Computing Orthonormal Sets in 2D, 3D, and 4D

David Eberly
Geometric Tools, LLC
http://www.geometrictools.com/

Created: March 22, 2010
Last Modified: March 23, 2010

Contents

1 Introduction **2**

2 Orthonormal Sets in 2D **2**

2.1 Right-Handed and Left-Handed Orthonormal Sets 3

3 Orthonormal Sets in 3D **3**

3.1 One Vector from Two Inputs 3

3.2 Two Vectors from One Input 4

3.3 Right-Handed and Left-Handed Orthonormal Sets 5

4 Orthonormal Sets in 4D **5**

4.1 One Vector from Three Inputs 5

4.2 Two Vectors from Two Inputs 5

4.3 Three Vectors from One Input 6

4.4 Right-Handed and Left-Handed Orthonormal Sets 6

5 An Application to 3D Rotations **7**

6 An Application to 4D Rotations **8**

7 Implementations **8**

7.1 2D Pseudocode 8

7.2 3D Pseudocode 9

7.3 3D Pseudocode 11

1 Introduction

A frequently asked question in computer graphics is: Given a unit-length vector $\mathbf{N} = (x, y, z)$, how do you compute two unit-length vectors $\mathbf{U}$ and $\mathbf{V}$ so that the three vectors are mutually perpendicular? For example, $\mathbf{N}$ might be a normal vector to a plane, and you want to define a coordinate system in the plane, which requires computing $\mathbf{U}$ and $\mathbf{V}$. There are many choices for the vectors, but for numerical robustness, I have always recommended the following algorithm.

Locate the component of maximum absolute value. To illustrate, suppose that x is this value, so $|x| \geq |y|$ and $|x| \geq |z|$. Swap the first two components, changing the sign on the second, and set the third component to 0, obtaining $(y, -x, 0)$. Normalize this to obtain

$$\mathbf{U} = \frac{(y, -x, 0)}{\sqrt{x^2 + y^2}} \tag{1}$$

Now compute a cross product to obtain the other vector,

$$\mathbf{V} = \mathbf{N} \times \mathbf{U} = \frac{(xz, yz, -x^2 - y^2)}{\sqrt{x^2 + y^2}} \tag{2}$$

As you can see, a division by $\sqrt{x^2 + y^2}$ is required, so it is necessary that the divisor not be zero. In fact it is not, because of how we choose x. For a unit-length vector (x, y, z) where $|x| \geq |y|$ and $|x| \geq |y|$, it is necessary that $|x| \geq 1/\sqrt{3}$. The division by $\sqrt{x^2 + y^2}$ is therefore numerically robust–you are not dividing by a number close to zero.

In linear algebra, we refer to the set $\{\mathbf{U}, \mathbf{V}, \mathbf{N}\}$ as an *orthonormal set*. By definition, the vectors are unit length and mutually perpendicular. Let $\langle \mathbf{N} \rangle$ denote the *span* of $\mathbf{N}$. Formally, this is the set

$$\langle \mathbf{N} \rangle = \{t\mathbf{N} : t \in \mathbb{R}\} \tag{3}$$

where $\mathbb{R}$ is the set of real numbers. The span is the line that contains the origin $\mathbf{0}$ with direction $\mathbf{N}$. We may define the span of any number of vectors. For example, the span of $\mathbf{U}$ and $\mathbf{V}$ is

$$\langle \mathbf{U}, \mathbf{V} \rangle = \{s\mathbf{U} + t\mathbf{V} : s \in \mathbb{R}, t \in \mathbb{R}\} \tag{4}$$

This is the plane that contains the origin and has unit-length normal $\mathbf{N}$; that is, any vector in $\langle \mathbf{U}, \mathbf{V} \rangle$ is perpendicular to $\mathbf{N}$. The span of $\mathbf{U}$ and $\mathbf{V}$ is said to be the *orthogonal complement* of the span of $\mathbf{N}$. Equivalently, the span of $\mathbf{N}$ is said to be the orthogonal complement of the span of $\mathbf{U}$ and $\mathbf{V}$. The notation for orthogonal complement is to add a superscript "perp" symbol. $\langle \mathbf{N} \rangle^\perp$ is the orthogonal complement of the span of $\mathbf{N}$ and $\langle \mathbf{U}, \mathbf{V} \rangle^\perp$ is the orthogonal complement of the span of $\mathbf{U}$ and $\mathbf{V}$. Moreover,

$$\langle \mathbf{U}, \mathbf{V} \rangle^\perp = \langle \mathbf{N} \rangle, \;\; \langle \mathbf{N} \rangle^\perp = \langle \mathbf{U}, \mathbf{V} \rangle \tag{5}$$

2 Orthonormal Sets in 2D

The ideas in the introduction specialize to two dimensions. Given a unit-length vector $\mathbf{U}_0 = (x_0, y_0)$, a unit-length vector perpendicular to it is $\mathbf{U}_1 = (x_1, y_1) = (y_1, -x_1)$. The span of each vector is a line and the two lines are perpendicular; therefore,

$$\langle \mathbf{U}_0 \rangle^\perp = \langle \mathbf{U}_1 \rangle, \;\; \langle \mathbf{U}_1 \rangle^\perp = \langle \mathbf{U}_0 \rangle \tag{6}$$

The set $\{\mathbf{U}_0, \mathbf{U}_1\}$ is an orthonormal set.

