The Transformation between Positional Space and Texture Space

Kevin Wu
Computer Systems Laboratory
HP Laboratories Palo Alto
HPL-1999-103
September, 1999

E-mail: kevinwu@hpl.hp.com

computer graphics, transformations, texture mapping

In a 3D graphics application, a common operation is the binding of a 2D texture onto a triangle in a 3D positional space. For light-dependent or view-dependent texture mapping, a need may arise in the graphics pipeline to transform the light or view direction vector into texture space. The affine transformation is unique within the plane of the triangle because the texture coordinates and the position coordinates are specified at three vertices. In the direction perpendicular to this plane, the scale factor between positional and texture spaces is unconstrained and the solution is not unique. In particular, the choice of the scale factor is quite flexible when the texture is anisotropically scaled across the surface. This paper describes some ways to calculate a scale factor with desirable properties and a transformation between positional space and texture space given a triangle's positional coordinates and texture coordinates.

Internal Accession Date Only

1. Introduction

Texture mapping is a shading operation in a graphics pipeline for modulating an attribute across geometric primitives to create the appearance of increased feature detail without higher geometric complexity. In the common case, a texture in the form of a color image is bound to an object by associating texture coordinates with vertices of polygons. Typically, the texture is static: the image and its binding to the surface do not change. The texture may be a digital photograph of a bumpy surface. Yet, a textured polygon has the appearance of a flat surface with wallpaper applied to it because the bumps do not change the occlusions, highlights, and shadows as the camera and lights move.

Texture mapping appears more realistic when the texture changes dynamically by depending on the viewing direction [1] or the light direction [2]. View- and light-dependent texture mapping begin with a photographer taking several pictures of a real scene and varying the camera location or light location. Each photograph is associated with a known camera and/or light position. A good approximation for a particular photograph is that a single direction describes the camera and/or light when they are far enough away from the scene. In the graphics system performing view- or light-dependent texture mapping, the synthetic camera or light moves around a geometric scene, and the most suitable texture is selected from among the digital photographs. The texture needs to be the closest match between the synthetic camera or light direction and the physical camera or light direction, respectively.

Textures can be bound to polygonal vertices in an arbitrary manner. In general, texture images can be warped onto surfaces. While texture warps can be nonlinear, we restrict them to affine transformations because interpolation of texture coordinates is linear in world space (or equivalently, rational-linear in screen space [3]) in present-day texture mapping hardware. On small triangles, this constraint is acceptable in real applications. The most general warp under consideration is an anisotropic scale with rotation and translation. This complicates the mapping of a direction vector in the positional space of the synthetic camera or light to texture space corresponding to a known physical camera or light direction. The mapping is unique within the plane of a triangle. However, the scalar mapping of the normal vector between positional space and texture space is unconstrained, in general, because the 2D texture has no binding beyond the triangle's plane. Thus, we are free to choose a mapping perpendicular to the plane as long as the assignment simplifies to the obvious choice of an angle-preserving transformation (isotropic scaling with rotation and translation) when texture warping is absent.

This paper describes a transformation that maps the positional space (for example, world space) of the geometry, camera, and lights to texture space. This is a general solution for a triangle in 3D positional space with a 2D texture bound to the three vertices. When the texture is warped across the triangle, the problem is unconstrained. The recommended solutions have desirable properties, and they simplify to the obvious choice of an angle-preserving transformation in the absence of texture warping.

2. Texture Binding

A graphics application controls the binding of a 2D texture image to a triangle in 3D positional space. The texture coordinates $(s \quad t)$ are associated with the vertex position $(x \quad y \quad z)$. In matrix notation, the transformation of positional coordinates to texture coordinates takes the following form:

$$\mathbf{c} = \mathbf{A}\mathbf{p}$$

$$\begin{bmatrix} s \\ t \\ u \\ 1 \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1}$$

where $u = 0$ for the 2D texture in 3D texture space. We seek an affine matrix $\mathbf{A}$ that maps the positional coordinates at three vertices $\mathbf{p}_0$, $\mathbf{p}_1$, and $\mathbf{p}_2$ to the corresponding texture coordinates $\mathbf{c}_0$, $\mathbf{c}_1$, and $\mathbf{c}_2$, respectively. These constraints specify the mapping within the plane of the polygon. The specified binding says nothing about how a direction vector that is normal to the triangle transforms to a 3D texture space $(s \quad t \quad u)$. A desirable property of the transformation would be for the triangle's normal vector to transform to a vector perpendicular to the 2D texture space, or a vector $(0 \quad 0 \quad u)$ in 3D texture space for some value of $u \neq 0$. What remains to be determined is the value of u for a triangle normal vector of a given length in positional space. While a matrix that transforms the triangle normal vector to an arbitrary vector $(s \quad t \quad u)$ is possible, its effect is unlikely to be useful for view- or light-dependent texture mapping.

3. Calculation of the Position-to-Texture Transformation

The position-to-texture transformation needs to be computed once per triangle. Since the frequency of these calculations is high, real-time systems can benefit from the simplest solution. We begin with the most general formulation before proceeding with a simpler one that meets our needs for transforming direction vectors.

