Bump Mapping

Anders Hast

Creative Media Lab
University of Gävle
Kungsbäcksvägen 47, S-801 76 Gävle, Sweden

aht@hig.se

1 Bump mapping

Blinn [1] introduced bump mapping as a technique that makes a surface appear rough or wrinkled. This affect is achieved by perturbing the normals used in the illumination computation. Hence, it only affects the shading, not the underlying geometry. The bump map can contain height values that will affect the normal of each pixel. However, it can as an alternative contain normals that have been computed from such a bump map and this map is then called a normal map [7].

Especially moving frame bump mapping[10] [7] is closely related to shading since it includes vector interpolation, which is the essence of Phong shading. Bump mapping is also closely related to texture mapping [5]. It will also suffer from similar aliasing problems when bumps are magnified or minified.

Blinn used an approximation for the perturbed normal, which depends on the surface normal and the height information in the bump map. The perturbed normal is calculated as

$$\mathbf{n}' = \mathbf{n} + \frac{F_u(\mathbf{n} \times \mathbf{P_v}) - F_v(\mathbf{n} \times \mathbf{P_u})}{||\mathbf{n}||} \tag{1}$$

where $\mathbf{n}$ is the surface normal, $\mathbf{P_u}$ and $\mathbf{P_v}$ are the partial derivatives of the surface in the u and v directions respectively. F_u and F_v are the gradients of

the bump map. Peercy et al. [10] use another approach for computing the normal. An orthonormal frame on the surface is used to rotate the vector in the direction to the light source $\mathbf{l}$ into that local frame. As an alternative, the bump normal could be rotated into the world by the same frame, i.e. the inverse of the same matrix. However, it is more efficient to rotate the light vector so it will be correct compared to the flat normal map. This can be done per vertex and then the rotated light vectors are linearly interpolated over the polygon. Hence, the diffuse intensity is computed as $I_d = \mathbf{n}' \cdot \mathbf{l}'$, where $\mathbf{n}'$ is the normal obtained from the bump map, and $\mathbf{l}'$ is the light vector rotated by the frame $(\mathbf{t}, \mathbf{b}, \mathbf{n})$, where $\mathbf{t}$ is the tangent vector of the surface at the point with the surface normal $\mathbf{n}$ and finally $\mathbf{b}$ is the bi-normal, computed as $\mathbf{n} \times \mathbf{t}$. They are taking this approach a step further by precomputing bump normals over the whole object. Nonetheless, this is the main idea of moving frame bump mapping. The perturbed normal is

$$\mathbf{n}' = (-F_u, -F_v, 1) \tag{2}$$

which is obtained by computing the cross product of the gradients as shown by Doggett et al. [2]. They also use Prewitt masks, which is a kind of convolution kernel, to obtain F_u and F_v. Kugler [8] uses a different representation of the normal perturbation based on spherical coordinates. Sung Kim et al. [11] elaborates this idea further by directly computing the inner products for the diffuse and specular light from the perturbed normal in this representation. A hardware implementation of a bump mapping chip is proposed by Ikedo and Ohbuchi [6], where vector normalization is avoided by using vectors in angular form. Miller and Halstead [9] discuss how hardware accelerated bump mapping using standard texture-mapping hardware can be performed. An overview of other bump map approaches is given by Ernst et al. [3] and Kilgard [7].

In this type of bump mapping, the frame is used to rotate the vector in the direction to the light source. This can be done since a rotation matrix is an orthonormal frame. That is, each column in a rotation matrix is a vector and each vector has unit length and they are orthogonal to each other. This fact is seldom mentioned in computer graphics text books, but it certainly helps understanding the concept of rotation matrices. A rotation of θ degrees around the z-axis is defined as

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

It is clear that the last column is a vector aligned with the z-axis. It is also easy to see that this last vector is orthogonal to both the two first vectors since the dot product between the last column and any of the two first is zero. The two first vectors are also orthogonal since the dot product once again is zero: $\cos\theta \cdot \sin\theta + (-\sin\theta \cdot \cos\theta) + 0 \cdot 0 = 0$. Similarly it can be shown that each vector has unit length. The length of the first vector is $\sqrt{\cos^2\theta + \sin^2\theta + 0^2}$. The trigonometric unity gives that this is equal to one. The same applies for the other two columns. Furthermore, each row is also a vector and they also constitute a frame.

The interpretation of this concept is that if this frame is the same as the world coordinate system, i.e. the unity matrix, then the object is not rotated. However, if we rotate the coordinate system (or frame) the object is defined in, then the object itself is rotated in the same way as the frame is rotated. The same concept applies for both vectors and objects.

The conclusion is that if we want to rotate an object or a vector to a certain position, we use a rotation matrix that contains a local frame that defines this rotation. Moreover, this frame can be regarded as the local coordinate system where the object or vector is defined.

If we use equation (3) to rotate a position vector $\mathbf{p}$ with

$$\mathbf{p}' = \mathbf{p}R_z(\theta) \tag{4}$$

then the frame will be found on the rows of $R_z(\theta)$. Note that the row vectors are the inverse of the column vectors since the inverse of a rotation matrix is the same as its transpose. Thus, in the moving frame bump mapping approach, the inverse of the frame is used, i.e. the vectors in the frame are stored in the columns.

Homogenous coordinates [4] expands this concept by allowing the frame to be translated to any position in the world coordinate system. This is achieved by using an extra coordinate telling whether the x, y and z coordinates define a point or a vector.

References

[1] J. F. Blinn, *Simulation of Wrinkled Surfaces*, In Proceedings SIGGRAPH 78, pp. 286-292, 1978

[2] M. Doggett, A. Kugler, W. Strasser, *Displacement Mapping using Scan Conversion Hardware Architectures* Computer Graphics Forum, Vol. 20 No 1. pp 13-26. 2000.

[3] I. Enrst, H. Rüssler, H. Schultz, O. Wittig *Gouraud Bump mapping* Workshop on Graphics Hardware, pp. 47-53. 1998.

[4] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, *Computer Graphics - Principles and Practice* Addison-Wesley, 1997.

[5] P. S. Heckbert, *Survey of Texture Mapping* IEEE Computer Graphics and Applications, Nov. pp. 56-67. 1986.

[6] T. Ikedo, E. Ohbuchi, *A Realtime Rough Surface Renderer*, Proc. of Computer Graphics International 2001 (CGI2001), IEEE Computer Socity Press, July 2001.

[7] M. J. Kilgard *A Practical and Robust Bump-mapping Technique for Today s GPUs* Game Developers Conference, Advanced OpenGL Game Development. 2000.

[8] A. Kugler *IMEM: An Intelligent Memory for Bump- and Reflection-Mapping* Workshop on Graphics Hardware, pp. 113-122. 1998.

[9] G. Miller, M. Halstead, M. Clifton *On-the-Fly Texture Computation for Real-Time Surface Shading*, IEEE Computer Graphics and Applications, Vol. 18, No. 2, March-April 1998

[10] M. Peercy, A. Airey, B. Cabral, *Efficient Bump Mapping Hardware*, In proceedings of SIGGRAPH 97, August 3-8, pp 303-306. 1997.

[11] J. Sung Kim, J. Hyun Lee, K. Ho Park *A Fast and Efficient Bump Mapping Algorithm by Angular Perturbation* Computers and Graphics No. 25, pp. 401-407, 2001.