Graphics Hardware (2003)
M. Doggett, W. Heidrich, W. Mark, A. Schilling (Editors)

An Effective Hardware Architecture for Bump Mapping Using Angular Operation

S. G. Lee, W. C. Park, W. J. Lee, T. D. Han, and S. B. Yang

Department of Computer Science, Yonsei University, Seoul, Republic of Korea

Abstract

In this paper, we propose an effective bump mapping algorithm that utilizes the reference space with the polar coordinate system and also propose a new hardware architecture associated with the proposed bump mapping algorithm. The proposed architecture reduces the computations to transform the vectors from the object space into the reference space by using a new vector rotation method. It also reduces the computations for the illumination calculation by using the law of cosine. Compared with the previous approaches, the proposed architecture reduces multiplication operations up to 78%.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Bump mapping, developed by Blinn[1], can represent effectively the bumpy parts of the object surface, such as a protuberance of the skin of a peanut, in detail using geometry mapping without complex modeling[2, 3, 4, 5, 6]. The following three steps are required for each pixel to perform the conventional bump mapping[1, 2, 3, 7]. In the first step, the displacement values or height values are fetched from a 2D bump map. In the second step, both the partial differentiations and the cross-products are calculated to perturb the normal vector N. In the last step, the illumination is calculated with the perturbed normal vector N', the light vector L, and the halfway vector H after the normalization of these three vectors. To accomplish the above steps, a large amount of per-pixel computations is required.

The normal vector perturbation can be preprocessed by defining the surface-independent space, called the reference space[8, 9, 10]. Instead, the transformations of the light and the halfway vectors from the object space into the reference space should be provided for each pixel (or for each small polygon). For these transformations, the definition of a 3×3 matrix and a 3×3 matrix multiplication are required for each pixel.

In Kim et al.[9] and Kugler[11], the polar coordinate system is used to represent vectors N, L, and H for bump mapping. Compared with the Cartesian coordinate system, the polar coordinate system is an effective approach from the viewpoint of hardware requirements. Contrary to three components in the Cartesian coordinate system, only two angles are sufficient to represent a vector. The normalization of the vectors for the illumination calculation can also be eliminated. However, the matrix multiplication for the transformation or a large map for the normal vector perturbation are still required.

In this paper, we propose a new transformation method for both the light vector L and the halfway vector H represented by the polar coordinate system and present its hardware architecture. The proposed method transforms directly the vectors L and H in the object space into the vectors L' and H' in the reference space only by vector rotations, respectively. Thus the proposed method does not need any hardware for matrix transformation, but requires only a few hardware logics for the vector rotations. It also reduces the illumination calculation hardware by applying the law of cosine to the previous illumination calculation equations.

The rest of this paper is organized as follows. The related work to bump mapping hardware architecture is described in Section 2. In Section 3 we explain the proposed algorithm for the vector rotation and the illumination calculation in de-

tail. In Section 4 we illustrate the proposed bump mapping hardware architecture and compare its hardware complexity with those of other approaches. Experimental environments and results are discussed in Section 5. Conclusions are given in Section 6.

2. Background and related work

In the normal vector perturbation step of the conventional bump mapping, as shown in (1), a perturbed normal vector N' is calculated by deviating a normal vector N by the displacement value B stored in the bump map.

$$N' = N + B_u(N \times P_v) - B_v(N \times P_u), \quad (1)$$

where B_u and B_v are the partial derivatives of the bump displacement value B with respect to the bump-map parameters, u and v, respectively. P_u and P_v are the partial derivatives of the surface at point P lying in the tangent plane. Then the illumination of the perturbed normal vector N' is calculated with N', L, and H using the Phong illumination model as shown in (2).

$$I = I_a k_a + I_i(k_d(L \cdot N') + k_s(N' \cdot H)^n), \quad (2)$$

where I_a is the intensity of ambient light, I_i is the intensity of incident light, and k_a, k_d and k_s are ambient, diffuse and specular reflection coefficients, respectively.

Peercy et al.[8] propose an efficient method to support bump mapping by adding minimal hardware into the Phong shading hardware. By introducing the local tangent space (the reference space) defined by a normal vector N, a tangent vector T, and a binormal vector B, the computation of the perturbed normal is pushed into a preprocessor. Ernst et al.[10] simplify the construction of the bump map by fixing the value k into -1, where k is a constant of the equation in Peercy et al.[8] for the normal vector perturbation. In Peercy et al.[8] and Ernst et al.[10], a 3×3 matrix transformation and a normalization for the light and the halfway vectors should be completed before the illumination calculation.

