Image Mapping

Matiu Carr*

Nov. 14, 1993

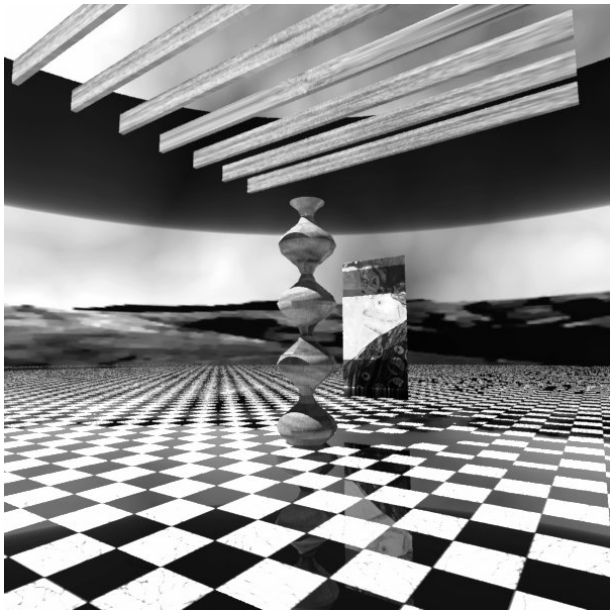

Figure 1: Image mapping

1 Introduction

The mapping of bitmaps (two dimensional arrays of colour values) to surfaces in the Radiance application involves the use of the **colorpict** pattern type.

As described in the manual it takes the form

modifier **colorpict** *pattern_ name*
7[+] *redfunc greenfunc bluefunc imagename funcfile ufunc vfunc* [trans.]
0
A1 A2 A3 ...An

*PDF by Axel Jacobs

Colorpict is a pattern: it is used as a modifier for a material, which can then be applied to a surface. By itself it will produce no visual results.

Colorpict brings together a set of functions, stored in the file *funcfile*, and applies these to the bitmap *imagename*.

Funcfile is a text file. Picture.cal and cyl_wrap.cal are examples of files that contain function definitions for use with **colorpict**, they are appended to the example Radiance script below. The manual describes these function files as auxilliary files.

The functions contained within the function file transform the colour of the image (*redfunc, greenfunc* and *bluefunc*: function files that can modify the red, green and blue components of the pixels in an image), and its geometry (*ufunc* and *vfunc*).

Ufunc and *vfunc* provide the means by which Radiance indexes into a bitmap during the rendering process.

2 Example

```
# a blue sky with white clouds
void colorfunc skycolour1
4 cloud cloud one cloudy.cal
0
0

# use the colour maps to create glow sources (no shadow casting)
skycolour1 glow skyglow1
0
0
4 1 1 1 0

# create the sky
skyglow1 source sky
0
0
4 0 0 1 360

# the other stuff:

# a light source
void illum sun_light
1 skyglow1
0
3 1.2 1.2 1.2

sun_light source sun
0
0
4 1 1 2 180
```

```
# a plane
# an object

# materials:

# a general shiny (glossy) material
void plastic shiny
0
0
5 1 1 1 .1 0

# a tile picture mapped onto a horizontal plane
# this serves as an example of a tiled bitmap
shiny colorpict floor
9 red green blue marble_floor.pic picture.cal tile_u tile_v -s 2
0
1 1
```

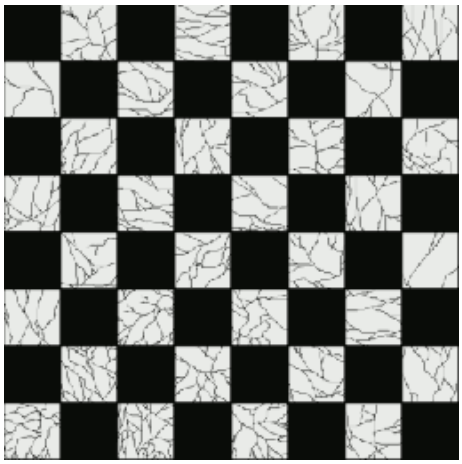

Figure 2: Marble floor

```
# a general white material
void plastic white
0
0
5 1 1 1 0 0

# a wood picture wrapped around a cylinder
# this serves as an example of a bitmap mapped cylindrically to a surface
white colorpict wood
9 red green blue wood.pic cyl_wrap.cal u v -s 3.5
0
2 590 290
```