3.1 The General Formulation

Complete specification of the mapping between 3D positional space and 3D texture space requires a rule for transforming the normal vectors. Assuming that the positional and texture coordinates of the triangle's three vertices are not collinear, the normal vectors in positional and texture spaces can be calculated by forming the cross products of corresponding triangle edges. These two cross products point in the triangle's normal directions of the two spaces.

$$\begin{aligned} \mathbf{d}_0 &= \mathbf{c}_1 - \mathbf{c}_0 & \mathbf{q}_0 &= \mathbf{p}_1 - \mathbf{p}_0 \\ \mathbf{d}_1 &= \mathbf{c}_2 - \mathbf{c}_1 & \mathbf{q}_1 &= \mathbf{p}_2 - \mathbf{p}_1 \\ \mathbf{m} &= \mathbf{d}_0 \times \mathbf{d}_1 & \mathbf{n} &= \mathbf{q}_0 \times \mathbf{q}_1 \end{aligned} \tag{2}$$

Figure 1 shows an example of the configuration of these vectors. In texture space, $\mathbf{d}_0$ and $\mathbf{d}_1$ are two edge vectors and $\mathbf{m}$ is their cross product; in positional space, $\mathbf{q}_0$, $\mathbf{q}_1$, and $\mathbf{n}$ are the corresponding vectors.

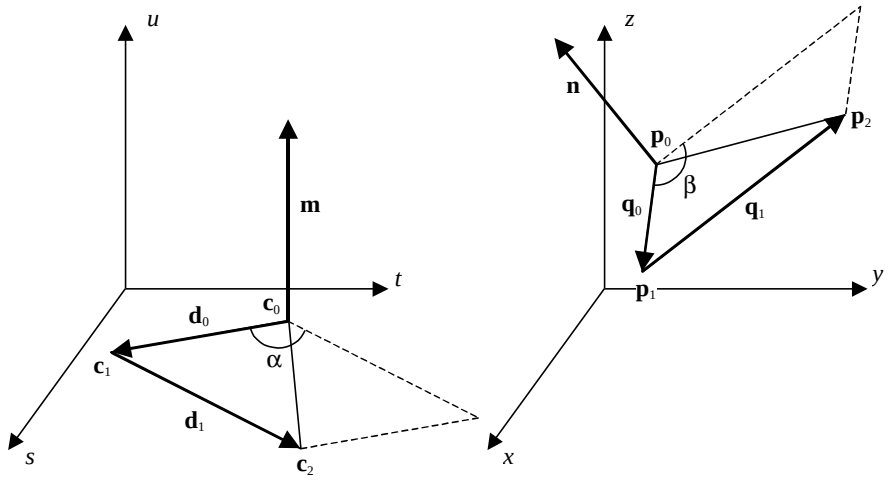

Figure 1: A triangle in texture and positional spaces.

The magnitude of a cross product vector is the product of the magnitudes of the individual vectors and the sine of the angle between them [4]. For the cross product vectors in texture and positional space, the magnitudes are

$$|\mathbf{m}| = |\mathbf{d}_0||\mathbf{d}_1|\sin a \qquad |\mathbf{n}| = |\mathbf{q}_0||\mathbf{q}_1|\sin b \tag{3}$$

where a is the angle between $\mathbf{d}_0$ and $\mathbf{d}_1$, and b is the angle between $\mathbf{q}_0$ and $\mathbf{q}_1$. The magnitude of a cross product is the area of a parallelogram formed with the individual vectors, or equivalently, twice the area of the triangle.

Let the normalized cross product vectors be denoted by the circumflex mark.

$$\hat{\mathbf{m}} = \frac{\mathbf{m}}{|\mathbf{m}|} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^{\mathrm{T}} \qquad \hat{\mathbf{n}} = \frac{\mathbf{n}}{|\mathbf{n}|} = \begin{bmatrix} n_x & n_y & n_z & 0 \end{bmatrix}^{\mathrm{T}} \tag{4}$$

A unique solution for the matrix $\mathbf{A}$ in Equation (1) requires four corresponding points, three of which are specified at the vertices of the triangle. The fourth comes from adding the normal

vectors $\hat{\mathbf{m}}$ and $\hat{\mathbf{n}}$ to corresponding points on the triangle, for example, $\mathbf{c}_0$ and $\mathbf{p}_0$. The full matrix form of Equation (1) becomes the following:

$$\begin{gathered}\mathbf{C} = \mathbf{AP} \\ \begin{bmatrix}\mathbf{c}_0 & \mathbf{c}_1 & \mathbf{c}_2 & \mathbf{c}_0 + a\hat{\mathbf{m}}\end{bmatrix} = \mathbf{A}\begin{bmatrix}\mathbf{p}_0 & \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_0 + b\hat{\mathbf{n}}\end{bmatrix} \\ \begin{bmatrix} s_0 & s_1 & s_2 & s_0 \\ t_0 & t_1 & t_2 & t_0 \\ 0 & 0 & 0 & a \\ 1 & 1 & 1 & 1 \end{bmatrix} = \mathbf{A}\begin{bmatrix} x_0 & x_1 & x_2 & x_0 + bn_x \\ y_0 & y_1 & y_2 & y_0 + bn_y \\ z_0 & z_1 & z_2 & z_0 + bn_z \\ 1 & 1 & 1 & 1 \end{bmatrix}\end{gathered} \tag{5}$$

where a and b are scale factors yet to be determined. The matrix $\mathbf{A} = \mathbf{CP}^{-1}$ transforms *points* in positional space to texture space. $\mathbf{P}^{-1}$ is a general 4×4 matrix, and its calculation would require a general-purpose matrix inversion routine, which is usually slow. Fortunately, we need not calculate $\mathbf{P}^{-1}$ and $\mathbf{A}$. For view- and light-dependent texture mapping, we seek the matrix that transforms *direction vectors* in positional space to texture space. This matrix is the same as the one that applies to surface-normal vectors, and its value is the transpose of the inverse of $\mathbf{A}$ [5]. So instead of calculating $\mathbf{A}$, we solve for its inverse.

$$\mathbf{A}^{-1} = \mathbf{PC}^{-1} \tag{6}$$

The matrix $\mathbf{C}$ has a simple form, as does its inverse.