The polar coordinates of a vector P are written into (φ_P, θ_P), where φ_P is an angle between the x-axis and a vector generated by projecting P onto the xy-plane and θ_P is an angle between P and the z-axis. The bump mapping methods using the polar coordinate system are discussed in Kim et al.[9], Kugler[11], and Ikedo et al.[12, 13, 14]. Kim et al.[9] gives only a small reduction to the hardware requirements for the illumination calculation by using (3) and (4) which compute the inner products in (2) with the perturbed angles $(\varphi_{n'}, \theta_{n'})$ obtained from the bump map.

$$n'_L \cdot l_L = \cos\theta_l \cos\theta_{n'} + \sin\theta_l \sin\theta_{n'} \cos(\varphi_l - \varphi_{n'}) \quad (3)$$

$$n'_L \cdot h_L = \cos\theta_h \cos\theta_{n'} + \sin\theta_h \sin\theta_{n'} \cos(\varphi_h - \varphi_{n'}) \quad (4)$$

Ikedo et al.[12, 13, 14] present a high performance Phong shading hardware supporting bump, reflection, refraction, and texture mapping in a single LSI chip. As mentioned in Kugler[11], it requires a large amount of logic for matrix operations because it has been implemented with the classical straightforward method. Moreover, it may not produce the correct reflection angle to calculate the intensity of specular light, especially in large-sized polygons.

Kugler[11] integrates the arithmetic units and the reference tables into one dedicated memory chip, called the intelligent memory (IMEM), to support texture-, bump-, and reflection-mapping. In this architecture, each normal vector is transformed from the Cartesian coordinates into the spherical coordinates. Then the transformed normal vector is perturbed by using three rotations, which are executed by two additions and one look-up table (LUT) access. After the normal vector perturbation, the illumination is calculated by referring to the diffuse and specular light maps. IMEM reduces the amount of computations by using the pre-computed LUTs and maps. However, it requires a map of large size, over 3 MBytes.

3. The proposed bump mapping algorithm

The processing flow of the proposed bump mapping algorithm consists of the Vector Rotation Stage, the Bump Vector Fetch Stage, and the Illumination Calculation Stage. In the Vector Rotation Stage, vectors in the 3D object space are transformed into the 3D reference space by the angular rotation operation. This operation generates the light and the halfway vectors, L' and H', in the reference space by transforming the vectors, L and H, in the object space. The two angles of the transformed vectors can be calculated by utilizing the projection onto the plane and the proportion onto the sphere. The detailed explanation of this method and its hardware architecture will be given in Section 3.1 and 4.1, respectively.

In the Bump Vector Fetch Stage, the perturbed normal vectors N''s are fetched from the bump vector map which is constructed in the reference (the tangent) space and is computed in the preprocess stage by the calculation method in Peercy[8] or by the filtering of the bump height map. A vector in the bump vector map is represented as $(\varphi_{N'}, \theta_{N'})$ using the polar coordinate system.

In the Illumination Calculation Stage, after the inner products, $N' \cdot L'$ and $N' \cdot H'$, are computed, the illumination is calculated by referring to the diffuse and specular tables with the values of the inner products. The algorithm for the illumination calculation and its hardware architecture are discussed in detail in Section 3.2 and 4.2, respectively.

3.1. The vector rotation

3.1.1. The proposed vector rotation method

In the proposed method, the vectors L and H are transformed directly into the tangent space by the rotation operations, where the polar coordinate (φ_N, θ_N) of the normal vector

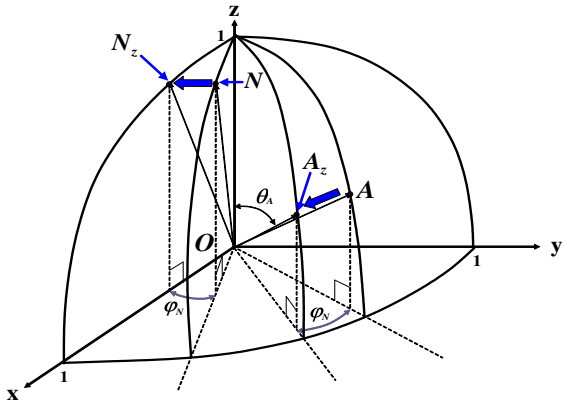

Figure 1: *The first rotation: It rotates $-\varphi_N$ around the z-axis.*

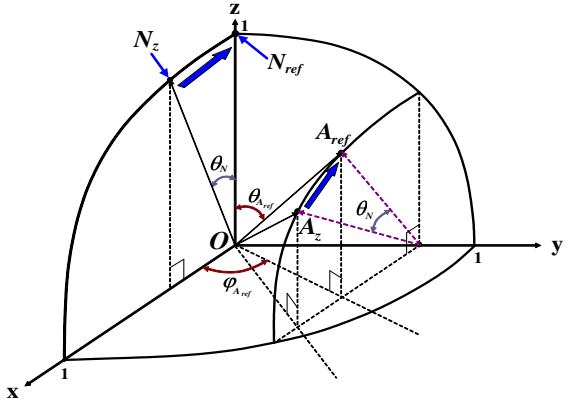

Figure 2: *The second rotation: It rotates $-\theta_N$ around the y-axis.*

N at a point on the object's surface are used as the rotation factors (angles) of the vector. Figures 1 and 2 show these rotation operations; a vector A is transformed into a vector A_{ref}. The polar coordinate $(\varphi_{A_{ref}}, \theta_{A_{ref}})$ can be found by the projection onto the xy-plane and the proportion of the angular variation on a sphere, respectively. $A = (\varphi_A, \theta_A)$ is an arbitrary vector on the world space, which corresponds to a light vector $L = (\varphi_L, \theta_L)$ and a halfway vector $H = (\varphi_H, \theta_H)$ in case of performing bump mapping.