A fancy way of constructing the perpendicular vector is to use a symbolic cofactor expansion of a determinant. Define $\mathbf{E}_0 = (1, 0)$ and $\mathbf{E}_1 = (0, 1)$. The perpendicular vector is

$$\mathbf{U}_1 = \det \begin{bmatrix} \mathbf{E}_0 & \mathbf{E}_1 \\ x & y \end{bmatrix} = y\mathbf{E}_0 - x\mathbf{E}_1 = (y, -x) \tag{7}$$

Although overkill in two dimensions, the determinant concept is useful for larger dimensions.

2.1 Right-Handed and Left-Handed Orthonormal Sets

The set $\{\mathbf{U}_0, \mathbf{U}_1\}$ is a *left-handed orthonormal set*. The vectors are unit length, perpendicular, and the matrix $M = [\mathbf{U}_0\ \mathbf{U}_1]$ whose columns are the two vectors is orthogonal with $\det(M) = -1$. To obtain a *right-handed orthonormal set*, negate the last vector: $\{\mathbf{U}_0, -\mathbf{U}_1\}$.

3 Orthonormal Sets in 3D

3.1 One Vector from Two Inputs

Define $\mathbf{E}_0 = (1, 0, 0)$, $\mathbf{E}_1 = (0, 1, 0)$, and $\mathbf{E}_2 = (0, 0, 1)$. Given two vectors $\mathbf{V}_0 = (x_0, y_0, z_0)$ and $\mathbf{V}_1 = (x_1, y_1, z_1)$, the cross product may be written as a symbolic cofactor expansion of a determinant,

$$\begin{aligned} \mathbf{V}_2 &= \mathbf{V}_0 \times \mathbf{V}_1 \\ &= \det \begin{bmatrix} \mathbf{E}_0 & \mathbf{E}_1 & \mathbf{E}_2 \\ x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \end{bmatrix} \\ &= (y_0 z_1 - y_1 z_0)\mathbf{E}_0 - (x_0 z_1 - x_1 z_0)\mathbf{E}_1 + (x_0 y_1 - x_1 y_0)\mathbf{E}_2 \\ &= (y_0 z_1 - y_1 z_0, x_1 z_0 - x_0 z_1, x_0 y_1 - x_1 y_0) \end{aligned} \tag{8}$$

If either of the input vectors is the zero vector or if the input vectors are nonzero and parallel, the cross product is the zero vector. If the input vectors are unit length and perpendicular, then the cross product is guaranteed to be unit length and $\{\mathbf{V}_0, \mathbf{V}_1, \mathbf{V}_2\}$ is an orthonormal set.

If the input vectors are linearly independent (not zero and not parallel), we may compute a pair of unit-length vectors that are orthogonal using Gram-Schmidt orthonormalization,

$$\mathbf{U}_0 = \frac{\mathbf{V}_0}{|\mathbf{V}_0|}, \quad \mathbf{U}_1 = \frac{\mathbf{V}_1 - (\mathbf{U}_0 \cdot \mathbf{V}_1)\mathbf{U}_0}{|\mathbf{V}_1 - (\mathbf{U}_0 \cdot \mathbf{V}_1)\mathbf{U}_0|} \tag{9}$$

$\mathbf{U}_0$ is a unit-length vector obtained by normalizing $\mathbf{V}_0$. We project out the $\mathbf{U}_0$ component of $\mathbf{V}_1$, which produces a vector perpendicular to $\mathbf{U}_0$. This vector is normalized to produce $\mathbf{U}_1$. We may then compute the cross product to obtain another unit-length vector, $\mathbf{U}_2 = \mathbf{U}_0 \times \mathbf{U}_1$.

If we start with *two* unit-length vectors that are perpendicular, we can obtain a third vector using the cross product. And we have shown that

$$\langle \mathbf{U}_2 \rangle = \langle \mathbf{U}_0, \mathbf{U}_1 \rangle^{\perp} \tag{10}$$

3.2 Two Vectors from One Input

If we start only with *one* unit-length vector $\mathbf{U}_2$, we wish to find two unit-length vectors $\mathbf{U}_0$ and $\mathbf{U}_1$ such that $\{\mathbf{U}_0, \mathbf{U}_1, \mathbf{U}_2\}$ is an orthonormal set, in which case

$$\langle \mathbf{U}_0, \mathbf{U}_1 \rangle = \langle \mathbf{U}_2 \rangle^{\perp} \tag{11}$$

But we have already seen how to do this–in the introduction section. Let us be slightly more formal and use the symbolic determinant idea. This idea allows us to generalize to four dimensions.

