# NURBS Textures

Peter Salvi

June 30, 2008

## 1 Introduction

A NURBS surface can be visualized very easily using the GLU library. However we often also want to put some texture on it, which is not so straightforward, although not at all difficult. Section 3 will show how to do this. A more interesting problem is creating dynamic textures, ie. textures that move as the view changes. Section 4 and 5 show such maps, using the eye position and viewing direction as a reference. (These techniques are not restricted to NURBS surfaces, and can be applied to different objects, too.)

All code snippets and computations in this file are based on the program *SFView* I've written a few days ago.

## 2 Displaying NURBS Surfaces

Before we talk about textures, let's review how NURBS surfaces are displayed. First a `GLUnurbs` object has to be created, and we can also set a few options:

```
GLUnurbs *globj = gluNewNurbsRenderer();
gluNurbsProperty(globj, GLU_CULLING, GL_TRUE);
gluNurbsProperty(globj, GLU_SAMPLING_METHOD,
  GLU_PARAMETRIC_ERROR);
gluNurbsProperty(globj, GLU_PARAMETRIC_TOLERANCE, 0.05);
```

These mean that we don't want to display the parts of the surface whose control points are not visible on the screen and that we want a fairly accurate triangulation. For a much faster (but less precise) triangulation, we may set

```
gluNurbsProperty(globj, GLU_SAMPLING_METHOD, GLU_PATH_LENGTH);
gluNurbsProperty(globj, GLU_SAMPLING_TOLERANCE, 50.0);
```

which is, by the way, the default.

Once we have set these in some initialization routine, we can display the surface

$$S(u,v) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} P_{ij} N_i^{U,p}(u) N_j^{V,q}(v)$$

with the commands

```
gluBeginSurface(globj);
gluNurbsSurface(globj, n+p+1, U, m+q+1, V, 3m, 3,
  cpts, p+1, q+1, GL_MAP2_VERTEX_3);
gluEndSurface(globj);
```

where cpts $= [P^x_{0,0},\ P^y_{0,0},\ P^z_{0,0},\ P^x_{0,1},\ \ldots,\ P^z_{0,m-1},\ P^x_{1,0},\ \ldots,\ P^z_{n-1,m-1}]$. For correct lighting effects, make sure that the following OpenGL flags are set: `GL_LIGHTING`, `GL_AUTO_NORMAL` and `GL_NORMALIZE`. Now that we have a working NURBS Surface, we can move on to textures.

# 3 Standard textures

Textures in OpenGL are bound to integers. Let's see first an example on how to set up a 2-dimensional texture:

```
glGenTextures(1, &texture_id);
glBindTexture(GL_TEXTURE_2D, texture_id);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
  GL_LINEAR);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
  GL_LINEAR);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
  GL_CLAMP_TO_EDGE);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
  GL_CLAMP_TO_EDGE);
unsigned char *data = new data[width * height * 3];
// Put texture in data...
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
  GL_RGB, GL_UNSIGNED_BYTE, data);
delete[] data;
```

This will create a texture with the given parameters. Now we use clamping at the edges, but Section 5 will show an example where we want to repeat. Also, the min and mag filter were both linear in the example, but you may use mipmaps, like this:

Figure 1: Surface with isophote map.

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
  GL_LINEAR_MIPMAP_NEAREST);
// ...
gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, width, height,
  GL_RGB, GL_UNSIGNED_BYTE, data);
```

For a more detailed explanation, see the manual pages.

When we want to actually use the textures, we have to do three things: set the texture active with a call to `glBindTexture`, enable texture mapping with `glEnable(GL_TEXTURE_2D)` and set the texture coordinates. The latter is done for NUBRS *inside* the `gluBeginSurface` / `gluEndSurface` region:

```
gluBeginSurface(globj);
gluNurbsSurface(globj, 4, Û, 4, V̂, 4, 2, texcpts, 2, 2,
  GL_MAP2_TEXTURE_COORD_2);
gluNurbsSurface(globj, ..., GL_MAP2_VERTEX_3);
gluEndSurface(globj);
```

where $\hat{U} = [U_p,\ U_p,\ U_n,\ U_n]$, $\hat{V} = [V_q,\ V_q,\ V_m,\ V_m]$ and

$$\text{texcpts} = [0,\ 0,\ 0,\ 1,\ 1,\ 0,\ 1,\ 1],$$

that is a mapping from the original domain to $[0,\ 1]$.

# 4 Isophote map

Isophotes are points on a surface whose normals have the same angle with a vector pointing to a reference point, i.e. $c = \underline{n}\frac{p_{\text{ref}}-p}{\|p_{\text{ref}}-p\|}$, where $\underline{n}$ is the normal vector at $p$. Instead of displaying just one isophote on the surface, we can create stripes that correspond to the change in the angle ($\cos^{-1} c$). For example, we can change the color every 5 degrees, resulting in a surface like the one on Fig. 1. The reference point is usually the eye position, thus this will be a dynamic texture.

The trick to do this is texture coordinate generation. There are three generation methods, and for this map we will use the sphere map. The generation can be enabled by these commands:

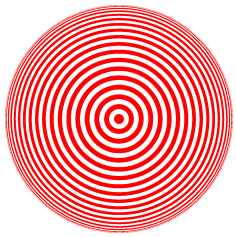

Figure 2: Isophote map image (1024x1024 pixels).

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
```

This sets sphere map texture generation for both texture coordinates. Of course, using texture generation means that we do not have to call `gluNurbsSurface` with `GL_MAP2_TEXTURE_COORD_2`.

Now I will describe how to compute the texture map, but as an alternative, you can just extract the image from this PDF file (see Fig. 2). Sphere mapping gives the 3-dimensional (unit) reflection vector $(x, y, z)$ in 2-dimensional form: $(\frac{x}{m}+\frac{1}{2}, \frac{y}{m}+\frac{1}{2})$, where $m = 2\sqrt{x^2+y^2+(1+z)^2}$. We want to get the reflection angle in the line of sight, whose (doubled) cosine can be approximated by the $z$ coordinate of the reflection vector, since the reflection vector is given in eye coordinates.

Using the equation $x^2+y^2+z^2=1$, it turns out that

$$z = 8\left(\frac{1}{4} - \left(\frac{x}{m}\right)^2 - \left(\frac{y}{m}\right)^2\right) - 1,$$

from which $\alpha = \cos^{-1} z$. This way we can compute the map in the $[0, 1]\times[0, 1]$ domain.

# 5 Slicing map

Slicing maps also show stripes, but in a more straightforward manner: the colors change according to the distance from some reference plane. For this we can use the `GL_OBJECT_LINEAR` map, that gives the distance from a given plane.

The texture itself is very simple: it is one-dimensional (so use `GL_TEXTURE_1D` in place of `GL_TEXTURE_2D`), and the first half of the values are of the first color, the second are of the other color. We want this to repeat, so we set

```
glTexParameter(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_REPEAT);
```

The map creation also needs a one-dimensional function: `glTexImage1D`.

The commands to display the texture are the same as in Section 4, but with only one parameter (`GL_S`). The plane can be set with

```
glTexGenfv(GL_S, GL_OBJECT_PLANE, plane);
```

where plane = [A, B, C, D] such that the plane's equation is $Ax+By+Cz = 0$. Note that $A$, $B$ and $C$ can be scaled — we can use this to set the density of the map. Dividing with $\frac{1}{50}$ of the axis of the scene's bounding box seems to be a reasonable choice.