Figure 1 shows the first rotation that rotates N and A by $-\varphi_N$ around the z-axis. Through this rotation, N and A are converted into N_z and A_z, respectively. This makes it possible to rotate accurately around the y-axis by moving N into N_z on the xz-plane.

Next, as shown in Figure 2, the second rotation is performed by rotating the transferred vectors, N_z and A_z, by $-\theta_N$ around the y-axis. As a result, N_z is identical to the unit

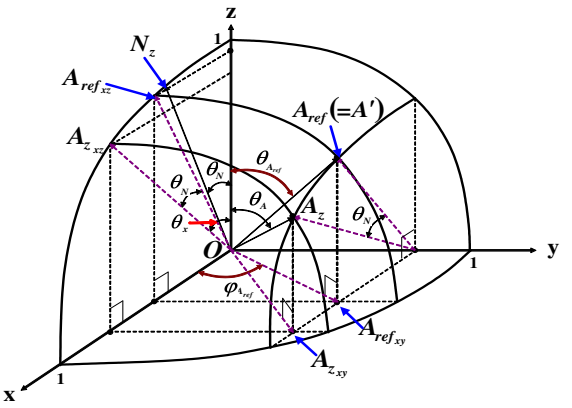

Figure 3: *The geometric information of vectors that are required to find the components of the transformed vector $A_{ref}(=A')$.*

vector with the z-axis direction in the world space and is in parallel with the normal vector N_{ref} in the reference space. Similarly, A is also moved into $A_{ref} = (\varphi_{A_{ref}}, \theta_{A_{ref}})$ which is identical to A' in the reference space.

3.1.2. The information for finding the angles of the vector after the rotation operation

Figure 3 shows the geometric information of the vectors that are required to find the polar coordinate $(\varphi_{A'}, \theta_{A'})$ of the transformed vector A'. More detailed descriptions to the newly defined vectors in Figure 3 are given below.

- $A_{z_{xz}}$: It is a unit vector that begins with the origin and ends with a point intersecting both the xz-plane and a circle which passes the end point of A_z and which is in parallel with the yz-plane. Let an angle between $A_{z_{xz}}$ and the z-axis be θ_x which is required to find an angle $\theta_{A'}$ of A'.
- $A_{z_{xy}}$: It is a vector that is generated by projecting A_z onto the xy-plane. The x and y coordinates of the end point of $A_{z_{xy}}$ are $\sin\theta_A \cos(\varphi_A - \varphi_N)$ and $\sin\theta_A \sin(\varphi_A - \varphi_N)$, respectively.
- $A_{ref_{xz}}$: It is a unit vector that begins with the origin and ends with a point intersecting both the xz-plane and a circle which passes the end point of A' and which is in parallel with the yz-plane. An angle between $A_{ref_{xz}}$ and the z-axis is $(\theta_x - \theta_N)$ and the x and z coordinates of the end point of $A_{ref_{xz}}$ are $\sin(\theta_x - \theta_N)$ and $\cos(\theta_x - \theta_N)$, respectively.
- $A_{ref_{xy}}$: It is a vector that is generated by projecting A' onto the xy-plane. The x and y coordinates of the end point of $A_{ref_{xy}}$ are $\sin(\theta_x - \theta_N)$ and $\sin\theta_A \sin(\varphi_A - \varphi_N)$, respectively.

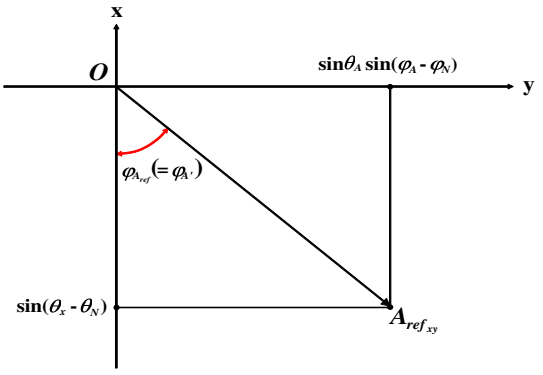

Figure 4: *The calculation of $\varphi_{A'}$: This method calculates an angle $\varphi_{A'}$ between a projected vector and the x-axis by projecting A' onto the xy-plane.*

