4.2

Fast Heightfield Normal Calculation

Jason Shankel, Maxis

shankel@pobox.com

Heightfields are two-dimensional arrays of height values, commonly used to store terrain or water surface data, and are also commonly used for calculating bump maps. This gem will describe how we can take advantage of the special characteristics of heightfield meshes to significantly optimize vertex normal calculation.

Normals on an Arbitrary Mesh

Each vertex in a heightfield mesh requires at least one corresponding surface normal if lighting- and/or environment-mapping calculations are needed for the final rendering result. There are two kinds of normals typically associated with a 3D mesh—face normals and vertex normals. Face normals are, as the name implies, normals associated with each face in a mesh. Vertex normals are normals associated with each vertex.

Face Normals

Calculating face normals is relatively straightforward. Pick two edges of the face that share a common vertex and define two vectors (v_1 and v_2) pointing along the edges with their origin at the shared vertex. The face normal (n_f) is a unit vector pointing in the direction of the cross-product, n_f, of v_1 and v_2. Note that any two edges will do for triangles, as shown in Figure 4.2.1.

$$n_f = (v_1 \times v_2) / | v_1 \times v_2 | \tag{4.2.1}$$

Vertex Normals

Vertex normals are a little less straightforward. While there is only one correct normal for a given face, a given vertex might have multiple normals, each associated with a particular face or group of faces. However, for meshes that are fairly smooth (which is typical for heightfields), we can find a reasonably unique vertex normal by averaging the normals of each face that touches the vertex. This average should be weighted by the relative angle of each face at the vertex to prevent thin or highly tesselated faces from skewing the result.

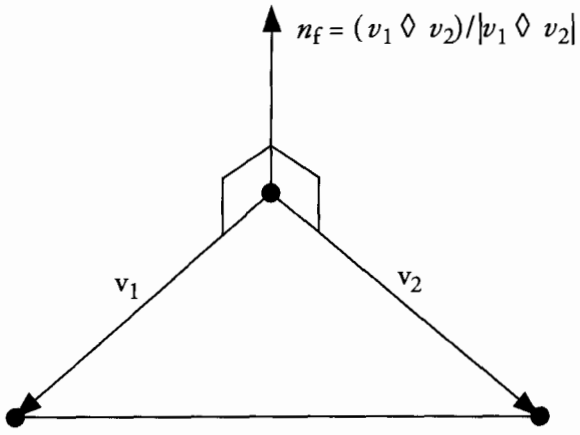

FIGURE 4.2.1 *Calculating a face normal for a triangle.*

Let $\{n_1, n_2, n_3 \ldots n_n\}$ be the normals of the faces touching vertex v, and let $\{a_1, a_2, a_3 \ldots a_n\}$ be the angles between the edges of faces 1 through n.

The normal at vertex v is given by Equation 4.2.2, as shown in Figure 4.2.2.

$$n_v = \frac{\sum_{i=0}^{n} n_i a_i}{\sum_{i=0}^{n} a_i} \tag{4.2.2}$$

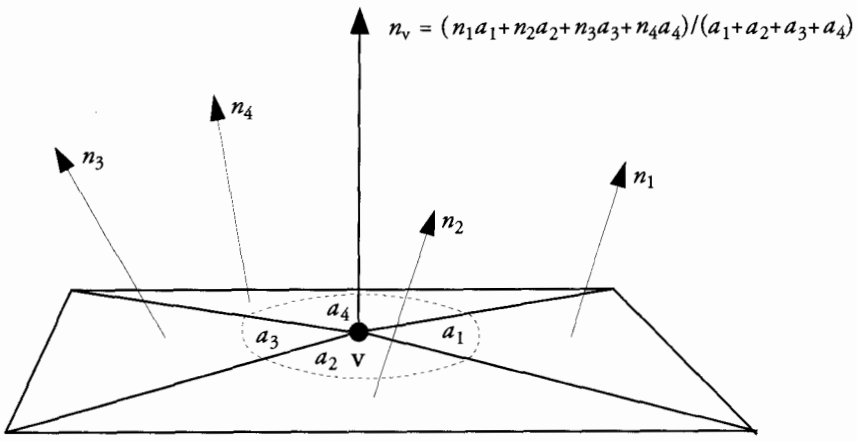

FIGURE 4.2.2 *Calculating a vertex normal using face normals.*

Heightfield Normals

The vertices of a heightfield mesh can be defined as follows by $v_{x,y}$ where

$$v_{x,y} = \{x, y, h(x, y)\}. \tag{4.2.3}$$

In Equation 4.2.3, x and y are regularly-spaced indices into the heightfield grid, and $h(x,y)$ is the height at x,y. For a given vertex v in the heightfield, we can arrange the neighboring vertices as shown in Figure 4.2.3. For simplicity, $h_{1...4}$ refer to the values of $h()$ for the four neighbors, and the four vectors which have their origins at v and point to the neighboring vertices are labeled $v_{1...4}$.