$$\mathbf{C}^{-1} = \frac{1}{\Omega}\begin{bmatrix} t_1 - t_2 & s_2 - s_1 & -\Omega/a & w_0 \\ t_2 - t_0 & s_0 - s_2 & 0 & w_1 \\ t_0 - t_1 & s_1 - s_0 & 0 & w_2 \\ 0 & 0 & \Omega/a & 0 \end{bmatrix} \qquad \text{where} \quad \begin{aligned} w_0 &= s_1t_2 - s_2t_1 \\ w_1 &= s_2t_0 - s_0t_2 \\ w_2 &= s_0t_1 - s_1t_0 \\ \Omega &= w_0 + w_1 + w_2 \end{aligned} \tag{7}$$

A property of this matrix is that the sum of the columns is $\begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}$. Given the form of matrix $\mathbf{P}$, this turns out to be a requirement for $\mathbf{A}^{-1}$ to be affine. The upper-left 3×3 submatrix of $\left(\mathbf{A}^{-1}\right)^{\mathrm{T}}$ is used for transforming the three components of direction vectors [5]. The full 4×4 matrix is given below. The fourth row is not needed for direction vectors.

$$\left(\mathbf{A}^{-1}\right)^{\mathrm{T}} = \left(\mathbf{PC}^{-1}\right)^{\mathrm{T}} = \frac{1}{\Omega}\begin{bmatrix} (t_1 - t_2)\mathbf{p}_0{}^{\mathrm{T}} + (t_2 - t_0)\mathbf{p}_1{}^{\mathrm{T}} + (t_0 - t_1)\mathbf{p}_2{}^{\mathrm{T}} \\ (s_2 - s_1)\mathbf{p}_0{}^{\mathrm{T}} + (s_0 - s_2)\mathbf{p}_1{}^{\mathrm{T}} + (s_1 - s_0)\mathbf{p}_2{}^{\mathrm{T}} \\ \dfrac{b\Omega}{a}\hat{\mathbf{n}}^{\mathrm{T}} \\ w_0\mathbf{p}_0{}^{\mathrm{T}} + w_1\mathbf{p}_1{}^{\mathrm{T}} + w_2\mathbf{p}_2{}^{\mathrm{T}} \end{bmatrix} \tag{8}$$

All values in this equation are known except for a and b, the lengths of the triangle normal vectors in texture space and positional space, respectively. The first two rows depend only on the texture binding in the plane of the triangle; they are independent of a and b, which describe the scaling of the normal vector. The third row depends only on the polygon normal vector scaled by the relative value b/a; this row is independent of the texture binding. The complexity of calculating the ratio b/a depends on whether the texture binding is isotropic or anisotropic. We defer the discussion of these two cases to Sections 3.3 and 3.4.

3.2 A Simplified Formulation

The position-to-texture transformation is computed once per triangle so minimizing the number of arithmetic operations is desirable in real-time systems. Calculation of the matrix $\left(\mathbf{A}^{-1}\right)^{\mathrm{T}}$ in Equation (8) involves 4×4 matrices, yet we need only the upper-left 3×3 submatrix to transform direction vectors. The goal of this section is to calculate the 3×3 matrix directly.

Linear systems theory often encounters block matrices, where the elements of a matrix are themselves matrices [6]. Consider the following representation of the matrix $\mathbf{A}$ as a block matrix.

$$\mathbf{A} = \begin{bmatrix} \mathbf{B} & \mathbf{E} \\ \mathbf{0} & \mathbf{F} \end{bmatrix} \tag{9}$$

If $\mathbf{A}$ is an affine matrix for transforming points and $\mathbf{B}$ is the upper-left 3×3 submatrix, then $\mathbf{B}$ is the matrix for transforming tangent vectors (a.k.a. edge vectors or the differences between pairs of points) [5]. This suggests a simplified method for formulating the problem: we seek a matrix $\mathbf{B}$, which transforms the edge vectors from positional space to texture space. We select two triangle edges and the edge representing the triangle normal from Figure 1.

$$\begin{aligned}
\mathbf{D} &= \mathbf{BQ} \\
\begin{bmatrix} \mathbf{c}_1 - \mathbf{c}_0 & \mathbf{c}_2 - \mathbf{c}_1 & (\mathbf{c}_0 + a\hat{\mathbf{m}}) - \mathbf{c}_0 \end{bmatrix} &= \mathbf{B}\begin{bmatrix} \mathbf{p}_1 - \mathbf{p}_0 & \mathbf{p}_2 - \mathbf{p}_1 & (\mathbf{p}_0 + b\hat{\mathbf{n}}) - \mathbf{p}_0 \end{bmatrix} \\
\begin{bmatrix} \mathbf{d}_0 & \mathbf{d}_1 & a\hat{\mathbf{m}} \end{bmatrix} &= \mathbf{B}\begin{bmatrix} \mathbf{q}_0 & \mathbf{q}_1 & b\hat{\mathbf{n}} \end{bmatrix} \\
\begin{bmatrix} d_{s,0} & d_{s,1} & 0 \\ d_{t,0} & d_{t,1} & 0 \\ 0 & 0 & a \end{bmatrix} &= \mathbf{B}\begin{bmatrix} q_{x,0} & q_{x,1} & bn_x \\ q_{y,0} & q_{y,1} & bn_y \\ q_{z,0} & q_{z,1} & bn_z \end{bmatrix} \\
\begin{bmatrix} s_1 - s_0 & s_2 - s_1 & 0 \\ t_1 - t_0 & t_2 - t_1 & 0 \\ 0 & 0 & a \end{bmatrix} &= \mathbf{B}\begin{bmatrix} x_1 - x_0 & x_2 - x_1 & bn_x \\ y_1 - y_0 & y_2 - y_1 & bn_y \\ z_1 - z_0 & z_2 - z_1 & bn_z \end{bmatrix}
\end{aligned} \tag{10}$$

Note that $\mathbf{B}$ is a major part of the solution for $\mathbf{A}$. If $\mathbf{A}$ ever needs to be calculated, the following sequence may be faster than calculating $\mathbf{A}$ directly:

1. Calculate $\mathbf{B}$.
2. $\mathbf{A}$ is affine so let $\mathbf{0} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}}$ and $\mathbf{F} = \begin{bmatrix} 1 \end{bmatrix}$.
3. Solve for $\mathbf{E}$ by substituting a pair of corresponding positional and texture points into Equation (1).