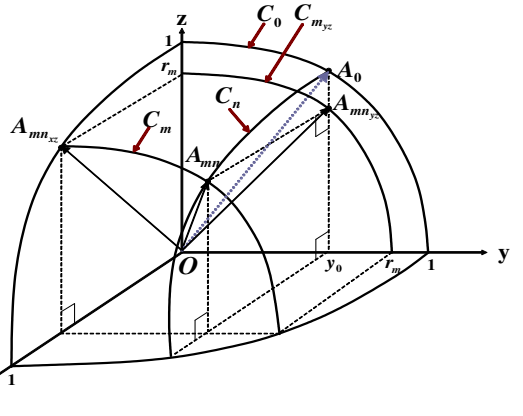

Figure 5: *The geometric relationship of the vectors on a sphere.*

By applying the law of cosine to the x coordinate of $A_{z_{xz}}$, we can find θ_x which is expressed as (5).

$$\theta_x = Sin^{-1}\frac{1}{2}\left[\sin\left(\frac{\theta_A+(\varphi_A-\varphi_N)}{2}\right)+\sin\left(\frac{\theta_A-(\varphi_A-\varphi_N)}{2}\right)\right] \quad (5)$$

Let X and Y be

$$X = \frac{\theta_A+(\varphi_A-\varphi_N)}{2},\ Y = \frac{\theta_A-(\varphi_A-\varphi_N)}{2}. \quad (6)$$

Then θ_x can be rewritten as

$$\theta_x = Sin^{-1}\left[\frac{\sin X+\sin Y}{2}\right]. \quad (7)$$

3.1.3. The calculation of $\varphi_{A'}$

To find an angle $\varphi_{A'}$ of A', the projection of A' onto the xy-plane is required as shown in Figure 4. In the figure, $\varphi_{A'}$ can be calculated with

$$\varphi_{A'} = Tan^{-1}\left[\frac{\sin\theta_A\sin(\varphi_A-\varphi_N)}{\sin(\theta_x-\theta_N)}\right]. \quad (8)$$

Let $Z = \theta_x - \theta_N$. Then $\varphi_{A'}$ can be rewritten as in (9) by applying the law of cosine to $\sin\theta_A\sin(\varphi_A-\varphi_N)$.

$$\varphi_{A'} = Tan^{-1}\left[\frac{\cos Y-\cos X}{2\sin Z}\right] \quad (9)$$

3.1.4. The derivation of an angular equation using the geometric relationship of the vectors on a sphere

Compared with the method for finding $\varphi_{A'}$, it is relatively complex to find the angle $\theta_{A'}$ between A' and the z-axis. We will explain how to calculate the angle $\theta_{A'}$ from this section through Section 3.1.6. First, specifying the geometric relationship of the vectors on a sphere is required to find $\theta_{A'}$. In this section, the relationship of the vectors on a sphere is specified and the calculation method for finding $\theta_{A_{mn}}$ of an arbitrary vector A_{mn} is described.

Figure 5 shows the geometric relationship among a vector A_{mn}, the xz-plane, and the yz-plane. Detailed descriptions for the circles and the vectors defined in Figure 5 are given below.

- C_m: It is a circle that is in parallel with the yz-plane and passes a point $(\varphi_{A_{mn}}, \theta_{A_{mn}})$. Let the radius of this circle be r_m.
- $C_{m_{yz}}$: It is a circle that is generated by projecting the circle C_m onto the yz-plane.
- C_n: It is a circle that is in parallel with the xz-plane and passes a point $(\varphi_{A_{mn}}, \theta_{A_{mn}})$.
- $A_{mn_{xz}}$: It is a vector that begins with the origin and ends with a point $(0, \theta_{A_{mn_{xz}}})$ at which the circle C_m intersects the xz-plane.
- A_0: It is a vector that begins with the origin and ends with a point $(\varphi_{A_0}, \theta_{A_0})$ at which the circle C_n intersects the yz-plane. Let the y coordinate of the point $(\varphi_{A_0}, \theta_{A_0})$ be y_0.
- $A_{mn_{yz}}$: It is a vector that is generated by projecting A_{mn} onto the yz-plane. The end point, $(\varphi_{A_{mn_{yz}}}, \theta_{A_{mn_{yz}}})$, of $A_{mn_{yz}}$ is on the circle $C_{m_{yz}}$.

