

Linear Algebra Review Session Notes

Marco da Silva
dasilva@mit.edu

September 20, 2006

Linear Algebra in Six Easy Pieces

These notes consist of several computer graphics related problems and their solutions using techniques from linear algebra. The problems were worked out on the whiteboard in the review session and summarized here. These notes assume previous exposure to linear algebra and vector calculus and are meant more to jar the memory than to introduce these concepts. Here are some useful references: [1] and [2].

Vectors

Vectors are everywhere in graphics. We use them to represent light rays bouncing around a room, surface normals, and much more. Two operations you'll use all the time are the *dot* product and the *cross* product. Let's illustrate these operations by solving a couple of graphics related problems.

Generating perpendicular vectors

Suppose you have a unit vector, $u \in \mathbb{R}^3$, and you want to find two more vectors, v , and w to build an orthonormal basis for the space. You have an infinite number of choices for v as any vector in the plane normal to u works. How do we find just one?

If you remember, the cross product of two vectors gives you a third vector that is perpendicular to both. If $a \times b = c$, then $c \perp a$ and $c \perp b$. The only caveat is that if $a \parallel b$, then c is the zero vector. In our case, we have our a .

It's just u . If we can just find a good b we could use c as our v vector. Let's just pick b to be $(1 \ 0 \ 0)$. Then,

$$v = \det\left(\begin{array}{ccc} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ u_1 & u_2 & u_3 \\ 1 & 0 & 0 \end{array}\right) = (0 \ u_3 \ -u_2)$$

This will work great unless both u_3 and u_2 are really small numbers. If we could detect this, we could use $b = (0 \ 1 \ 0)$ instead.

We can detect this using the dot product. Remember, the dot product is a binary operator that takes two vectors and produces a scalar proportional to the cosine of the angle between the two vectors. The smaller in absolute value the dot product is the closer two vectors are to being perpendicular to each other. So we calculate the dot product of u with $(1 \ 0 \ 0)$ and $(0 \ 1 \ 0)$,

$$u \cdot x = u_1 \cdot 1 + u_2 \cdot 0 + u_3 \cdot 0 = u_1$$

$$u \cdot y = u_1 \cdot 0 + u_2 \cdot 1 + u_3 \cdot 0 = u_2$$

and see which is smaller in absolute value. If $|u_1| < |u_2|$, we cross u with x , otherwise, we cross it with y .

Now we have u and v . To get w , just use the cross product again to get a vector that is perpendicular to u and v , $u \times v = w$ if you're feeling right handed or $v \times u$ if you want to be left handed. A diagram of this process is shown in figure 1.

Orthogonal Basis Vectors

You're ray tracing this wonderfully complex hierarchical model when you notice a problem at one of the leaves. The object looks like it has been skewed slightly when it shouldn't have been. You've fallen victim to some roundoff error. As you calculated the ctm for the object, small rounding errors accumulated, turning the object's orthonormal frame into some skewed mess.

One way to fix this is through a process called Gram-Schmidt orthonormalization. Let your frame be defined by the basis vectors, u, v , and w . The goal is to produce a set of orthogonal basis vectors $\hat{u}, \hat{v}, \hat{w}$. To start, set $\hat{u} = u$, and arbitrarily, decide to leave \hat{u} alone. Now, to make \hat{v} perpendicular to \hat{u} , simply subtract any component of v that's pointed in \hat{u} 's direction.

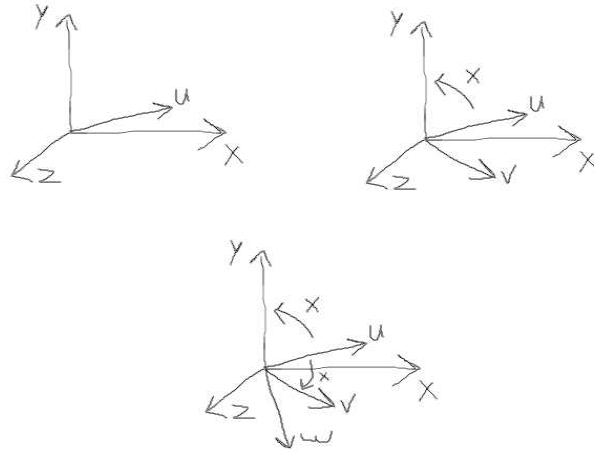


Figure 1: We cross u with the canonical y axis since u is closer to x . Then, we cross u and v to get w .

We'll call the component vector of v that's pointed in \hat{u} 's direction, $proj_{\hat{u}}(v)$. This gives us

$$\hat{v} = v - proj_{\hat{u}}(v)$$

To get \hat{w} we subtract any components pointed in \hat{u} and the new basis vector \hat{v} .

$$\hat{w} = w - proj_{\hat{u}}(w) - proj_{\hat{v}}(w)$$

This procedure is called the Gram-Schmidt process and it does produce an orthogonal set of basis vectors.