In computer graphics, fast inversions of affine matrices are possible with block matrix operations [7, 8]. If $\mathbf{B}^{-1}$ and $\mathbf{F}^{-1}$ exist, then the inverse and inverse-transpose of $\mathbf{A}$ are as follows.

$$\begin{aligned} \mathbf{A}^{-1} &= \begin{bmatrix} \mathbf{B}^{-1} & -\mathbf{B}^{-1}\mathbf{E}\mathbf{F}^{-1} \\ \mathbf{0} & \mathbf{F}^{-1} \end{bmatrix} \\ \left(\mathbf{A}^{-1}\right)^{\mathrm{T}} &= \begin{bmatrix} \left(\mathbf{B}^{-1}\right)^{\mathrm{T}} & \mathbf{0} \\ -\left(\mathbf{F}^{-1}\right)^{\mathrm{T}}\mathbf{E}^{\mathrm{T}}\left(\mathbf{B}^{-1}\right)^{\mathrm{T}} & \left(\mathbf{F}^{-1}\right)^{\mathrm{T}} \end{bmatrix} \end{aligned} \tag{11}$$

The matrix for transforming direction vectors from positional space to texture space is the upper-left 3×3 submatrix of $\left(\mathbf{A}^{-1}\right)^{\mathrm{T}}$. From Equation (11), this matrix is $\left(\mathbf{B}^{-1}\right)^{\mathrm{T}}$ and the formulation in Equation (10) leads to a simplified way to calculate it.

$$\mathbf{B}^{-1} = \mathbf{Q}\mathbf{D}^{-1} \tag{12}$$

The matrix $\mathbf{D}$ has a simple form as does its inverse.

$$\mathbf{D}^{-1} = \frac{1}{\Delta}\begin{bmatrix} d_{t,1} & -d_{s,1} & 0 \\ -d_{t,0} & d_{s,0} & 0 \\ 0 & 0 & \frac{\Delta}{a} \end{bmatrix} \qquad \text{where} \quad \Delta = d_{s,0}d_{t,1} - d_{s,1}d_{t,0} \tag{13}$$

For the solution, we substitute Equation (13) into Equation (12) and take the transpose.

$$\left(\mathbf{B}^{-1}\right)^{\mathrm{T}} = \left(\mathbf{Q}\mathbf{D}^{-1}\right)^{\mathrm{T}} = \frac{1}{\Delta}\begin{bmatrix} d_{t,1}\mathbf{q}_0{}^{\mathrm{T}} - d_{t,0}\mathbf{q}_1{}^{\mathrm{T}} \\ -d_{s,1}\mathbf{q}_0{}^{\mathrm{T}} + d_{s,0}\mathbf{q}_1{}^{\mathrm{T}} \\ \frac{b\Delta}{a}\hat{\mathbf{n}}^{\mathrm{T}} \end{bmatrix} \tag{14}$$

This equation for $\left(\mathbf{B}^{-1}\right)^{\mathrm{T}}$ yields the matrix that transforms direction vectors from positional space to texture space. This solution requires ten fewer multiplications than Equation (8). The comments at the end of Section 3.1 about dependence on the texture binding and the ratio b/a apply to this matrix.

3.3 Normal Vector Scaling for Isotropic Texture Binding

An affine transformation that scales isotropically also preserves angles [8]. Since a translation can occur in an affine transformation, the scaling applies to the difference between two points or to direction vectors. For a triangle in texture and positional spaces shown in Figure 1, the transformation between the two spaces scales isotropically in the plane of the triangle if it observes the following two properties.

$$\frac{|\mathbf{q}_0|}{|\mathbf{d}_0|} = \frac{|\mathbf{q}_1|}{|\mathbf{d}_1|} \quad \text{and} \quad a = b, \quad 0 < a, b < p \tag{15}$$

Divisions, square roots, and inverse trigonometric functions are slow to evaluate. In the interest of computational efficiency, these can be eliminated by rearranging the equations. For the first condition, we square both sides, multiply by the denominators, and rewrite the squared magnitudes as dot products.

$$\left(\mathbf{q}_0{}^{\mathrm{T}}\mathbf{q}_0\right)\left(\mathbf{d}_1{}^{\mathrm{T}}\mathbf{d}_1\right) = \left(\mathbf{q}_1{}^{\mathrm{T}}\mathbf{q}_1\right)\left(\mathbf{d}_0{}^{\mathrm{T}}\mathbf{d}_0\right) \tag{16}$$

For the second condition, we take the sine of both sides, substitute Equations (3), square both sides, multiply by denominators, and rewrite the squared magnitudes as dot products.

$$\left(\mathbf{m}^{\mathrm{T}}\mathbf{m}\right)\left(\mathbf{q}_0{}^{\mathrm{T}}\mathbf{q}_0\right)\left(\mathbf{q}_1{}^{\mathrm{T}}\mathbf{q}_1\right) = \left(\mathbf{n}^{\mathrm{T}}\mathbf{n}\right)\left(\mathbf{d}_0{}^{\mathrm{T}}\mathbf{d}_0\right)\left(\mathbf{d}_1{}^{\mathrm{T}}\mathbf{d}_1\right) \tag{17}$$

For this case of isotropic texture binding, a reasonable design choice is to make the transformation isotropic in all three dimensions. Then the following property applies.

$$\frac{b}{a} = \frac{|\mathbf{q}_0|}{|\mathbf{d}_0|} = \frac{|\mathbf{q}_1|}{|\mathbf{d}_1|} \tag{18}$$

This makes the 3D transformation angle-preserving: isotropic scaling with rotation and translation. Substitution of Equation (18) into Equation (8) or (14) yields the full transformation.