In Figure 5, we assume two arbitrary vectors that begin with the origin and end with points at which the plane in parallel with the xz-plane intersects the circles, C_m and $C_{m_{yz}}$. Then, for C_m the variation range of an angle between A_{mn} and the z-axis is from 0 to 90 - $\theta_{A_{mn_{xz}}}$. Similarly, for $C_{m_{yz}}$ the variation range is from 0 to 90. When these vectors move on C_m and $C_{m_{yz}}$ under the above assumption, the ratio of the angular variation to the variation range of each vector for C_m is equal to that of $C_{m_{yz}}$. Therefore, the following expression holds.

$$\frac{\theta_{A_{mn}}-\theta_{A_{mn_{xz}}}}{90-\theta_{A_{mn_{xz}}}} = \frac{\theta_{A_{mn_{yz}}}}{90} \quad (10)$$

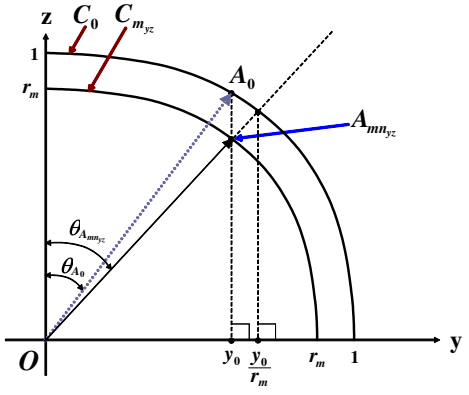

Figure 6: *The projection of A_{mn} and its related components onto the yz-plane.*

Hence, $\theta_{A_{mn}}$ can be written as

$$\theta_{A_{mn}} = \theta_{A_{mn_{xz}}} + \frac{\theta_{A_{mn_{yz}}}}{90}(90 - \theta_{A_{mn_{xz}}}). \quad (11)$$

We will define the extent to the angular variation, θ_{prop}, as in (12), whose calculation method will be described in detail in Section 3.1.5. Finally, $\theta_{A_{mn}}$ can be rewritten as (13).

$$\theta_{prop} = \frac{\theta_{A_{mn_{yz}}}}{90} \quad (12)$$

$$\theta_{A_{mn}} = \theta_{A_{mn_{xz}}} + \theta_{prop}(90 - \theta_{A_{mn_{xz}}}) \quad (13)$$

3.1.5. The calculation of θ_{prop} depending on the location of vectors

θ_{prop} can be calculated by using an angle $\theta_{A_{mn_{yz}}}$ of $A_{mn_{yz}}$ as shown in Figure 5. Figure 6 shows the projection of A_{mn} and its related components onto the yz-plane. As shown in Figure 6, the y coordinate of a point at which the extension line of $A_{mn_{yz}}$ intersects a unit circle C_0 in the yz-plane is y_0/r_m. The cosine of an angle between $A_{mn_{yz}}$ and the y-axis is calculated as in (14).

$$\frac{y_0}{r_m} = \cos(90 - \theta_{A_{mn_{yz}}}) \quad (14)$$

$$\theta_{A_{mn_{yz}}} = 90 - Cos^{-1}\frac{y_0}{r_m} \quad (15)$$

As a result, θ_{prop} is calculated by (16) , where both y_0 and r_m are already calculated and θ_{prop} ranges from 0 to 1.

$$\theta_{prop} = 1 - \frac{1}{90}Cos^{-1}\frac{y_0}{r_m} \quad (16)$$

3.1.6. The calculation of $\theta_{A'}$

The angle $\theta_{A'}$ of the transformed vector A' is calculated by using the methods derived from Sections 3.1.4 and 3.1.5. From (13), $\theta_{A'}$ can be calculated as follows.

$$\theta_{A'} = \theta_{A'_{xz}} + (90 - \theta_{A'_{xz}}) \times \theta_{prop} \quad (17)$$

$\theta_{A'_{xz}}$ $(= \theta_{A_{ref_{xz}}})$ of A' is $(\theta_x - \theta_N)$, thus (17) is rewritten into (18).

$$\theta_{A'} = (\theta_x - \theta_N) + [90 - (\theta_x - \theta_N)] \times \theta_{prop} \quad (18)$$

$\theta_{A'}$ can be rewritten as follows.

$$\theta_{A'} = Z + (90 - Z) \times \theta_{prop} \quad (19)$$

θ_{prop} that can be calculated by the method presented in Section 3.1.4 is represented as (20).

$$\theta_{prop} = 1 - \frac{1}{90}Cos^{-1}\frac{\sin\theta_A \sin(\varphi_A - \varphi_N)}{\cos(\theta_x - \theta_N)} \quad (20)$$

Through the use of the law of cosine along with X, Y, and Z, (20) can be rewritten into (21). Finally, $\theta_{A'}$ can be determined by using (19) and (21).

$$\theta_{prop} = 1 - \frac{1}{90}Cos^{-1}\frac{\cos Y - \cos X}{2\cos Z} \quad (21)$$

3.2. The illumination calculation

In the Illumination Calculation Stage, the intensity of a pixel is computed with the light and the halfway vectors, $L' = (\varphi_{L'}, \theta_{L'})$ and $H' = (\varphi_{H'}, \theta_{H'})$, and the perturbed normal vector $N' = (\varphi_{N'}, \theta_{N'})$. In order to calculate the inner products of the vectors in (2), the appropriate vectors represented by the polar coordinates should be transformed into the vectors in the Cartesian coordinates. The inner products of the transformed vectors, $N' \cdot L'$ and $N' \cdot H'$, are calculated as follows.