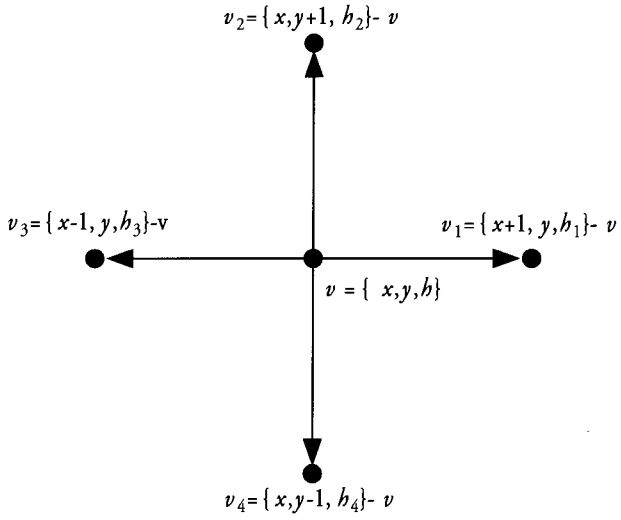

FIGURE 4.2.3 *Heightfield vertex neighbors.*

Simplifying with Assumptions

Given the unique characteristics of heightfield vertices, we can make a number of assumptions that simplify the general vertex normal formula.

First, we can assume that each vertex in the heightfield belongs to exactly four faces. Note that, strictly speaking, this might not be the case. We are assuming that the mesh is composed of quads, when it will most likely be composed of triangles. However, this assumption should not adversely affect the quality of our normals.

Second, we can assume that each of a vertex's faces contributes equally to the vertex normal, thus eliminating the need for performing a weighted average. Again, this assumption is not completely safe. If two neighboring height values greatly diverge, then the angle of their corresponding face will significantly diverge from 90°, thus

changing the 'correct' contribution of the face. However, in cases where neighboring height values diverge significantly, the local vertices no longer have unique normals; and any formula, even the 'correct' one, will produce suspect results.

Finally, since the x and y values are regularly spaced, when we set the origin at v, the neighboring vertices will all have x and y values of 1, –1, or 0. These constants will greatly simplify the cross-product formula.

Doing The Math

Let's start with the original formula:

$$n_v = \frac{\sum_{i=0}^{n} n_i a_i}{\sum_{i=0}^{n} a_i} \tag{4.2.4}$$

Next, since there are only four faces and each contributes equally to the normal, we can simplify the average:

$$n_v=(n_1 + n_2 + n_3 + n_4)/4 \tag{4.2.5}$$

Now, when we go to calculate $n_{1...4}$, we will find that the cross-products reduce to simple terms:

$$\begin{aligned} n_1 &= v_1 \times v_2 = \{-h_1, -h_2, 1\} \\ n_2 &= v_2 \times v_3 = \{h_3, -h_2, 1\} \\ n_3 &= v_3 \times v_4 = \{h_3, h_4, 1\} \\ n_4 &= v_4 \times v_1 = \{-h_1, h_4, 1\} \end{aligned} \tag{4.2.6}$$

Ideally, we would like $n_{1...4}$ to have equal magnitudes, since variation in their magnitudes will affect the averaging formula. However, since the magnitudes of $n_{1...4}$ only significantly vary in cases where the neighboring height values diverge, we can get away with using $n_{1...4}$ as they are.

Adding up, we get:

$$n_v = (n_1 + n_2 + n_3 + n_4) / 4 = \{2(h_3 - h_1), 2(h_4 - h_2), 4\} / 4 \tag{4.2.7}$$

Since the magnitude of n_v is not important at this stage, we can multiply the whole thing by two just to simplify the arithmetic:

$$n_v = \{(h_3 - h_1), (h_4 - h_2), 2\} \tag{4.2.8}$$

Since $h_{1...4}$ are most likely just memory lookups, it is clear that this formula is significantly faster than averaging four cross-products. It is important to remember that n_v is not a unit vector and might need to be normalized, depending on your application.

Conclusion

Heightfields are often used to store landscapes and other static objects. For these kinds of applications, the speed of vertex normal calculation is probably not an issue, as normals will most likely be calculated offline or only once at data load.

However, for applications that use dynamic heightfields (say, to simulate the surface of a body of water or for procedural bump-map animation), speed is of the essence. This gem has shown how we can significantly improve the performance of the standard vertex normal formula by taking advantage of the special characteristics of heightfield vertices.

Sample Code

The sample code applies the fast normal formula to a heightfield, which animates between flat and a fractally-generated random landscapes. The sample code uses OpenGL and GLUT. See Color Plate 4 for some screenshots of this application.

References

[Ebert94] Ebert, D., et al., *Texturing and Modeling*, AP Professional, 1994.

[Fernandes00] Fernandes, António Ramires, "Terrain Tutorial," available online at *http://www.lighthouse3d.com/opengl/terrain/*, September, 2000.