3.4 Normal Vector Scaling for Anisotropic Texture Binding

When the binding of a texture to a triangle is such that the properties in Equation (15) (or equivalently, in Equations (16) and (17)) are not observed, then the transformation is anisotropic in the plane of the polygon. In order to choose a scale factor in the direction normal to the polygon, the minimum and maximum scale factors in the plane of the polygon need to be found. The scaling properties of matrix $\mathbf{B}$ in Equation (10) can be determined by transforming a sphere in positional space to texture space. In this discussion, we work with the edge vectors or the differences between pairs of points. Consider the equation of a sphere in positional space.

$$x^2 + y^2 + z^2 = \begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{q}^{\mathrm{T}}\mathbf{q} = 1$$

$\mathbf{B}$ is the matrix that transforms edge vectors from positional space to texture space: $\mathbf{d} = \mathbf{B}\mathbf{q}$. Its inverse does the converse: $\mathbf{q} = \mathbf{B}^{-1}\mathbf{d}$. Upon substituting this into the equation for a sphere in positional space, an equation emerges in terms of edge vectors in texture space.

$$\mathbf{q}^{\mathrm{T}}\mathbf{q} = \mathbf{d}^{\mathrm{T}}\left(\mathbf{B}^{-1}\right)^{\mathrm{T}}\mathbf{B}^{-1}\mathbf{d} = \mathbf{d}^{\mathrm{T}}\mathbf{G}\mathbf{d} = 1 \qquad (19)$$

All matrices of the form $\mathbf{G} = \left(\mathbf{B}^{-1}\right)^{\mathrm{T}}\mathbf{B}^{-1}$ are symmetric and positive definite if the matrix $\mathbf{B}^{-1}$ is nonsingular [9]. The condition is true when the triangle's vertices are not collinear. Positive definite matrices have a number of important properties including the following:

1. All eigenvalues of $\mathbf{G}$ are real and positive ($l_i > 0$).
2. The eigenvectors corresponding to different eigenvalues are orthogonal to one another. This applies to all Hermitian matrices, a superset of positive definite matrices.
3. $\mathbf{d}^{\mathrm{T}}\mathbf{G}\mathbf{d} = 1$ is the equation of an ellipsoid.
4. $\mathbf{G}$ can be decomposed into $\mathbf{G} = \mathbf{V}\mathbf{L}\mathbf{V}^{\mathrm{T}}$, where $\mathbf{L}$ is the diagonal matrix with eigenvalues $l_0,\ l_1,\ l_2$ on the main diagonal and $\mathbf{V} = \begin{bmatrix} \hat{\mathbf{v}}_0 & \hat{\mathbf{v}}_1 & \hat{\mathbf{v}}_2 \end{bmatrix}$ is the orthogonal matrix with matching unit eigenvectors in the columns.
5. Each eigenvalue/eigenvector pair describes one axis of the ellipsoid, which has a half-length of $1/\sqrt{l_i}$ and points in the direction of $\hat{\mathbf{v}}_i$.

The largest eigenvalue is associated with the shortest axis, the smallest with the longest, and the mid-size with the mid-length. In other words, the largest and smallest eigenvalues place bounds on the scaling by matrix $\mathbf{B}$. From Equation (14), we consolidate rows before calculating $\mathbf{G}$.

$$\left(\mathbf{B}^{-1}\right)^{\mathrm{T}} = \frac{1}{\Delta}\begin{bmatrix} d_{t,1}\mathbf{q}_0{}^{\mathrm{T}} - d_{t,0}\mathbf{q}_1{}^{\mathrm{T}} \\ -d_{s,1}\mathbf{q}_0{}^{\mathrm{T}} + d_{s,0}\mathbf{q}_1{}^{\mathrm{T}} \\ \frac{b\Delta}{a}\hat{\mathbf{n}}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} \frac{\tilde{\mathbf{b}}_0{}^{\mathrm{T}}}{\Delta} \\ \frac{\tilde{\mathbf{b}}_1{}^{\mathrm{T}}}{\Delta} \\ \frac{b}{a}\hat{\mathbf{n}}^{\mathrm{T}} \end{bmatrix} \tag{20}$$

Note that $\tilde{\mathbf{b}}_0$ and $\tilde{\mathbf{b}}_1$ are linear combinations of $\mathbf{q}_0$ and $\mathbf{q}_1$, which are orthogonal to $\hat{\mathbf{n}}$. Therefore, $\tilde{\mathbf{b}}_0$ and $\tilde{\mathbf{b}}_1$ are orthogonal to $\hat{\mathbf{n}}$, and the dot products vanish: $\tilde{\mathbf{b}}_i{}^{\mathrm{T}}\hat{\mathbf{n}} = 0$. Also, $\hat{\mathbf{n}}$ is a unit vector: $\hat{\mathbf{n}}^{\mathrm{T}}\hat{\mathbf{n}} = 1$. This leads to a simple form for $\mathbf{G}$.