$$N' \cdot L' = \cos\theta_{L'}\cos\theta_{N'} + \sin\theta_{L'}\sin\theta_{N'}\cos(\varphi_{L'} - \varphi_{N'}) \quad (22)$$

$$N' \cdot H' = \cos\theta_{H'}\cos\theta_{N'} + \sin\theta_{H'}\sin\theta_{N'}\cos(\varphi_{H'} - \varphi_{N'}) \quad (23)$$

(22) and (23) are identical to (3) and (4) in Kim et al.[9], respectively. However, applying the law of cosine to these equations makes it possible to reduce the amount of computations for the inner products. (24) and (25) are the final equations of the inner products, in which the numbers of multiplications and cosines, required in the inner products of the vectors represented by the polar coordinates, are reduced from 6 and 10 to 2 and 6, respectively.

$$N'L' = \frac{1}{2}[\cos(\theta_{L'} + \theta_{N'}) + \cos(\theta_{L'} - \theta_{N'})] + \frac{1}{2}[\cos(\theta_{L'} + \theta_{N'}) - \cos(\theta_{L'} - \theta_{N'})]\cos(\varphi_{L'} - \varphi_{N'}) \quad (24)$$

$$N'H' = \frac{1}{2}[\cos(\theta_{H'} + \theta_{N'}) + \cos(\theta_{H'} - \theta_{N'})] + \frac{1}{2}[\cos(\theta_{H'} + \theta_{N'}) - \cos(\theta_{H'} - \theta_{N'})]\cos(\varphi_{H'} - \varphi_{N'}) \quad (25)$$

The intensities of the diffuse and the specular lights are computed by multiplying these inner products by the light parameters. Then the final color of a bump-mapped pixel is determined by adding these intensities together.

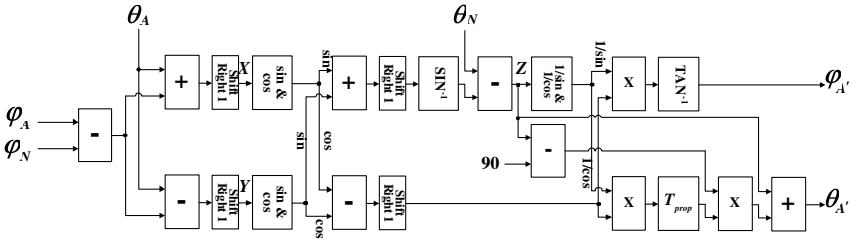

Figure 7: *The hardware architecture for the proposed vector rotation method.*

4. The hardware architecture for the proposed bump mapping algorithm

4.1. The vector rotation unit

The hardware architecture for the proposed vector rotation method is shown in Figure 7. It requires six tables, three multipliers, and eight adders/subtractors to implement (9) and (19). For the sine and cosine calculations, we use the sin & cos LUTs to reduce the computations for per-pixel operations. The reciprocals of the sine and cosine are calculated by combining the sine and cosine calculation step and the reciprocal calculation step, not by two separate steps.

For arbitrary values of y_k's, θ_{prop}'s are precomputed by (26) and are stored in the proportion table T_{prop}.

$$\theta_{prop} = 1 - \frac{Cos^{-1} y_k}{90} \tag{26}$$

Note that y_k's in the above equation are calculated by (27) which is obtained from (21). Therefore, θ_{prop}'s are fetched from T_{prop} with the indices of y_k's.

$$y_k = \frac{\cos Y - \cos X}{2 \cos Z} \tag{27}$$

4.2. The illumination calculation unit

Figure 8 shows the whole bump mapping hardware architecture including both the vector rotation unit and the illumination calculation unit. The illumination calculation unit consists of two parts that calculate the intensities of the diffuse and the specular light. The intensities of lights are obtained from the light tables referred to by the values of the inner products. The proposed illumination calculation unit can be implemented with six cosine tables, one diffuse table, one specular table, two multipliers, and thirteen adders/subtractors. Compared with Kim's architecture[9] requiring twelve tables, six multipliers, and five adders/subtractors, the proposed architecture is more efficient in terms of hardware complexity. More detailed comparisons of the proposed architecture with other architectures are made in Section 4.3.

The light table method for the illumination calculation,

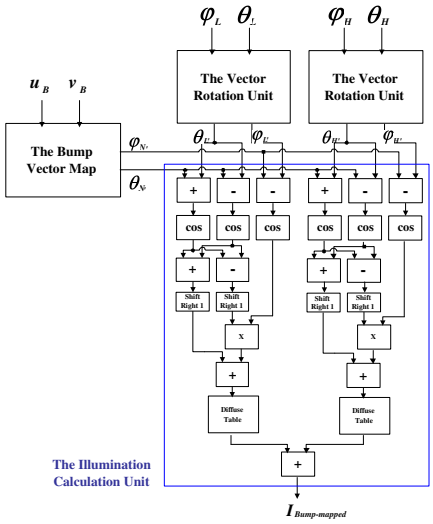

Figure 8: *The proposed bump mapping hardware.*

used in Kim et al.[9], finds the intensities of a pixel by referring to the diffuse and the specular light tables which are precomputed. In this method, the table entries are indexed by scalar values, i.e., inner product values. Therefore, the size of the table is $2^8 \times 16 \sim 2^{10} \times 16$ bits, which is quite small. However, in the light map method[11], the map's entries are indexed by vector values with two or three coordinates. Thus the size of the map is at least $2^{20} \times 16$ bits, which is much larger size than that of the light table method.