Let $\mathbf{U}_2 = (x, y, z)$ be a unit-length vector. Suppose that x has the largest absolute value of the three components. We may construct a determinant whose last row is one of the basis vectors $\mathbf{E}_i$ that does not have a zero in its first component (the one corresponding to the location of x). Let us choose $(0, 0, 1)$ as this vector; then

$$\det \begin{bmatrix} \mathbf{E}_0 & \mathbf{E}_1 & \mathbf{E}_2 \\ x & y & z \\ 0 & 0 & 1 \end{bmatrix} = y\mathbf{E}_0 - x\mathbf{E}_1 + 0\mathbf{E}_2 = (y, -x, 0) \tag{12}$$

which matches the construction in the introduction. This vector cannot be the zero vector, because we know that x has largest absolute magnitude and so cannot be zero (the initial vector is not the zero vector). Normalizing this vector, we have $\mathbf{U}_0 = (y, -x, 0)/\sqrt{x^2 + y^2}$. We may then compute $\mathbf{U}_1 = \mathbf{U}_2 \times \mathbf{U}_0$.

If y has the largest absolute magnitude, then the last row of the determinant can be either $(1, 0, 0)$ or $(0, 0, 1)$; that is, we may not choose the Euclidean basis vector with a 1 in the same component that corresponds to y. For example,

$$\det \begin{bmatrix} \mathbf{E}_0 & \mathbf{E}_1 & \mathbf{E}_2 \\ x & y & z \\ 1 & 0 & 0 \end{bmatrix} = 0\mathbf{E}_0 + z\mathbf{E}_1 - y\mathbf{E}_2 = (0, z, -y) \tag{13}$$

Once again the result cannot be the zero vector, so we may robustly compute $\mathbf{U}_0 = (0, z, -y)/\sqrt{y^2 + z^2}$ and $\mathbf{U}_1 = \mathbf{U}_2 \times \mathbf{U}_0$.

And finally, let z have the largest absolute magnitude. We may compute

$$\det \begin{bmatrix} \mathbf{E}_0 & \mathbf{E}_1 & \mathbf{E}_2 \\ x & y & z \\ 0 & 1 & 0 \end{bmatrix} = -z\mathbf{E}_0 + 0\mathbf{E}_1 + x\mathbf{E}_2 = (-z, 0, x) \tag{14}$$

which cannot be the zero vector. Thus, $\mathbf{U}_0 = (-z, 0, x)/\sqrt{x^2 + z^2}$ and $\mathbf{U}_1 = \mathbf{U}_2 \times \mathbf{U}_0$. Of course, we could have also chosen the last row to be $(1, 0, 0)$.

3.3 Right-Handed and Left-Handed Orthonormal Sets

The set $\{\mathbf{U}_0, \mathbf{U}_1, \mathbf{U}_2\}$ is a *right-handed orthonormal set*. The vectors are unit length, mutually perpendicular, and the matrix $M = [\mathbf{U}_0\ \mathbf{U}_1\ \mathbf{U}_2]$ whose columns are the three vectors is orthogonal with $\det(M) = +1$. To obtain a *left-handed orthonormal set*, negate the last vector: $\{\mathbf{U}_0, \mathbf{U}_1, -\mathbf{U}_2\}$.

4 Orthonormal Sets in 4D

This section shows how the concepts in three dimensions extend to four dimensions.

4.1 One Vector from Three Inputs

Consider three vectors $\mathbf{V}_i = (x_i, y_i, z_i, w_i)$ for $i = 0, 1, 2$. If the vectors are linearly independent, then we may compute a fourth vector that is perpendicular to the three inputs. This is a generalization of the cross product method in 3D for computing a vector that is perpendicular to two linearly independent vectors. Specifically, define $\mathbf{E}_0 = (1, 0, 0, 0)$, $\mathbf{E}_1 = (0, 1, 0, 0)$, $\mathbf{E}_2 = (0, 0, 1, 0)$, and $\mathbf{E}_3 = (0, 0, 0, 1)$. We may compute a symbolic determinant

$$\begin{aligned}
\mathbf{V}_3 &= \det\begin{bmatrix} \mathbf{E}_0 & \mathbf{E}_1 & \mathbf{E}_2 & \mathbf{E}_3 \\ x_0 & y_0 & z_0 & w_0 \\ x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & w_2 \end{bmatrix} \\
&= \det\begin{bmatrix} y_0 & z_0 & w_0 \\ y_1 & z_1 & w_1 \\ y_2 & z_2 & w_2 \end{bmatrix}\mathbf{E}_0 - \det\begin{bmatrix} x_0 & z_0 & w_0 \\ x_1 & z_1 & w_1 \\ x_2 & z_2 & w_2 \end{bmatrix}\mathbf{E}_1 + \det\begin{bmatrix} x_0 & y_0 & w_0 \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix}\mathbf{E}_2 - \det\begin{bmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{bmatrix}\mathbf{E}_3
\end{aligned} \tag{15}$$