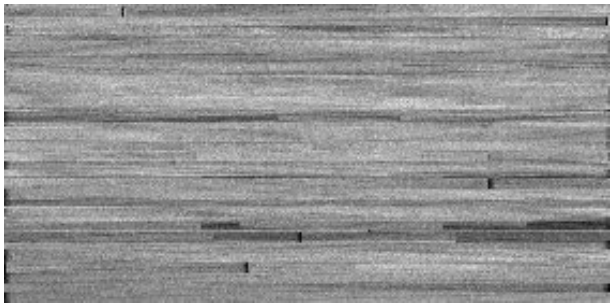

Figure 3: Wood

```
# a scanned image mapped onto a wall behind the wooden wobbly thing
# this is an example of specifically positioning a bitmap in space
white colorpict fresco
17 red green blue klimt.pic picture.cal pic_u pic_v -s .75 \
-t -.375 .35 0 -rz 180 -rx -90
0
1 2
```

Figure 4: Gustav Klimt: Danae

```
# an image to provide a backdrop cylindrically mapped to a large tube
void colorpict hillspic
13 red green blue hills.pic cyl_wrap.cal u v -s 20 -t 0 0 -10
0
2 773 200
```

Figure 5: Panorama of hills

```
hillspic dielectric hills
0
0
5 1 1 1 1 0

# the geometry
floor ring ground_plane
0
0
8 0 0 0
0 0 1
0 30

!genbox wood lintel 4 .05 .2 |xform -t -2 -1.525 3
!genbox wood lintel 4 .05 .2 |xform -t -2 -1.025 3
!genbox wood lintel 4 .05 .2 |xform -t -2 -0.525 3
!genbox wood lintel 4 .05 .2 |xform -t -2 -.025 3
!genbox wood lintel 4 .05 .2 |xform -t -2 .475 3
!genbox wood lintel 4 .05 .2 |xform -t -2 .975 3
!genbox wood lintel 4 .05 .2 |xform -t -2 1.475 3

!genrev wood wobble '2*t' '.15+.1*sin(t*PI*8)' 32 -s

fresco polygon picture
0
0
12 -.375 -1.299 .25
.375 -1.299 .25
.375 -1.299 1.75
-.375 -1.299 1.75

hills tube distant_view
0
0
7 0 0 0
0 0 20
30
```

3 picture.cal

Picture.cal is a standard set of functions provided as part of the Radiance distribution.

Two useful sets of function definitions are the pic_u,pic_v and tile_u,tile_v pairs.

The pic_u,pic_v pair map the image into space such that it lies parallel to the XY plane, with the bottom left hand corner of the image sitting on the Z axis (ie. x=0,y=0).

The smallest dimension of the image (in the case that the image is not square) will now have a length of 1 unit in Radiance space, while the larger will have a length equal to the ratio of the longest dimension to the shortest.

It is then possible to scale and move this image through space using the transformation tools available in Radiance.

The positioning of the Klimt image (*Danae*) is an example of this procedure. Imagine the image placed at the origin. It is twice as tall as it is wide. This means that in Radiance space it is 1 unit long and 2 units tall. The first transformation performed is -s .75. This scales the image down 75% about the origin. The next transformation moves the image so that the Y axis passes through its centre. It is then rotaed 180° about the Z axis followed by a 90° rotation about the X axis to bring it to an upright position, with its base 0.35 units above the origin.

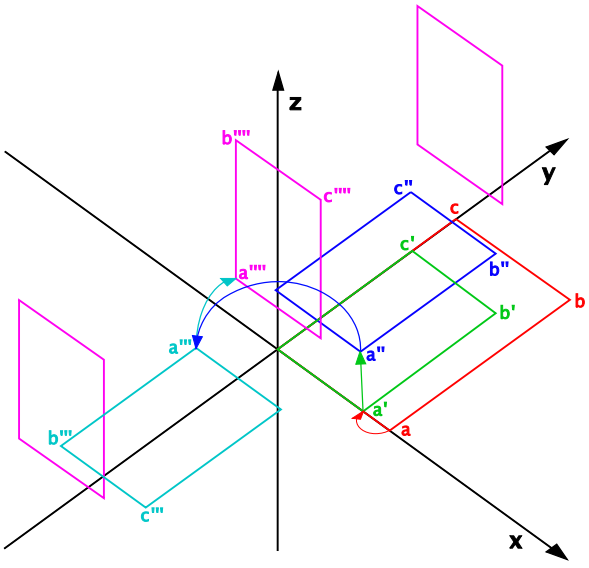

Figure 6: Transforming the Klimt image

Having positioned the image here, imagine a tube extending perpendicular to the image in both directions with a rectangular section the same as the profile of the image. This defines the space within which the image will map on to surfaces: ie. any surface that has this pattern assigned to it will have the image mapped onto it where it cuts this tube.