$$\mathbf{G} = \left(\mathbf{B}^{-1}\right)^{\mathrm{T}}\mathbf{B}^{-1} = \begin{bmatrix} \frac{\tilde{\mathbf{b}}_0{}^{\mathrm{T}}\tilde{\mathbf{b}}_0}{\Delta^2} & \frac{\tilde{\mathbf{b}}_0{}^{\mathrm{T}}\tilde{\mathbf{b}}_1}{\Delta^2} & 0 \\ \frac{\tilde{\mathbf{b}}_1{}^{\mathrm{T}}\tilde{\mathbf{b}}_0}{\Delta^2} & \frac{\tilde{\mathbf{b}}_1{}^{\mathrm{T}}\tilde{\mathbf{b}}_1}{\Delta^2} & 0 \\ 0 & 0 & \frac{b^2}{a^2} \end{bmatrix} = \begin{bmatrix} h & r & 0 \\ r & x & 0 \\ 0 & 0 & \frac{b^2}{a^2} \end{bmatrix} \tag{21}$$

The eigenvalues of $\mathbf{G}$ are as follows.

$$\begin{aligned} l_0 &= \frac{h+x}{2} + \sqrt{\left(\frac{h-x}{2}\right)^2 + r^2} \\ l_1 &= \frac{h+x}{2} - \sqrt{\left(\frac{h-x}{2}\right)^2 + r^2} \\ l_2 &= \frac{b^2}{a^2} \end{aligned} \tag{22}$$

Recall that we seek to choose a value of (b/a) to completely specify the matrix $\left(\mathbf{B}^{-1}\right)^{\mathrm{T}}$ in Equation (14). The eigenvector for l_2 is $\hat{\mathbf{v}}_2 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^{\mathrm{T}}$, a unit vector parallel to the u axis in texture space. The ellipsoid has an axis length of (a/b) in this direction. The eigenvectors for l_0 and l_1 have the form $\hat{\mathbf{v}}_i = \begin{bmatrix} * & * & 0 \end{bmatrix}^{\mathrm{T}}$: they must be orthogonal to $\hat{\mathbf{v}}_2$. While they lie in the *s*-*t* plane, they generally do not coincide with the s and t axes.

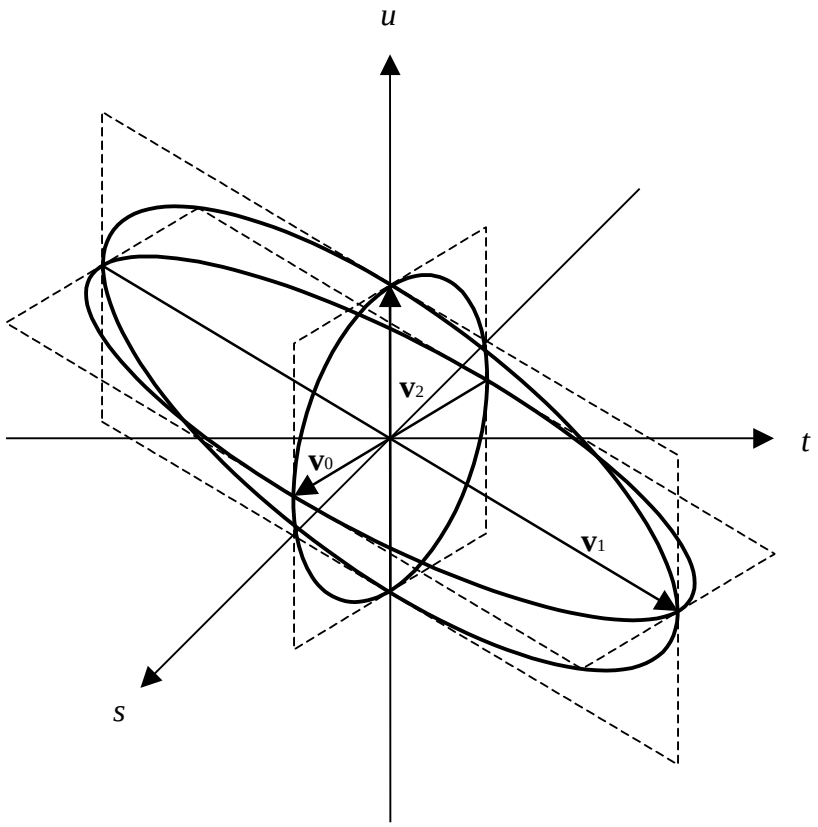

Figure 2: Three orthogonal cross sections of an ellipsoid through its axes.

Figure 2 illustrates the intersections of three orthogonal planes with the ellipsoid. Each plane is orthogonal to one eigenvector, and the span of the other two eigenvectors is the plane itself. The parallelograms enclosing the elliptical cross sections are rectangles aligned to the eigenvectors in 3D texture space. The axes have a half-length of $1/\sqrt{l_i}$. Since $l_0 \geq l_1$, the axis corresponding to l_0 is shorter than or equal to the one for l_1.

The ellipsoid's two axes in the plane of the texture image place minimum and maximum bounds on the scaling in this plane. The texture binding puts no constraint on the scaling in the direction normal to the plane. Thus, we are free to choose a scale factor in this direction. A reasonable design choice is to make the scale factor in the normal direction reside in the range of values for the plane. From Equations (22), the range is as follows.

$$\begin{aligned} \sqrt{l_1} \leq \sqrt{l_2} \leq \sqrt{l_0} \\ \sqrt{l_1} \leq \frac{b}{a} \leq \sqrt{l_0} \end{aligned} \tag{23}$$

An implementation should choose one value of b/a in this range. The obvious choices are $\sqrt{l_0}$, $\sqrt{l_1}$, or some intermediate value such as the arithmetic, geometric, or harmonic means.