If the input vectors $\mathbf{U}_0$, $\mathbf{U}_1$, and $\mathbf{U}_2$ are unit length and mutually perpendicular, then it is guaranteed that the output vector $\mathbf{U}_3$ is unit length and that the four vectors form an orthonormal set. If the input vectors themselves do not form an orthonormal set, we may use Gram-Schmidt orthonormalization to generate an input orthonormal set:

$$\mathbf{U}_0 = \frac{\mathbf{V}_0}{|\mathbf{V}_0|};\ \ \mathbf{U}_i = \frac{\mathbf{V}_i - \sum_{j=0}^{i-1}(\mathbf{U}_j \cdot \mathbf{V}_i)\mathbf{U}_j}{|\mathbf{V}_1 - \sum_{j=0}^{i-1}(\mathbf{U}_j \cdot \mathbf{V}_i)\mathbf{U}_j|},\ i \geq 1 \tag{16}$$

4.2 Two Vectors from Two Inputs

Let us consider two unit-length and perpendicular vectors $\mathbf{U}_i = (x_i, y_i, z_i, w_i)$ for $i = 0, 1$. If the inputs are only linearly independent, we may use Gram-Schmidt orthonormalization to obtain the unit-length and perpendicular vectors. The inputs have six associated 2×2 determinants: $x_0y_1 - x_1y_0$, $x_0z_1 - x_1z_0$, $x_0w_1 - x_1w_0$, $y_0z_1 - y_1z_0$, $y_0w_1 - y_1w_0$, and $z_0w_1 - z_1w_0$. It is guaranteed that not all of these determinants are zero when the input vectors are linearly independent. We may search for the determinant of largest absolute magnitude, which is equivalent in the three-dimensional setting when searching for the largest absolute magnitude component.

For simplicity, assume that $x_0y_1 - x_1y_0$ has the largest absolute magnitude. The handling of other cases is similar. We may construct a symbolic determinant whose last row is either $(0,0,1,0)$ or $(0,0,0,1)$. The idea is that we need a Euclidean basis vector whose components corresponding to the x and y locations are zero. We did a similar thing in three dimensions. To illustrate, let us choose $(0,0,0,1)$. The determinant is

$$\det\begin{bmatrix} \mathbf{E}_0 & \mathbf{E}_1 & \mathbf{E}_2 & \mathbf{E}_3 \\ x_0 & y_0 & z_0 & w_0 \\ x_1 & y_1 & z_1 & w_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = (y_0z_1 - y_1z_0)\mathbf{E}_0 - (x_0z_1 - x_1z_0)\mathbf{E}_1 + (x_0y_1 - x_1y_0)\mathbf{E}_2 + 0\mathbf{E}_3 \tag{17}$$

This vector cannot be the zero vector, because we know that $x_0y_1 - x_1y_0$ has the largest absolute magnitude and is not zero. Moreover, we know that this vector is perpendicular to the first two row vectors in the determinant. We can choose the unit-length vector

$$\mathbf{U}_2 = (x_2, y_2, z_2, w_2) = \frac{(y_0z_1 - y_1z_0, x_1z_0 - x_0z_1, x_0y_1 - x_1y_0, 0)}{|(y_0z_1 - y_1z_0, x_1z_0 - x_0z_1, x_0y_1 - x_1y_0, 0)|} \tag{18}$$

Observe that (x_2, y_2, z_2) is the normalized cross product of (x_0, y_0, z_0) and (x_1, y_1, z_1), and $w_2 = 0$.

We may not compute

$$\mathbf{U}_3 = \det\begin{bmatrix} \mathbf{E}_0 & \mathbf{E}_1 & \mathbf{E}_2 & \mathbf{E}_3 \\ x_0 & y_0 & z_0 & w_0 \\ x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & 0 \end{bmatrix} \tag{19}$$

which is guaranteed to be unit length. Moreover,

$$\langle \mathbf{U}_2, \mathbf{U}_3 \rangle = \langle \mathbf{U}_0, \mathbf{U}_1 \rangle^{\perp} \tag{20}$$

That is, the span of the output vectors is the orthogonal complement of the span of the input vectors.

4.3 Three Vectors from One Input

Let $\mathbf{U}_0 = (x_0, y_0, z_0, w_0)$ be a unit-length vector. Similar to the construction in three dimensions, search for the component of largest absolute magnitude. For simplicity, assume it is x_0. The other cases are handled similarly.

Choose $\mathbf{U}_1 = (y_0, -x_0, 0, 0)/\sqrt{x_0^2 + y_0^2}$, which is not the zero vector. $\mathbf{U}_1$ is unit length and perpendicular to $\mathbf{U}_1$. Now apply the construction of the previous section to obtain $\mathbf{U}_2$ and $\mathbf{U}_3$.

4.4 Right-Handed and Left-Handed Orthonormal Sets

The set $\{\mathbf{U}_0, \mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3\}$ is a *left-handed orthonormal set*. The vectors are unit length, mutually perpendicular, and the matrix $M = [\mathbf{U}_0\mathbf{U}_1\mathbf{U}_2\mathbf{U}_3]$ whose columns are the four vectors is orthogonal with $\det(M) = -1$. To obtain a *right-handed orthonormal set*, negate the last vector: $\{\mathbf{U}_0, \mathbf{U}_1, \mathbf{U}_2 - \mathbf{U}_3\}$.

