the objects in 3d dimensions. So in case we want to center the objects up_down then we may write ***`prm center file.prm Y.`*** Again to save time, if we're to center every axis EXCEPT Z axis, we can write ***`prm center file.prm XY.`***

Centeroid or centroid (version 0.02)

"Centroid" uses another way to calculate the center, this time using CoM `prm centroid file.prm`.

Examples:

```
prm double file.prm
prm doubleside file.prm
prm dblsd file.prm
Prm single  body.prm
```

3 commands, all the same: making the object double sided

one command: single siding an object (makes only one side visible)

UV map and texture manipulation

Version 0.02

It was possible to resize a PRM using version 0.01 of PRM tool, and with version 0.02 being out, we have added more features.

Version 0.02

Texmap/rvtexmap (Remap UV)

PRM Texmap is made to be compatible with RVTexMap thus it borrows the syntax from it. (remapping UV)

And for compatiblity the bitmap is supposed to be 256x256 forever, so if you work with Paint, Paint.NET, GIMP or Photoshop etc. make sure that the coordinates are taken from a 256x256 UV map bitmap copy out of the original (512x512 for instance) bitmap.

Consider the following instruction `prm texmap body.prm h:0,64:256x128=a:0,0:128x128`

While `256x128` stands for *Width×Height*, the `0,64` stands for Coordinates (padding from left, top corner) `h:0,64:256x128` means that UV map from (0,64) to (256,192) will be selected if the polygon is textured on channel H (nhood1h.bmp for instance) then the other part of the "equation" `a:0,0:128x128` it means that everything inside the selected zone will be all transformed to be fit in the new coordinates in A channel from (0,0) to (128,128).

syntax

```
prm texmap body.prm h:0,0:256x256=a:0,0:128x128
prm texmap body.prm *:0,0:256x128=*:0,0:256x256
```

If you're unsure about the channel (for instance car texture), just use * as channel in both of

The exact syntax is : Source_uv=dest_uv where both of them is composed from CHANNEL:POSITION:DIMENSIONS

N.B: Spaces will be ignored so you may want to use as much space characters as you want (which is not recommended anyway)

Examples:

```
Prm texmap body.prm *:0,0:256x256=*:0,0:128x128
```

resize the UV from full-size to quarter-size (location: 0)

Version 0.02

Uvremap (Remap UV)

UV Remap is the original syntax for PRM Toolkit. It's meant to remap the UV. While the execution of UVREMAP and TEXMAP are handled by the same function, the syntax is <u>totally</u> different

syntax

```
prm uvremap body.prm "H 0.0x0.0 1.0x1.0" "A 0.0x0.0 0.5x0.5"
prm uvremap body.prm "* 0.0x0.0 1.0x0.5" "* 0.0x0.0 1.0x1.0"
```

Consider the instruction `prm uvremap body.prm "H 0.0x0.25 1.0x0.75" "A 0.0x0.0 0.5x0.5"`
The first `"H 0.0x0.25 1.0x0.75"`, selects all the UV in H channel, starting from UV (0.00, 0.25 to 1.00,0.75) and transform them to A-channel (0.00, 0.00 to 0.5, 0.5) This would mean, for a 256x256 bitmap, it's like transforming H channel's 0,64 to 256,192
to A Channel's 0,0 to 128,128 which would be then compatible with `prm texmap body.prm h:0,64:256x128=a:0,0:128x128.`

In other words, the first 0.00x0.25 stands for the first corner of the rectangle while 1.0x0.75 stands for the second corner of the rectangle.

For calculating the UV, you have to do small calcs, for instance. X=192 Y=128 in a 256x256 bitmap is equivalent to UV 192/256 x 128/256 or again UV=0.75x0.5.

If you're unsure about the channel (for instance car texture), just use * as channel in both of

N.B: Spaces are important, make sure to stick to the exact syntax.

Examples:

```
Prm uvremap body.prm “* 0x0 1.0x1.0” “* 0x0 0.5x0.5”
```

resize the UV from full-size to quarter-size (location: 0)

Version 0.01

Remap channel only

It's possible to remap a channel (like transporting everything mapped to A channel to H channel etc.).

syntax

```
prm remap body.prm A B
```

Examples:

`Prm remap body.prm A B` *remap all the "UV" from A to B (like trackA.bmp to trackB.bmp)*

Version 0.02

UV Texture generation

TMVfR and **UV Viewer for Re-Volt** both could easily be competitors for prm toolkit, the PRM toolkit offers the best best quality in the easiest command with no prior configuration.

syntax

```
prm texgen body.prm
prm texgenwire body.prm
prm texgenmix body.prm
```

After the command is executed 8 bitmaps will generated named file.prm_8192.png (8192x8192), ..., file.prm_512.png (512x512),..., file.prm_64.png (64x64).

There are 3 possible commands: `texgen` and `texgenwire` (`texgenwire` to mix both of them)

- Texgen: generates filled polygons UV

- Texgenwire: generates firewire (borders) polygons UV

- Texgenmix: both of them

Normal manipulation

Version 0.02

A Normal Vector is a vector that describes the forward face (ok… you may look into Wikipedia to get more acquited)

Normals are used for "Env" effect (the shiny reflection thing). If you're still not sure about them feel free to skip reading this section and go ask about them in Re-Volt Live or Our Re-Volt Pub .

Multiply Normal

This will multiply normals with X, Y and Z values

Examples:

```
Prm normal body.prm 2 1 1
```

syntax

```
prm normal file.prm X Y Z
```

Force Normal

Force one normal value on the whole prm file. This will cause the object to be seen as one unified object.

Examples:

```
Prm setnormal body.prm 0.5 0.25 0.25
```

syntax

```
prm setnormal file.prm X Y Z
```

Break normals (reassign them)

This can be described as reassigning normals either using vertex or polygons.
While **breaknormals** uses vertices' position to retreive normal back, **normalize** retreive the values as seen from polygon normals. Test

```
Prm normalize body.prm
Prm breaknormals body.prm
```

syntax

```
prm normalize file.prm
prm breaknormals file.prm
```

Misc

Getting Polygon count

syntax

```
prm count file.prm
prm info file.prm
```

Restore the previous position

PRM tends to back the PRM file before proceeding to the action, this is a reversible effect and using “Restore” it’s possible to undo the last action done in PRM toolcase

syntax

```
prm restore file.prm
```