4.3. The comparison of other existing approaches

Table 1 shows the comparison of the proposed architecture with other architectures based on the hardware complexity. For the comparison, we split the process of bump mapping into two parts, the illumination environment setup and the illumination calculation. The front-end is the part that prepares the environment for the illumination calculation and includes the transformation into the reference space, the normal vector perturbation, and the normalization.

First, we will observe the hardware complexity of the il-

Architecture		Peercy[8]	Kugler[11]	Kim[9]	Ernst[10]	Ours
Coordinate System		Cartesian	Polar	Polar	Cartesian	Polar
Processing Steps	Illumination Environment Setup	27 Multipliers 9 Division Units 3 SQRTs 18 Adders	1 Map 5 Multipliers 3 LUTs 2 Adders	31 Multipliers 6 LUTs 18 Adders	27 Multipliers 9 Division Units 3 SQRTs 18 Adders	6 Multipliers 12 LUTs 16 Adders
	Illumination Calculation	Not Available	8 Multipliers 2 Maps 5 LUTs 4 Adders	6 Multipliers 12 LUTs 5 Adders	7 Multipliers 1 LUT 5 Adders	2 Multipliers 8 LUTs 11 Adders

Table 1: *The comparison of the architectures based on the hardware complexity.*

lumination environment setup. Twenty multiplications and eight additions are required to perform the normal vector perturbation by the Blinn's method[1]. To simplify this computation, Kugler[11] uses LUTs and a map. His architecture is implemented by five multipliers, two adders, three LUTs, and one map; but the size of the map reaches 32 Kbytes. For Peercy[8] and Ernst[10] that use the reference space, the transformation of L and H into the reference space, which requires two 3×3 matrix operations, is implemented by eighteen multipliers and twelve adders, and the normalization of the transformed vectors is organized into nine multipliers, three SQRTs, nine division units, and six adders. In Kim et al.[9], the transformation computations in the polar coordinate system is performed by thirty one multipliers, six LUTs, and eighteen adders. However, the above-mentioned methods[8, 9, 10] have to reconstruct a matrix for the transformation of each pixel or polygon, hence these methods require larger amount of computations than those shown in Table 1. The proposed architecture reduces the hardware requirements for the transformation into the reference space to six multipliers, twelve LUTs, and sixteen adders by using only rotations and table references without any matrix operations.

In the Illumination Calculation Stage, Kugler's approach calculates the illumination through the address generation of the light map, the access to the light map, and the color blending without the inner product operation. His method uses eight multipliers, four adders, and five LUTs to generate the address of the light map and exploits two 1-MB light maps to calculate the intensity of the light. Ernst uses a straightforward method for the Phong illumination model, which requires seven multipliers, five adders, and one power table. Kim's architecture performs the illumination calculation by computing the inner product of the vectors with angles and by referring to the diffuse/specular light table with the computed inner-product value. Their illumination calculation unit is composed of six multipliers, twelve LUTs, and five adders. We use the same method as the one used in Kim et al. but two multipliers, eight LUTs, and eleven adders are only needed for our architecture.

In comparisons with other approaches using the Cartesian coordinate system, the amount of hardware for the illumination calculation is increased a little, but the amount of hardware for the illumination environment setup is decreased relatively a lot. Observe that our hardware architecture reduces the hardware requirement considerably, compared with other polar-coordinate approaches.

5. Experimental results

The algorithm and hardware architecture proposed in this paper have been simulated in C. For the simulation, we modified Mesa 3.0[15] to implement the conventional bump mapping method and the proposed bump mapping method. To investigate the quality differences between ours and the conventional approach, we performed texture- and bump-mapping using various objects with various texture- and bump-maps. Figure 9 shows the results generated by mapping a wooden wall texture- and bump-map onto a plane. Figures 9(a) and 9(b) are made by the conventional method using the object space with the Cartesian coordinate system and by the proposed method using the reference space with the polar coordinate system, respectively. When Comparing these images, there is little difference in the quality between these two images, to the extent of not being differentiated by the naked eyes. Figures 10(a) and 10(b) show the results of mapping a brick wall texture- and bump-map onto a cube by the conventional method and by the proposed method, respectively. There is also little difference in the image quality as in the case of a plane in Figure 9.