5 An Application to 3D Rotations

In three dimensions, a rotation matrix R has an axis of rotation and an angle about that axis. Let $\mathbf{U}_2$ be a unit-length direction vector for the axis and let θ be the angle of rotation. Construct the orthogonal complement of $\mathbf{U}_2$ by computing vectors $\mathbf{U}_0$ and $\mathbf{U}_1$ for which $\langle \mathbf{U}_0, \mathbf{U}_1 \rangle = \langle \mathbf{U}_2 \rangle^\perp$.

The rotation does not modify the axis; that is, $R\mathbf{U}_2 = \mathbf{U}_2$. Using set notation,

$$R(\langle \mathbf{U}_2 \rangle) = \langle \mathbf{U}_2 \rangle \tag{21}$$

which says the span of $\mathbf{U}_2$ is *invariant* to the rotation.

The rotation does modify vectors in the orthogonal complement, namely,

$$R\left(x_0\mathbf{U}_0 + x_1\mathbf{U}_1\right) = (x_0\cos\theta - x_1\sin\theta)\mathbf{U}_0 + (x_0\sin\theta + x_1\cos\theta)\mathbf{U}_1 \tag{22}$$

This is a rotation in the plane perpendicular to the rotation axis. In set notation,

$$R(\langle \mathbf{U}_0, \mathbf{U}_1 \rangle) = \langle \mathbf{U}_0, \mathbf{U}_1 \rangle \tag{23}$$

which says the span of $\mathbf{U}_0$ and $\mathbf{U}_1$ is invariant to the rotation. This does not mean that individual elements of the span do not move–in fact they all do (except for the zero vector). The notation says that the plane is mapped to itself as an entire set.

A general vector $\mathbf{V}$ may be written in the coordinate system of the orthonormal vectors, $\mathbf{V} = x_0\mathbf{U}_0 + x_1\mathbf{U}_1 + x_2\mathbf{U}_2$, where $x_i = \mathbf{U}_i \cdot \mathbf{V}$. The rotation is

$$R\mathbf{V} = (x_0\cos\theta - x_1\sin\theta)\mathbf{U}_0 + (x_0\sin\theta + x_1\cos\theta)\mathbf{U}_1 + x_2\mathbf{U}_2 \tag{24}$$

Let $U = [\mathbf{U}_0\ \mathbf{U}_1\ \mathbf{U}_2]$ be a matrix whose columns are the specified vectors. The previous equation may be written in coordinate-free format as

$$R\mathbf{V} = RU\mathbf{X} = U\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}\mathbf{X} = U\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}U^{\mathrm{T}}\mathbf{V} \tag{25}$$

The rotation matrix is therefore

$$R = U\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}U^{\mathrm{T}} \tag{26}$$

If $\mathbf{U}_2 = (a, b, c)$, then $\mathbf{U}_0 = (b, -a, 0)/\sqrt{a^2 + b^2}$ and $\mathbf{U}_1 = (ac, bc, -a^2 - b^2)/\sqrt{a^2 + b^2}$. Define $\sigma = \sin\theta$ and $\gamma = \cos\theta$. Some algebra will show that the rotation matrix is

$$R = \begin{bmatrix} a^2(1-\gamma) + \gamma & ab(1-\gamma) - c\sigma & ac(1-\gamma) + b\sigma \\ ab(1-\gamma) + c\sigma & b^2(1-\gamma) + \gamma & bc(1-\gamma) - a\sigma \\ ac(1-\gamma) - b\sigma & bc(1-\gamma) + a\sigma & c^2(1-\gamma) + \gamma \end{bmatrix} \tag{27}$$

6 An Application to 4D Rotations

A three-dimensional rotation has two invariant sets, a line and a plane. A four-dimensional rotation also has two invariant sets, each a two-dimensional subspace. Each subspace has a rotation applied to it, the first by angle θ_0 and the second by angle θ_1. Let the first subspace be $\langle \mathbf{U}_0, \mathbf{U}_1 \rangle$ and the second subspace be $\langle \mathbf{U}_2, \mathbf{U}_3 \rangle$, where $\{\mathbf{U}_0, \mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3\}$ is an orthonormal set. A general vector may be written as

$$\mathbf{V} = x_0\mathbf{U}_0 + x_1\mathbf{U}_1 + x_2\mathbf{U}_2 + x_3\mathbf{3} = U\mathbf{X} \tag{28}$$

where U is the matrix whose columns are the $\mathbf{U}_i$ vectors. Define $\gamma_i = \cos\theta_i$ and $\sigma_i = \sin\theta_i$ for $i = 0, 1$. The rotation of $\mathbf{V}$ is

$$R\mathbf{V} = (x_0\gamma_0 - x_1\sigma_0)\mathbf{U}_0 + (x_0\sigma_0 + x_1\gamma_0)\mathbf{U}_1 + (x_2\gamma_1 - x_3\sigma_1)\mathbf{U}_2 + (x_2\sigma_1 + x_3\gamma_1)\mathbf{U}_3 \tag{29}$$

Similar to the three dimensional case, the rotation matrix may be shown to be

$$R = U \begin{bmatrix} \gamma_0 & -\sigma_0 & 0 & 0 \\ \sigma_0 & \gamma_0 & 0 & 0 \\ 0 & 0 & \gamma_1 & -\sigma_1 \\ 0 & 0 & \sigma_1 & \gamma_1 \end{bmatrix} U^{\mathrm{T}} \tag{30}$$

In a numerical implementation, you specify $\mathbf{U}_0$ and $\mathbf{U}_1$ and the angles θ_0 and θ_1. The other vectors $\mathbf{U}_2$ and $\mathbf{U}_3$ may be computed using the ideas shown previously in this document.