1. Arithmetic mean: $\frac{1}{2}\left(\sqrt{l_0}+\sqrt{l_1}\right)$
2. Geometric mean: $\sqrt{\sqrt{l_0}\sqrt{l_1}}$
3. Harmonic mean: $\frac{2\sqrt{l_0}\sqrt{l_1}}{\sqrt{l_0}+\sqrt{l_1}}$

All these choices simplify to the simple case for isotropic texture binding in Equation (18) when $l_0 = l_1$. The selected value of b/a needs to be substituted into Equation (8) or (14) to yield the full transformation.

The choice of a value for b/a in the Range (23) makes $\hat{\mathbf{v}}_0$ the direction of the minor axis, and $\hat{\mathbf{v}}_1$ the direction of the major axis. The interpretation of the scale factor for the normal direction is straightforward for the two boundaries of the range. The lower boundary means that the 3D texture space is stretched in the direction of the major axis relative to the minor axis and normal directions, which share the same scale factor. The upper boundary means that the 3D texture space is compressed in the direction of the minor axis relative to the major axis and normal directions, which share the same scale factor. An intermediate value of b/a means that the 3D texture space is stretched in the direction of the major axis and compressed in the direction of the minor axis relative to the normal direction. All the cases are correct in the sense that the solution is not unique for lack of information from the texture binding.

The scale factor in the normal direction determines the length of shadows cast by bumps. Larger values lead to shorter shadows and give the impression that the bumps are shallower. Likewise, smaller values have the effect of longer shadows and deeper bumps. Since the texture binding does not constrain the solution, a graphics API may allow the application to control the value selected.

3.5 Estimate of Normal Vector Scaling

The transformation between positional and texture spaces is calculated once per triangle. Since the calculation involves many arithmetic operations, a simple estimate of the scale factor for the normal vector may be adequate in the interest of increasing performance. Consider the following estimate.

$$\frac{b}{a} = \sqrt{\frac{l_0 + l_1}{2}} = \sqrt{\frac{h + x}{2}} \tag{24}$$

This estimate is the square root of the average of the eigenvalues. The value can be calculated without the full expense of computing the eigenvalues themselves. We need only h and x from Equation (21) with no need for calculating r and the square root in Equations (22).

For isotropic texture binding, the estimate simplifies to the desired scale factor in Equation (18).

$$\frac{b}{a} = \sqrt{\frac{l_0 + l_1}{2}} = \sqrt{l_0} = \sqrt{l_1} \qquad \text{(isotropic case)}$$

An implementation can apply this estimate without determining whether the texture binding is isotropic or anisotropic. Moreover, this value satisfies Range (23). Given the lower and upper bounds, the average obviously lies in their range, and application of square root preserves the relationships.

$$\begin{aligned} l_1 \le \frac{l_0 + l_1}{2} \le l_0 \\ \sqrt{l_1} \le \sqrt{\frac{l_0 + l_1}{2}} \le \sqrt{l_0} \end{aligned} \tag{25}$$

As the ellipse becomes very elongated, the estimate approaches ~70.7% of the upper bound.

$$\lim_{l_0 \to \infty} \left\{ \sqrt{\frac{l_0 + l_1}{2}} - \sqrt{\frac{l_0}{2}} \right\} = 0$$

Hence, the estimate satisfies the following properties with $l_0 \ge l_1$.

$$\sqrt{l_1} \le \sqrt{\frac{l_0 + l_1}{2}} \le \sqrt{l_0} \qquad \text{and} \qquad \sqrt{\frac{l_0 + l_1}{2}} \ge \sqrt{\frac{l_0}{2}} \tag{26}$$

This estimate is relatively simple to compute, simplifies to the desired value for isotropic texture binding, and always lies in the desired range for the anisotropic case.

3.6 Operations Count

In real-time systems, a count of the operations is an indication of the complexity and performance of a particular feature. For transforming direction vectors from positional space to texture space, the fastest approach in this paper includes evaluation of Equations (14) and (24). For

light-dependent texture mapping, N lights may be active so N light direction vectors need to be transformed by the final matrix. The following table gives a summary of the calculations for this approach and a count of the elementary operations.

Table 1: Calculation Sequence and Operations Count

Calculation	Add	Mult	Div	Sqrt
$\mathbf{d}_0 = \mathbf{c}_1 - \mathbf{c}_0$	2			
$\mathbf{d}_1 = \mathbf{c}_2 - \mathbf{c}_1$	2			
$\mathbf{q}_0 = \mathbf{p}_1 - \mathbf{p}_0$	3			
$\mathbf{q}_1 = \mathbf{p}_2 - \mathbf{p}_1$	3			
$\mathbf{n} = \mathbf{q}_0 \times \mathbf{q}_1$	3	6		
$\lvert\mathbf{n}\rvert^2 = \mathbf{n}^{\mathrm{T}}\mathbf{n}$	2	3		
$\Delta = d_{s,0}d_{t,1} - d_{s,1}d_{t,0}$	1	2		
$\frac{1}{\Delta}$			1	
Δ^2		1		
$\mathbf{H} = \begin{bmatrix} \Delta & 0 & 0 \\ 0 & \Delta & 0 \\ 0 & 0 & \frac{a\lvert\mathbf{n}\rvert}{b} \end{bmatrix} \left(\mathbf{B}^{-1}\right)^{\mathrm{T}} = \begin{bmatrix} \tilde{\mathbf{b}}_0^{\mathrm{T}} \\ \tilde{\mathbf{b}}_1^{\mathrm{T}} \\ \mathbf{n}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} d_{t,1}\mathbf{q}_0^{\mathrm{T}} - d_{t,0}\mathbf{q}_1^{\mathrm{T}} \\ -d_{s,1}\mathbf{q}_0^{\mathrm{T}} + d_{s,0}\mathbf{q}_1^{\mathrm{T}} \\ \mathbf{n}^{\mathrm{T}} \end{bmatrix}$	6	12		
$h\Delta^2 = \tilde{\mathbf{b}}_0^{\mathrm{T}}\tilde{\mathbf{b}}_0$	2	3		
$x\Delta^2 = \tilde{\mathbf{b}}_1^{\mathrm{T}}\tilde{\mathbf{b}}_1$	2	3		
$\frac{b}{a\lvert\mathbf{n}\rvert} = \frac{1}{\lvert\mathbf{n}\rvert}\sqrt{\frac{h+x}{2}} = \sqrt{\frac{\left(h\Delta^2\right)+\left(x\Delta^2\right)}{2\lvert\mathbf{n}\rvert^2\Delta^2}}$	1	2	1	1
$\mathbf{L}_i' = \mathbf{H}\mathbf{L}_i$	$6N$	$9N$		