Where a surface falls outside of this image space it is coloured black, sometimes punctuated with flecks of colour from the edges of the image.

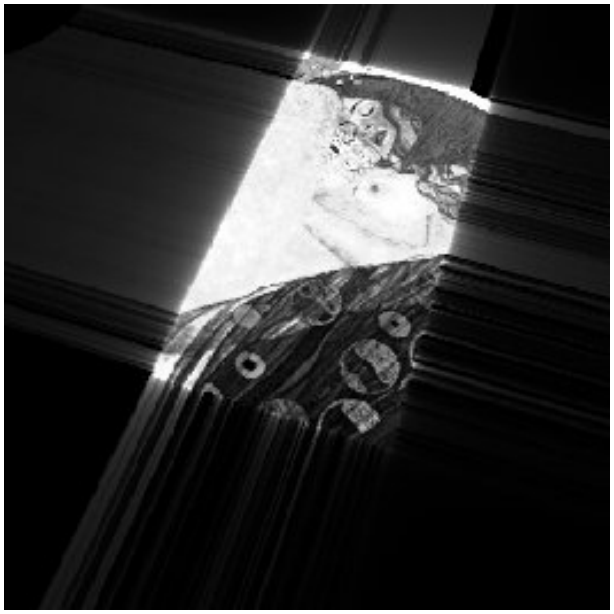

Figure 7: Mapped image

The tile_u, tile_v pair is similar to the pic_u, pic_v pair except that the image is tiled to infinity in the plane of which it forms a subregion.

The image is placed at the origin as with the pic_u, pic_v pair.

The tile_u, tile_v pair requires a parameter. This is a real number equal to the ratio of the longest side length to the shortest (in the diagram above this would be /). This ratio (once again as it was with the pic_u, pic_v pair) determines the size of the image (in Radiance units), as it is mapped into the space of a scene.

The checkered floor pattern in the example script places a square image of a chessboard pattern, doubles its linear dimensions (so that each checker is 0.25 Radiance units long) and applies this image map to a predefined shiny material, all to create the effect of a polished marble floor.

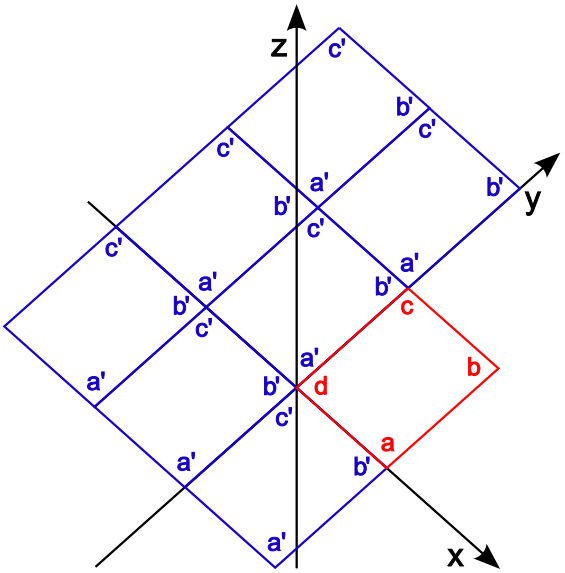

Figure 8: Tiling an image

```
{
picture.cal

Calculation of 2d picture coordinates.
Picture is projected onto xy plane with lower left corner at origin.

A1 Ratio of height to width for tiles.
A2 Average red value for fadered or grey for fadegrey
A3 Average green value for fadegreen
A4 Average blue value for fadeblue
}

pic_u = Px;
pic_v = Py;

tile_u = mod(pic_u,max(1,1/pic_aspect));
tile_v = mod(pic_v,max(1,pic_aspect));

match_u = tri(pic_u,max(1,1/pic_aspect));
match_v = tri(pic_v,max(1,pic_aspect));

stag_u = if(pic_aspect - 1,
frac(if(frac(pic_v/pic_aspect/2) - .5,pic_u,pic_u+.5)),
mod(if(frac(pic_v/2) - .5,pic_u,pic_u+.5/pic_aspect),
1/pic_aspect));
stag_v = tile_v;

pic_aspect = if(arg(0) - .5, arg(1), 1);

fadered(r,g,b) = fade(r, A2, T*.1);
fadegreen(r,g,b) = fade(g, A3, T*.1);
fadeblue(r,g,b) = fade(b, A4, T*.1);
fadegrey(r,g,b) = fade(grey(r,g,b), A2, T*.1);
```

4 cyl_wrap.cal

Cyl_wrap.cal is included here as an example of an image mapping auxiliary file not included as part of the standard Radiance distribution.