For example, to specify a rotation in the xy-plane, choose $\mathbf{U}_0 = (1, 0, 0, 0)$, $\mathbf{U}_1 = (0, 1, 0, 0)$, and $\theta_1 = 0$. The angle θ_0 is the rotation angle in the xy-plane and $\mathbf{U}_2 = (0, 0, 1, 0)$ and $\mathbf{U}_3 = (0, 0, 0, 1)$.

Another example is to specify a rotation in xyz-space. Choose $\mathbf{U}_0 = (x, y, z, 0)$, where (x, y, z) is the 3D rotation axis. Choose $\mathbf{U}_1 = (0, 0, 0, 1)$ and $\theta_0 = 0$, so that all vectors in the span of $\mathbf{U}_0$ and $\mathbf{U}_1$ are unchanged by the rotation. The other vectors are $\mathbf{U}_2 = (y, -x, 0, 0)/\sqrt{x^2 + y^2}$ and $\mathbf{U}_3 = -(xz, yz, -x^2 - y^2)/\sqrt{x^2 + y^2}$, and the angle of rotation in their plane is θ_1.

7 Implementations

7.1 2D Pseudocode

```
float Dot (Vector2 v0, Vector2 v1)
{
    return v0[0]*v1[0] + v0[1]*v1[1];
}

void Normalize (Vector2& v)
{
    v /= sqrt(Dot(v,v));
}

// Compute a vector perpendicular to v0.
Vector2 Perp (Vector2 v0)
{
    return Vector2(v0[1], -v0[0]);
```

```
}

// Gram-Schmidt orthonormalization.  On input, {v0,v1} is a linearly
// independent set.  On output, {u0,u1} is an orthonormal set.
//   u0 = v0/|v0|
//   u1 = (v1-(u0*v1)u0)/|v1-(u0*v1)u0|
void Orthonormalize (Vector2& v0, Vector2& v1)
{
    // Compute u0.
    Normalize(v0);

    // Compute u1.
    v1 -= Dot(v0,v1)*v0;
    Normalize(v1);
}

// Input v1 must be a unit-length vector.  The output is a right-handed
// orthonormal set {u0,u1}.
void GenerateOrthonormalSet (const Vector2& v0, Vector2& v1)
{
    v1 = -Perp(v0);
}

// Create a rotation matrix (angle is in radians).
Matrix2 CreateRotation (float angle)
{
    float cs = cos(angle), sn = sin(angle);
    Matrix2 rot;
    rot[0][0] = cs;
    rot[0][1] = -sn;
    rot[1][0] = sn;
    rot[1][1] = cs;
    return rot;
}
```