$\mathbf{L}_i'' = \left(\mathbf{B}^{-1}\right)^{\mathrm{T}} \mathbf{L}_i = \begin{bmatrix} \frac{1}{\Delta} & 0 & 0 \\ 0 & \frac{1}{\Delta} & 0 \\ 0 & 0 & \frac{b}{a\lvert\mathbf{n}\rvert} \end{bmatrix} \mathbf{H}\mathbf{L}_i = \begin{bmatrix} \frac{1}{\Delta} & 0 & 0 \\ 0 & \frac{1}{\Delta} & 0 \\ 0 & 0 & \frac{b}{a\lvert\mathbf{n}\rvert} \end{bmatrix} \mathbf{L}_i'$		$3N$		
$\hat{\mathbf{L}}'' = \frac{\mathbf{L}''}{\lvert\mathbf{L}''\rvert} = \frac{\mathbf{L}''}{\sqrt{\left(\mathbf{L}''\right)^{\mathrm{T}} \mathbf{L}''}}$	$2N$	$6N$	N	N
Total per triangle	$27+8N$	$32+18N$	$2+N$	$1+N$

The sequence consists of two parts: calculation of the matrix and transformation of the light direction vectors. Table 1 shows deferral of row scaling until after transformation of direction vectors. In a scalar implementation, deferral is beneficial for less than three lights, requiring $32+18N$ multiplications. Performing row scaling during calculation of the matrix requires $41+15N$ multiplications, and this is more efficient for more than three lights.

In this table, the light direction vectors for light *i* are $\mathbf{L}_i$ in positional space and $\mathbf{L}_i''$ in texture space; normalization of $\mathbf{L}_i''$ yields $\hat{\mathbf{L}}_i''$. If the transformation preserves angles, then normalization can be simplified. The isotropic scale factor needs to be determined and inverted once for the triangle, and the result can scale each component of $\mathbf{L}_i''$. However, determining whether a transformation preserves angles and then extracting the isotropic scale factor can involve many operations. An alternative is to apply a fast algorithm for calculating reciprocal square root [10] for each transformed light vector without determining whether the transformation preserves angles. Turkowski's algorithm uses a table lookup and two iterations of a simple formula without divisions for improving accuracy.

Calculation of the scale factor for the normal direction requires a square root. However, this function can be a low-precision approximation for improved performance. The effect of a small numerical error would be a small change in the length of shadows cast by bumps.

4. Conclusion

In the absence of nonlinear texture warps, the transformation between the positional space of a triangle and texture space can be expressed as an affine matrix. Calculation of this matrix begins with the triangle's edge vectors and cross product vectors as well as their magnitudes in both spaces. These values determine whether the scaling is isotropic. In case of anisotropic scaling, an implementation needs to select a scale factor for mapping the normal vector: this paper describes some reasonable choices. Calculation of the final matrix for transforming direction vectors is

straightforward. The matrix plays an important role in view- and light-dependent texture mapping.

1. References

[1] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik, Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach, in *Proceedings of SIGGRAPH '96 Conference, Computer Graphics*, pp. 11–20, 1996.

[2] Dan Gelb, Tom Malzbender, and Kevin Wu, *Light-Dependent Texture Mapping*, Hewlett-Packard Laboratories Technical Report, HPL-98-131, July 1998.

[3] Kevin Wu, *Rational-Linear Interpolation of Texture Coordinates and Their Partial Derivatives*, Hewlett-Packard Laboratories Technical Report, HPL-98-113, June 1998.

[4] Harvey P. Greenspan and David J. Benney, *Calculus: An Introduction to Applied Mathematics*, McGraw-Hill, 1973.

[5] Ken Turkowski, Properties of Surface-Normal Transformations, in *Graphics Gems*, Andrew S. Glassner, ed., Academic Press, pp. 539–547, 1990.

[6] Thomas Kailath, *Linear Systems*, Prentice-Hall, 1980.

[7] Kevin Wu, Fast Matrix Inversion, in *Graphics Gems II*, James Arvo, ed., Academic Press, pp. 342–350, 1991.

[8] Kevin Wu, Fast Inversion of Length- and Angle-Preserving Matrices, in *Graphics Gems IV*, Paul S. Heckbert, ed., AP Professional, pp. 199–206, 1994.

[9] Gilbert Strang, *Linear Algebra and Its Applications*, third ed., Harcourt Brace and Company, 1988.

[10] Ken Turkowski, Computing the Inverse Square Root, in *Graphics Gems V*, Alan W. Paeth, ed., AP Professional, pp. 16–21, 1995.