To see this though, we need to define what $proj_s(t)$ actually is. In words, it is the orthogonal projection of the vector t onto the vector s . For a picture, see figure 2. From the figure, it's clear that $proj_s(t) = \alpha s$ where α is some scalar. Since the vector $t - \alpha s$ is perpendicular to s we know that,

$$\begin{aligned} (t - \alpha s) \cdot s &= 0 \\ t \cdot s - \alpha s \cdot s &= 0 \\ \frac{t \cdot s}{s \cdot s} &= \alpha \end{aligned}$$

so $proj_s(t) = \frac{t \cdot s}{s \cdot s} s$. Using this definition of the projection operator, you should now go back and prove that $\hat{u}, \hat{v}, \hat{w}$ are all mutually orthogonal.

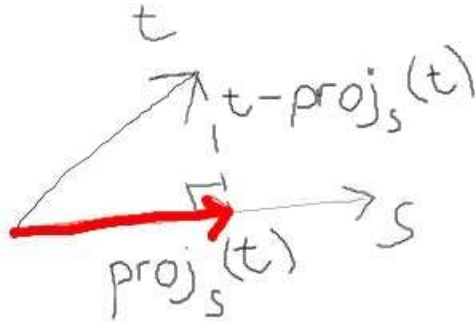


Figure 2: The projection of the vector t onto s

Calculating a Reflection Vector

Suppose a light ray ℓ bounces off a surface with normal n . How do we calculate the reflection vector, r ? Well, it's the dot product to the rescue again. Take a look at 3. We immediately get the set of equations

$$r = r_1 + r_2$$

$$\ell = r_2 - r_1$$

$$r_2 = (r \cdot n)n$$

where we assumed that n is of unit length and we used the projection operator from above to calculate r_2 . Since $r_1 = r_2 - \ell$, we see that

$$r = 2(r \cdot n)n - \ell$$

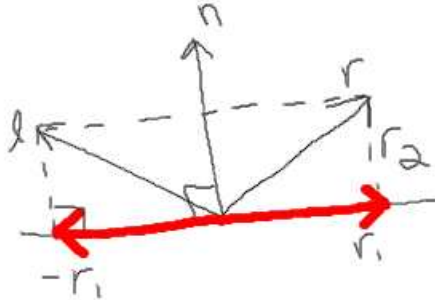


Figure 3: Calculating a reflection vector

Matrices

A chief concern of linear algebra is solving systems of linear equations of the form

$$\begin{aligned}
 a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
 a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
 &\vdots \\
 a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m
 \end{aligned}$$

By convention, mathematicians typically use the shorthand notation

$$Ax = b$$

where A is a *matrix* and x and b are *vectors*.

A matrix is just a 2D array with m rows and n columns. There is one row for each equation and one column for each variable. The entries in the array are the coefficients of the system of equations, the a_{ij} 's. A vector is really just a special case of a matrix. x and b have only one column and a number of rows. By staring at the two equations above you should be able to deduce the rule for multiplying matrices and vectors together. It gets only slightly more complicated when you multiply matrices together but let's not worry about that for now.

Hermite curve basis

The simple setup of $Ax = b$ has provided the tools for a surprising array of scientific applications. To demonstrate this, let's use a system of equations to derive the cubic Hermite curve basis.

If you recall from lecture, we can write any cubic polynomial as

$$P(t) = c_0 + c_1t + c_2t^2 + c_3t^3$$

This can be rewritten in matrix notation as

$$P(t) = \begin{pmatrix} 1 & t & t^2 & t^3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

This is called the power basis for cubic polynomials. It's a useful basis from a computation standpoint since you can skip the matrix multiply but it's fairly useless in practice because it has unintuitive controls. It's hard for a user to control the shape of a curve using these four numbers, the c_i 's.

What if we wanted to control the start and end position of the curve as well as the start and end derivative as in 4. That is, we'd like to specify that the curve starts at P_0 with derivative T_0 and ends with P_1 and derivative T_1 . This gives us the following system of equations:

$$\begin{aligned} P(0) &= c_0 = P_0 \\ P(1) &= c_0 + c_1 + c_2 + c_3 = P_1 \\ P'(0) &= c_1 = T_0 \\ P'(1) &= c_1 + 2c_2 + 3c_3 = T_1 \end{aligned}$$

which can be expressed more succinctly as

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = Bc = \begin{pmatrix} P_0 \\ P_1 \\ T_0 \\ T_1 \end{pmatrix}$$