7.2 3D Pseudocode

```
float Dot (Vector3 v0, Vector2 31)
{
    return v0[0]*v1[0] + v0[1]*v1[1] + v0[2]*v1[2];
}

void Normalize (Vector3& v)
{
    v /= sqrt(Dot(v,v));
}

// Compute a vector perpendicular to {v0,v1}.
Vector3 Cross (Vector3 v0, Vector3 v1)
{
    return Vector3
    (
        v0[1]*v1[2] - v0[2]*v1[1],
        v0[2]*v1[0] - v0[0]*v1[2],
        v0[0]*v1[1] - v0[1]*v1[0]
    );
}

// Gram-Schmidt orthonormalization.  On input, {v0,v1,v2} is a linearly
// independent.  On output, {u0,u1,u2} is an orthonormal set.
//   u0 = v0/|v0|
//   u1 = (v1-(u0*v1)u0)/|v1-(u0*v1)u0|
//   u2 = (v2-(u0*v2)u0-(u1*v2)u1)/|v2-(u0*v2)u0-(u1*v2)u1|
void Orthonormalize (Vector3& v0, Vector3& v1)
{
    // Compute u0.
    Normalize(v0);

    // Compute u1.
```

```
    v1 -= Dot(v0,v1)*v0;
    Normalize(v1);
}

void Orthonormalize (Vector3& v0, Vector3& v1, Vector3& v2)
{
    // Compute u0 and u1.
    Orthonormalize(v0,v1);

    // Compute u2.
    v2 -= Dot(v0,v2)*v0 + Dot(v1,v2)*v1;
    Normalize(v2);
}


// Inputs u0 and u1 must be unit-length vectors that are perpendicular.  The
// output is an orthonormal set {u0,u1,u2}.
void GenerateOrthonormalSet1 (const Vector3& u0, const Vector3& u1, Vector3& u2)
{
    u2 = Cross(u0,u1);
}

// Input u0 must be a unit-length vector.  The output is an orthonormal set
// {u0,u1,u2}.
void GenerateOrthonormalSet2 (const Vector3& u0, Vector3& u1, Float3& u2)
{
    float invLength;

    if (fabs(u0[0]) >= fabs(u0[1]))
    {
        // u0[0] or u0[2] is the largest magnitude component.
        invLength = 1.0f/sqrt(u0[0]*u0[0] + u0[2]*u0[2]);
        u1[0] = -u0[2]*invLength;
        u1[1] = 0.0f;
        u1[2] = +u0[0]*invLength;
        u2[0] = u0[1]*u1[2];
        u2[1] = u0[2]*u1[0] - u0[0]*u1[2];
        u2[2] = -u0[1]*u1[0];
    }
    else
    {
        // u0[1] or u0[2] is the largest magnitude component.
        invLength = 1.0f/sqrt(u0[1]*u0[1] + u0[2]*u0[2]);
        u1[0] = 0.0f;
        u1[1] = +u0[2]*invLength;
        u1[2] = -u0[1]*invLength;
        u2[0] = u0[1]*u1[2] - u0[2]*u1[1];
        u2[1] = -u0[0]*u1[2];
        u2[2] = u0[0]*u1[1];
    }
}

// Create a rotation matrix (axis is unit-length, angle is in radians).
Matrix3 CreateRotation (const Vector3& axis, float angle)
{
    float cs = cs(angle), sn = sin(angle);
    float sn = Sin(angle);
    float oneMinusCos = 1.0f - cs;
    float x2 = axis[0]*axis[0];
    float y2 = axis[1]*axis[1];
    float z2 = axis[2]*axis[2];
    float xym = axis[0]*axis[1]*oneMinusCos;
    float xzm = axis[0]*axis[2]*oneMinusCos;
    float yzm = axis[1]*axis[2]*oneMinusCos;
    float xSin = axis[0]*sn;
    float ySin = axis[1]*sn;
    float zSin = axis[2]*sn;

    Matrix3 rot;
    rot[0][0] = x2*oneMinusCos + cs;
    rot[0][1] = xym - zSin;
    rot[0][2] = xzm + ySin;
```

```
    rot[1][0] = xym + zSin;
    rot[1][1] = y2*oneMinusCos + cs;
    rot[1][2] = yzm - xSin;
    rot[2][0] = xzm - ySin;
    rot[2][1] = yzm + xSin;
    rot[2][2] = z2*oneMinusCos + cs;
    return rot;
}
```

