

# PN Patches

Gerald Farin

February 16, 2003

A recent use of triangular Bézier patches is in computer graphics, where they are used for rendering triangle meshes, see [3], [2], [1]. Suppose you have a relatively coarse triangle mesh, such that even with Gouraud or Phong shading you will not get a nice looking object. Suppose further that you have a normal defined at every vertex.

The goal is to build a cubic Bézier patch over each triangle, such that the resulting collection of Bézier triangles forms a continuous surfaces. This surface will not be  $G^1$ , however. This defect will be overcome by a display trick, explained below.

Let us concentrate on one triangular facet. It has three vertices, which we denote by  $\mathbf{b}_{300}$ ,  $\mathbf{b}_{030}$ ,  $\mathbf{b}_{003}$ . At each of these vertices, we also have a normal, and we denote them by  $\mathbf{n}_{200}$ ,  $\mathbf{n}_{020}$ ,  $\mathbf{n}_{002}$  – this notation will be explained shortly. Our desired Bézier patch needs three boundary curves; let's concentrate on the one from  $\mathbf{b}_{300}$  to  $\mathbf{b}_{030}$ . We need Bézier points  $\mathbf{b}_{210}$  and  $\mathbf{b}_{120}$  in order to have a boundary control polygon. They are found by using the point/normal interpolation algorithm from Section 7.7. This process is illustrated in Figure 1.

We repeat for the remaining two boundaries. One control point is still undetermined, namely  $\mathbf{b}_{111}$ . We determine it from (18.4).

At this point, we have built a cubic patch over each triangular facet. The resulting

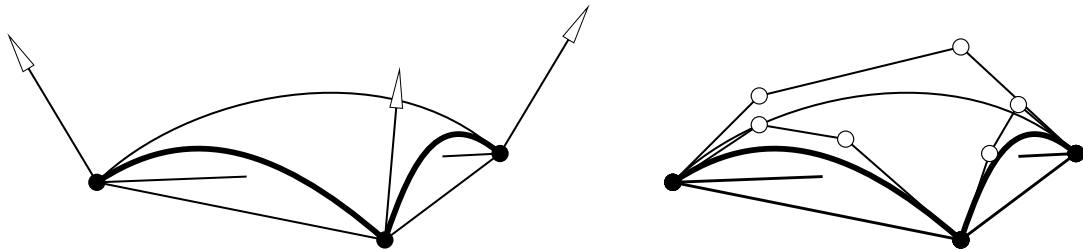


Figure 1: PN patches: left, input data. Right: boundary control polygons.

overall surface will not be smooth, for the reasons outlined in Section 18.4. The trick now is to modify the display algorithm such that it *looks* smooth.

When we render a surface, we need to break it down into an appropriate number of triangular facets, compute the normals at the vertices, and feed all to the shader. For a Bézier triangle, this is done by 1. running the de Casteljau algorithm for each vertex and 2. extracting the surface normal from it. For PN patches, these two processes are separated. The point evaluation is as usual, but the normal is computed differently. In fact, it is not the true surface normal but rather one that “looks better”.

The (non-normalized) normal  $\mathbf{n}(\mathbf{u})$  of a cubic triangular Bézier patch is a degree four polynomial of  $\mathbf{u}$ . For PN patches, we replace this degree four polynomial by a degree two one: a *display normal*  $\mathbf{d}(\mathbf{u})$  will be computed from

$$\mathbf{d}(\mathbf{u}) = \sum_{|\mathbf{i}|=n=2} B_{\mathbf{i}}^2(\mathbf{u})\mathbf{d}_{\mathbf{i}}. \quad (1)$$

The six display control normals  $\mathbf{d}_{\mathbf{i}}$  are found as follows:

$$\mathbf{d}_{200} = \mathbf{n}_{200}, \quad \mathbf{d}_{020} = \mathbf{n}_{020}, \quad \mathbf{d}_{002} = \mathbf{n}_{002}.$$