Conceptually this mapping procedure takes an image and rolls it around the z axis such that it will be "the right way around" when looked at from the outside.

As mentioned above, the *ufunc*, *vfunc* functions determine how an image is indexed according to particular conditions during rendering.

The diagram below shows the outline of an image, defined by vertices abcd. The x (or horizontal) dimension of the image is given by . The y (or vertical) dimension is given by .

Lets call the ratio of the larger dimension to the smaller of this image *rect*.

So, for this particular image: *rect* = /

Image mapping in Radiance, treats the image as a rectangular field of colour values indexed by coordinates u and v, where u serves as the horizontal index, and v as the vertical.

If the horizontal dimension of the image is the smaller then u extends from 0 to 1 and v extends from 0 to *rect*. If the vertical dimension is the smaller, this is reversed.

Ufunc and *vfunc* must produce values within these ranges to yield a "sensible" result.

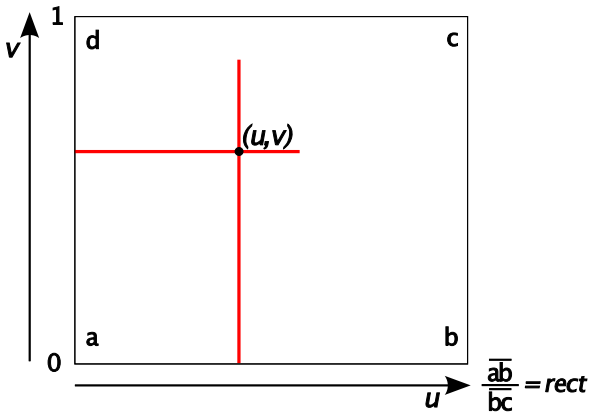

Figure 9: uv mapping

Cyl_wrap.cal uses the point where a ray hits the surface with the image mapped onto it (given by Px,Py,Pz), to calculate the values for u and v. v returns a value equal to Pz.

u, on the other hand, has a more complex derivation. Imagine a line drawn from the origin of a plane to the point Px,Py.

Let us call the smallest angle formed between this line and the ray starting at the origin and running along the positive x axis *theta*.

Now let us give *theta* a sign according to whether the line from the origin to Px,Py is above or below the x axis: positive for above, and negative for below.

u is given by multiplying the ratio of *theta* to the angle of a full revolution (2π) by the maximum value that u can have (1 or *rect* -as defined above).

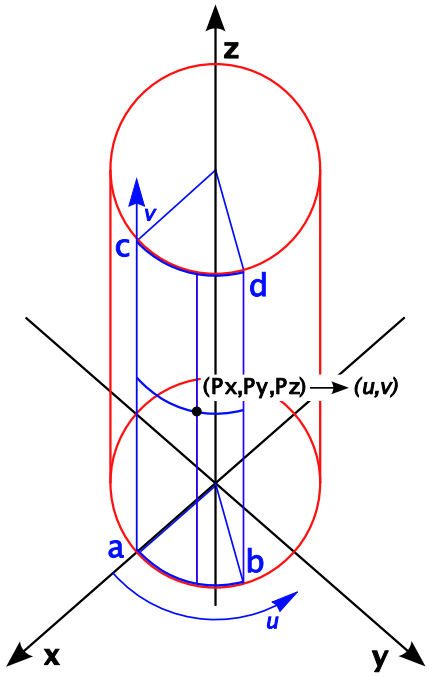

Figure 10: Cylindrical mapping

By scaling and translating the original mapping space, it is possible to position the cylindrical map anywhere it is needed, and, as with the functions in picture.cal, the mapping projects through space.

Cyl_wrap.cal is used in the example to map a wood pattern onto the tall vase in the centre of the composition, and this mapping is used again for the rafters hovering above the assemblage.

A small picture of hills around Helensville is wrapped around the inside of a large tube enclosing the model, giving the effect of a

{
cyl_wrap.cal
by Matiu Carr

2d coordinate mapping onto a cylinder

A1 = x dimension of picture
A2 = y dimension of picture
}

pPx = if(Px-FTINY,Px,if(Px+FTINY,FTINY,Px));

theta = atan2(Py,pPx);

u = theta* if(A1-A2,A1/A2,1)/(2*PI);
v = Pz;