7.3 3D Pseudocode

```
float Dot (Vector4 v0, Vector4 31)
{
    return v0[0]*v1[0] + v0[1]*v1[1] + v0[2]*v1[2] + v0[3]*v1[3];
}

void Normalize (Vector4& v)
{
    v /= sqrt(Dot(v,v));
}

// Compute a vector perpendicular to {v0,v1,v2}.
Vector4 GenCross (Vector4 v0, Vector4 v1, Vector4 v2)
{
    return Vector4
    (
        v0[1]*v1[2]*v2[3]+v0[2]*v1[3]*v2[1]+v0[3]*v1[1]*v2[2]-v0[3]*v1[2]*v2[1]-v0[2]*v1[1]*v2[3]-v0[1]*v1[3]*v2[2],
        v0[0]*v1[2]*v2[3]+v0[2]*v1[3]*v2[0]+v0[3]*v1[0]*v2[2]-v0[3]*v1[2]*v2[0]-v0[2]*v1[0]*v2[3]-v0[0]*v1[3]*v2[2],
        v0[0]*v1[1]*v2[3]+v0[1]*v1[3]*v2[0]+v0[3]*v1[0]*v2[1]-v0[3]*v1[1]*v2[0]-v0[1]*v1[0]*v2[3]-v0[0]*v1[3]*v2[1],
        v0[0]*v1[1]*v2[2]+v0[1]*v1[2]*v2[0]+v0[2]*v1[0]*v2[1]-v0[2]*v1[1]*v2[0]-v0[1]*v1[0]*v2[2]-v0[0]*v1[2]*v2[1]
    );
}

// Gram-Schmidt orthonormalization.  On input, {v0,v1,v2,v3} is a linearly
// independent vectors set.  On output, {u0,u1,u2,u3} is an orthonormal set.
//   u0 = v0/|v0|
//   u1 = (v1-(u0*v1)u0)/|v1-(u0*v1)u0|
//   u2 = (v2-(u0*v2)u0-(u1*v2)u1)/|v2-(u0*v2)u0-(u1*v2)u1|
//   u3 = (v3-(u0*v3)u0-(u1*v3)u1-(u2*v3)u2)/|v3-(u0*v3)u0-(u1*v3)u1-(u2*v3)u2|
void Orthonormalize (Vector4& v0, Vector4& v1)
{
    // Compute u0.
    Normalize(v0);

    // Compute u1.
    v1 -= Dot(v0,v1)*v0;
    Normalize(v1);
}

void Orthonormalize (Vector4& v0, Vector4& v1, Vector4& v2)
{
    // Compute u0 and u1.
    Orthonormalize(v0,v1);

    // Compute u2.
    v2 -= Dot(v0,v2)*v0 + Dot(v1,v2)*v1;
    Normalize(v2);
}

void Orthonormalize (Vector4& v0, Vector4& v1, Vector4& v2, Vector4& v3)
{
    // Compute u0, u1, and u2.
    Orthonormalize(v0,v1,v2);

    // Compute u3.
    v3 -= Dot(v0,v3)*v0 + Dot(v1,v3)*v1 + Dot(v2,v3)*v2;
    Normalize(v3);
}
```

```
// Inputs u0, u1, and u2 must be unit-length vectors that are mutually
// perpendicular.  The output is an orthonormal set {u0,u1,u2,u3}.
void GenerateOrthonormalSet1 (const Vector4& u0, const Vector4& u1, const Vector4& u2, Vector4& u3)
{
    u3 = -u0.GenCross(u1,u2);
}

// Inputs u0 and u1 must be unit-length vectors that are perpendicular.  The
// output is an orthonormal set {u0,u1,u2,u3}.
void GenerateOrthonormalSet2 (const Vector4& u0, const Vector4& u1, Vector4& u2, Vector4& u3)
{
    // Construct a vector u2 for which {u0,u1,u2} is orthonormal.
    float det[6] =
    {
        u0[0]*u1[1] - u1[1]*u0[0],  // x0*y1 - x1*y0
        u0[0]*u1[2] - u1[2]*u0[0],  // x0*z1 - x1*z0
        u0[0]*u1[3] - u1[3]*u0[0],  // x0*w1 - x1*w0
        u0[1]*u1[2] - u1[2]*u0[1],  // y0*z1 - y1*z0
        u0[1]*u1[3] - u1[3]*u0[1],  // y0*w1 - y1*w0
        u0[2]*u1[3] - u1[3]*u0[2]   // z0*w1 - z1*w0
    };

    int maxIndex = 0;
    float maxAbsValue = fabs(det[0]);
    for (int i = 1; i < 6; ++i)
    {
        float absValue = fabs(det[i]);
        if (absValue > maxAbsValue)
        {
            maxIndex = i;
            maxAbsValue = absValue;
        }
    }

    if (maxIndex == 0)
    {
        // Last row is (0,0,1,0).
        u2 = Vector4(-det[4], +det[2], 0.0f, -det[0]);
    }
    else if (maxIndex <= 2)
    {
        // Last row is (0,1,0,0).
        u2 = Vector4(+det[5], 0.0f, -det[2], +det[1]);
    }
    else
    {
        // Last row is (1,0,0,0).
        u2 = Vector4(0.0f, -det[5], +det[4], -det[3]);
    }
    u2.Normalize();

    GenerateOrthonormalSet1(u0,u1,u2,u3);
}

// Input u0 must be a unit-length vector.  The output is an orthonormal set
// {u0,u1,u2,u3}.
void GenerateOrthonormalSet3 (const Vector4& u0, Vector4& u1, Vector4& u2, Vector4& u3)
{
    // Construct a vector u1 for which {u0,u1} is orthonormal.
    int maxIndex = 0;
    float maxAbsValue = fabs(u0[0]);
    for (int i = 1; i < 4; ++i)
    {
        float absValue = fabs(u0[i]);
        if (absValue > maxAbsValue)
        {
            maxIndex = i;
            maxAbsValue = absValue;
        }
    }

    float invLength;
```

```
    if (maxIndex < 2)
    {
        // u0[0] or u0[1] is the largest magnitude component.
        invLength = 1.0f/sqrt(u0[0]*u0[0] + u0[1]*u0[1]);
        u1[0] = -u0[1]*invLength;
        u1[1] = +u0[0]*invLength;
        u1[2] = 0.0f;
        u1[3] = 0.0f;
    }
    else
    {
        // u0[2] or u0[3] is the largest magnitude component.
        invLength = 1.0f/sqrt(u0[2]*u0[2] + u0[3]*u0[3]);
        u1[0] = 0.0f;
        u1[1] = 0.0f;
        u1[2] = +u0[3]*invLength;
        u1[3] = -u0[2]*invLength;
    }

    GenerateOrthonormalSet2(u0,u1,u2,u3);
}

// Create a rotation matrix (input vectors are unit-length and perpendicular,
// angle01 is in radians and is rotation angle for plane of input vectors,
// angle23 is in radians and is rotation angle for plane of orthogonal
// complement of input vectors).
Matrix4 CreateRotation (const Vector4& u0, const Vector4& u1, float angle01, float angle23)
{
    Vector4 u2, u3;
    GenerateOrthonormalSet2(u0,u1,u2,u3);

    float cs0 = cos(angle01), sn0 = sin(angle01);
    float cs1 = cos(angle23), sn1 = sin(angle23);
    Matrix4 U(u0, u1, u2, u3);  // columns of U are the input vectors
    Matrix4 M
    (
        cs0,  -sn0, 0.0f, 0.0f,
        sn0,  cs0,  0.0f, 0.0f,
        0.0f, 0.0f, cs1,  -sn1,
        0.0f, 0.0f, sn1,  cs1
    );
    Matrix4 rot = U*M*Transpose(U);
    return rot;
}
```