This tells us how to convert c coordinates in to "Hermite" coordinates, but we need to convert from Hermite coordinates to c coordinates since we only have code to evaluate polynomials of the form $P(t) = c_0 + c_1t + c_2t^2 + c_3t^3$. In

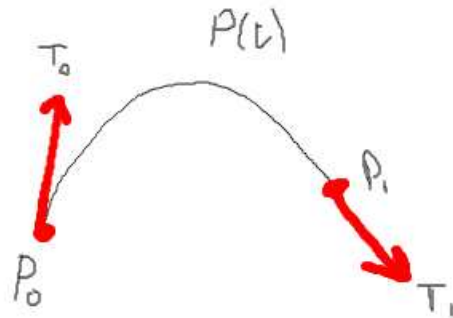


Figure 4: A Hermite curve and its control points

other words, given c coordinates we can compute Hermite coordinates, but what we need is to compute c coordinates given Hermite coordinates. To do this, we need to solve for the c 's in our system of equations. This process (which you should work out on your own) is the equivalent of “inverting” the matrix B . If you work it out, you should get:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ T_0 \\ T_1 \end{pmatrix} = M_{hermite} v = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

which gives us the recipe for converting from Hermite coordinates to power coordinates.

Inverting Matrices

If you worked through the previous problem, then you probably solved the system of equations by variable substitution which you learned a long time ago. This basic approach is known as Gauss-Jordan elimination and as you can see it's costly and error prone. There are more elaborate algorithms for solving linear systems of equations that are more efficient and robust. Fortunately, in many cases, matrix inversion becomes easy.

For an example, consider a hierarchical model. A node in the hierarchy might be transformed as

$$TRS p = \hat{p}$$

where T is a translation, R is a rotation, and S is a scale. Recall (or convince yourself), that $(TRS)^{-1} = S^{-1}R^{-1}T^{-1}$. So to invert the transformation on p we can invert each component transformation and multiply the results together. First, let's look at S^{-1} . S looks like:

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

It's easy to check that the following matrix when multiplied with S , yields the identity:

$$\begin{pmatrix} \frac{1}{s_x} & 0 & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 & 0 \\ 0 & 0 & \frac{1}{s_z} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

That was simple, so let's pick off another easy one T^{-1} . Remember that T looks like:

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

To invert this matrix, you simply translate by the opposite amount:

$$\begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This leaves us with R^{-1} . This at first seems a little trickier. R looks like:

$$\begin{pmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

It's not immediately obvious how to invert this matrix easily, but let's look at:

$$RR^T = \begin{pmatrix} u & v & w & 0_p \end{pmatrix} \begin{pmatrix} u \\ v \\ w \\ 0_p \end{pmatrix}$$

where $0_p = (0, 0, 0, 1)$ and u, v, w are vectors. Remember that the columns of any rigid rotation matrix form an orthonormal basis. In other words, $u \cdot u = 1$ but $u \cdot v = u \cdot w = 0$ and likewise for v and w . Expanding our expression for RR^T :

$$\begin{pmatrix} u \cdot u & u \cdot v & u \cdot w & u \cdot 0_p \\ v \cdot u & v \cdot v & v \cdot w & v \cdot 0_p \\ w \cdot u & w \cdot v & w \cdot w & w \cdot 0_p \\ 0_p \cdot u & 0_p \cdot v & 0_p \cdot w & 0_p \cdot 0_p \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

which is just the identity matrix. Thus, the inverse of an orthonormal matrix is just its transpose. Now, without doing any real hard work, we've inverted the transform.

Transforming Normals

Your interactive modeling app has transformed a surface but now the shading looks wrong. How come?

Well one possibility is that you have fallen prey to a common mistake. People often mistakenly transform surface normals using the same transformation matrix that transformed the surface. This is incorrect, however. To see why, recall that the surface normal n is defined as $\frac{t_1 \times t_2}{\|t_1 \times t_2\|}$. If the surface undergoes a transformation N , then the new (unnormalized) normal is $n' = Mt_1 \times Mt_2$ which in general is not equal to $M(t_1 \times t_2)$. So how do you transform the normal correctly?

If M is a non-singular transformation, then M^{-1} exists. Since $t_1 \perp n$, $n \cdot t_1 = n^T t_1 = 0$. Furthermore, since $M^{-1}M = I$, $n^T M^{-1}Mt_1 = 0$. This implies that $n' = (M^{-1})^T n$.

References

- [1] Jim Hefferon. *Linear Algebra*. <http://joshua.smcvt.edu/linearalgebra/>, 2006.

- [2] Ken Turkowski. Properties of surface-normal transformations. In Andrew Glassner, editor, *Graphics Gems*, pages 539–547. Academic Press, 1990.