Figure 11 has been generated by mapping a map of the world onto a sphere. The images of this figure are obtained by mapping a texture- and bump-map with 512×256 resolution onto a sphere object with 512×512 resolution. In Figure 11, the overall images don't look vivid because these mapping methods wear the maps converted from 512×256 resolution into 512×512 resolution on the surface of the object. However, we can hardly differentiate the image quality between these two images generated by both methods.

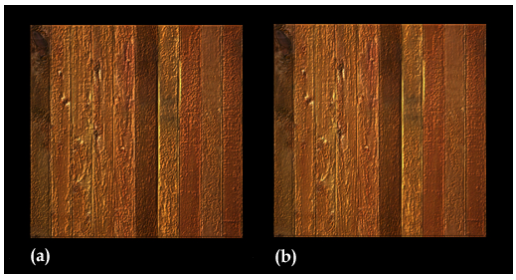

Figure 9: *Images generated by mapping a wooden wall onto a plane. (a) the conventional method (b) the proposed method*

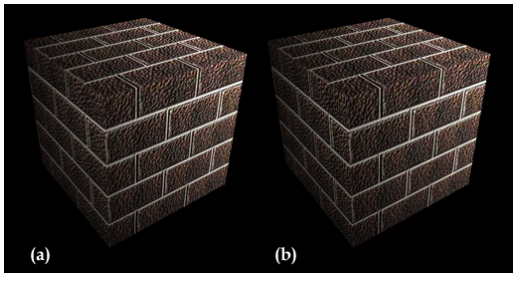

Figure 10: *Images generated by mapping a brick wall onto a cube. (a) the conventional method (b) the proposed method*

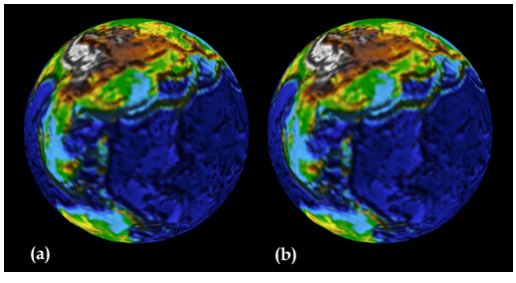

Figure 11: *Images generated by mapping a map of the world onto a sphere. (a) the conventional method (b) the proposed method*

6. Conclusions

In this paper, we have proposed a bump mapping method with the effective vector rotation and illumination calculation algorithm. The proposed architecture reduces a large amount of computations and hardwares required for the transformation and the illumination calculation in bump mapping. Furthermore, it could also generate nearly the same quality of images as the conventional method.

Acknowledgements

This work is supported by the NRL Fund from the Ministry of Science & Technology of Korea.

References

1. J. F. Blinn, "Simulation of Wrinkled Surfaces," *ACM Computer Graphics (Proc. of SIGGRAPH '78)*, pp. 286–292, 1978.
2. A. Watt, *3D Computer Graphics, 3rd Ed.*, Addison Wesley, 2000.
3. T. Moller and E. Haines, *Real-Time Rendering*, A K Peters, Ltd., 1999.
4. NVIDIA Corp., http://www.nvidia.com/view.asp?PAGE=products.
5. T. McReynolds, D. Blythe, B. Grantham, and S. Nelson, "Advanced Graphics Programming Techniques Using OpenGL," In *Course Notes of SIGGRAPH '98*, 1998.
6. M. Kilgard, "A Practical and Robust Bump-mapping Technique for Today's GPUs," NVIDIA Corporation, 2000.
7. G. Miller, M. Halstead, and M. Clifton, "On-the-Fly Texture Computation for Real-Time Surface Shading," *IEEE Computer Graphics and Applications* **18**(2), pp. 44–58, 1998.
8. M. Peercy, J. Airey, and B. Cabral, "Efficient Bump Mapping Hardware," *Computer Graphics* **31**(4), pp. 303–306, 1997.
9. J. S. Kim, J. H. Lee, and K. H. Park, "A fast and efficient bump mapping algorithm by angular perturbation," *Computers and Graphics* **25**(5), pp. 401–407, 2001.
10. I. Ernst, H. Russeler, H. Schulz, and O. Wittig, "Gouraud Bump Mapping," In *Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pp. 47–53, 1998.
11. A. Kugler, "IMEM: an intelligent memory for bump- and reflection-mapping," In *Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pp. 113–122, 1998.
12. T. Ikedo and J. Ma, "An Advanced Graphics Chip with Bump-mapped Phong Shading," In *Proc. of IEEE Computer Graphics International '97*, pp. 156–165, 1997.
13. T. Ikedo and J. Ma, "The Truga 001: A Scalable Rendering Processor," *IEEE Computer Graphics and Applications* **18**(2), pp. 59–79, 1998.
14. T. Ikedo and E. Obuchi, "A Realtime Rough Surface Renderer," In *Proc. of IEEE Computer Graphics International 2001*, pp. 355–358, 2001.
15. The Mesa 3D Graphics Library, http://www.mesa3d.org.