The three remaining ones are not that simple. Concentrating on the edge  $u_3 = 0$ , and the display normal  $\mathbf{d}_{1/2}$  at  $(\frac{1}{2}, \frac{1}{2}, 0)$ , we might set

$$\mathbf{d}_{1/2} = \frac{1}{2}\mathbf{d}_{200} + \frac{1}{2}\mathbf{d}_{020}$$

and normalize to get a unit normal. This choice, however, would give an undesirable normal if  $\mathbf{d}_{200}$  and  $\mathbf{d}_{020}$  were close to being equal, meaning that the corresponding boundary curve has an inflection point. See Figure 2

A better choice was found to be constructing a plane perpendicular to  $\overline{\mathbf{b}_{300}, \mathbf{b}_{030}}$  and reflecting the above  $\mathbf{d}_{1/2}$  across this plane for the final display normal value of  $\mathbf{d}_{1/2}$ . This gives

$$\mathbf{d}_{1/2} = \mathbf{d}_{200} + \mathbf{d}_{020} - 2 \frac{(\mathbf{b}_{030} - \mathbf{b}_{300})(\mathbf{d}_{200} + \mathbf{d}_{020})}{\|\mathbf{b}_{030} - \mathbf{b}_{300}\|^2} (\mathbf{b}_{030} - \mathbf{b}_{300}). \quad (2)$$

The display normal  $\mathbf{d}_{110}$  is then defined to equal  $\mathbf{d}_{1/2}$ .

With this arrangements of display control normals, we are guaranteed continuous normals along common boundary curves between patches. While the resulting display is not accurate, it is smooth – which is what matters if good looks for rendering is important!

An improvement over (2) is as follows. The assignment

$$\mathbf{d}_{110} = \mathbf{d}_{1/2}$$

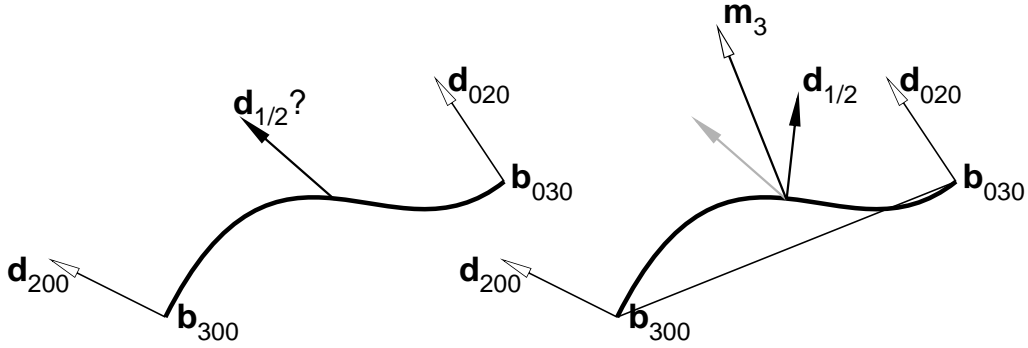


Figure 2: PN normals: left, simple averaging. Right, averaging followed by reflection.

does not guarantee that  $\mathbf{d}(\frac{1}{2}, \frac{1}{2}, 0) = \mathbf{d}_{1/2}$ . In order to achieve this, we need to set

$$\mathbf{d}_{110} = \frac{3}{2}\mathbf{d}_{1/2} - \frac{1}{4}\mathbf{d}_{200} - \frac{1}{4}\mathbf{d}_{020}. \quad (3)$$

## References

- [1] Y-C Lee and C-W Jen. Improved quadratic normal vector interpolation for realistic shading. *The Visual Computer*, 17(6):337–352, 2001.
- [2] C. van Overveld and B. Wyvill. An algorithm for polygon subdivision based on vertex normals. In *Computer Graphics International '97*, pages 3–12, 1997.
- [3] A. Vlachos, J. Peters, C. Boyd, and J. Mitchell. Curved PN triangles. In *ACM Symposium on Interactive 3D Graphics 2001*, pages 159–166, 2001.