IMAGE PROCESSING USING LINEAR ALGEBRA

Paul Bouthellier
Department of Mathematics and Computer Science
University of Pittsburgh-Titusville
Titusville, PA 16354
pbouthe@pitt.edu

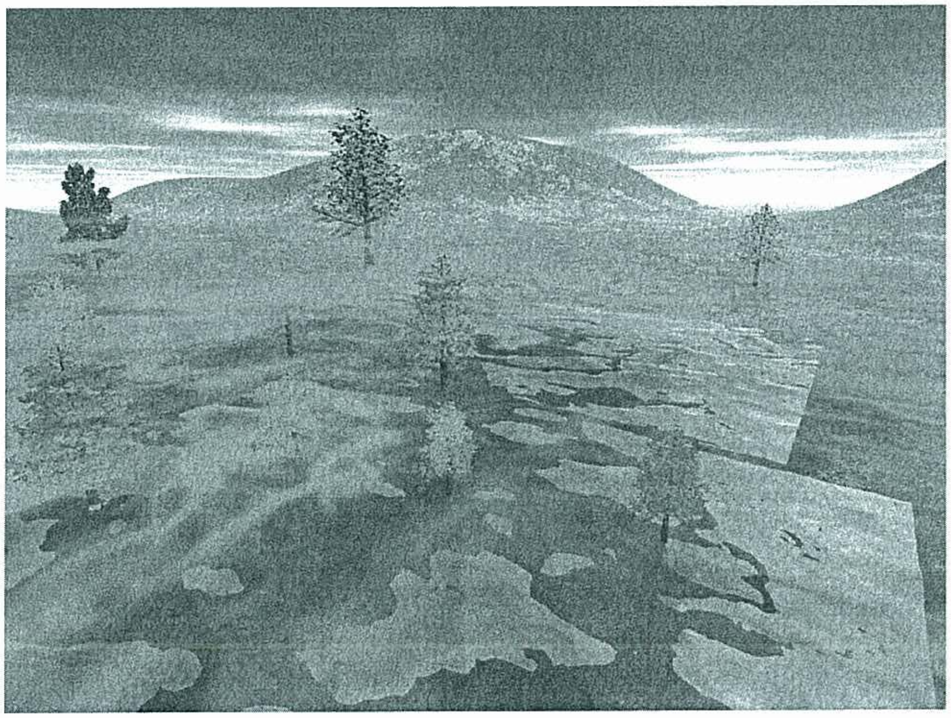


Figure 1 The Eerie Golf Course (Created Using Carrara)

Introduction:

A small subset of the linear algebra concepts that computer graphics can illustrate are: rotations, skewing, scaling, rotation matrices and quaternions, Bezier curves, reflections, dot and cross products, projections, and vector fields. Computer graphics can illustrate mathematical concepts from precalculus to graduate level courses.

In this paper we will illustrate some of the above concepts using two games. The first is a simple golf game. The second is from a children's game where, when the screen prompts the name of a given country, the player has to click on the corresponding animated flag (which then disappear). The flags rotate in R^3 and enlarge as they appear to approach the screen.

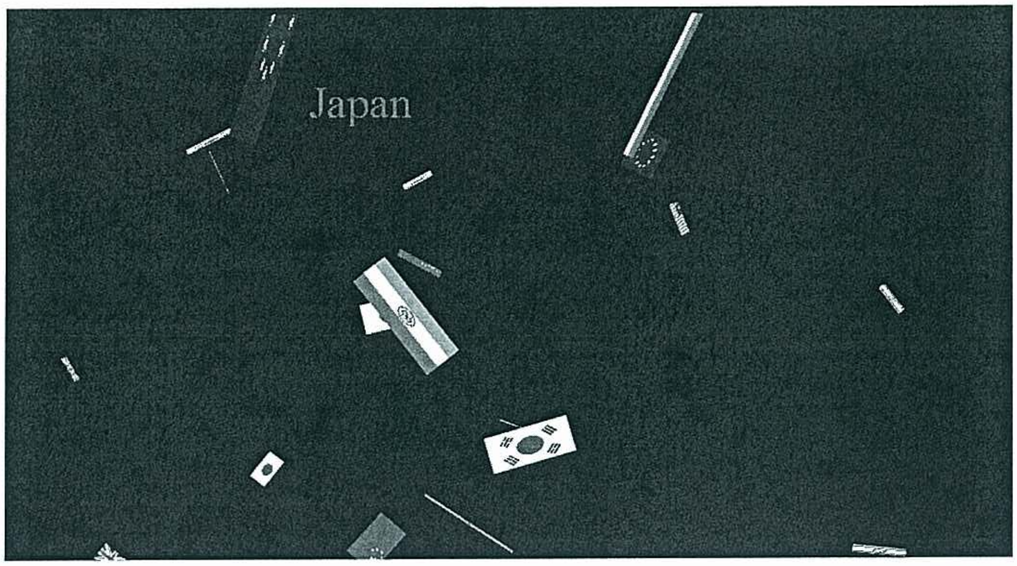


Figure 2 A Simple Flag Recognition Game (Created using Flash)

Translations and rotations are effected using the matrix

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & \Delta x \\ r_{21} & r_{22} & r_{23} & \Delta y \\ r_{31} & r_{32} & r_{33} & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

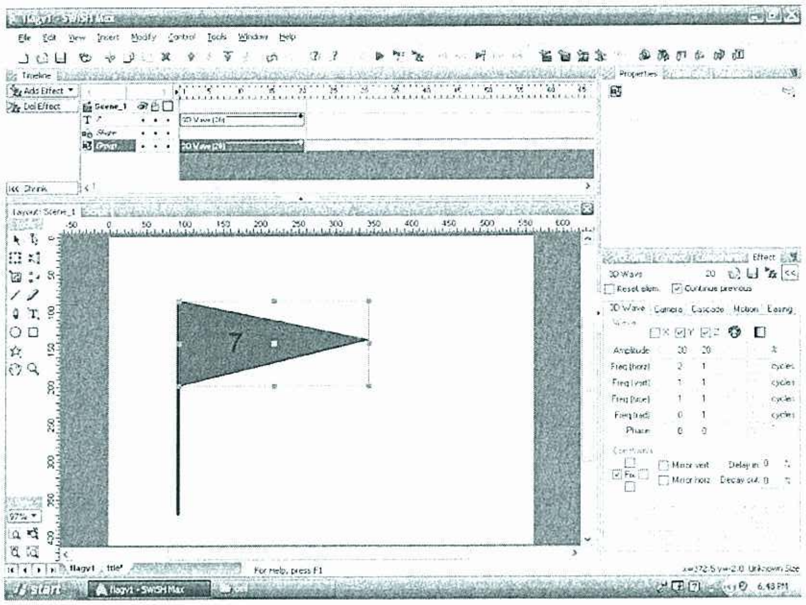
where

$$R_n(\theta) = \begin{bmatrix} c+(1-c)n_x^2 & (1-c)n_x n_y + sn_z & (1-c)n_x n_z - sn_y \\ (1-c)n_x n_y - sn_z & c+(1-c)n_y^2 & (1-c)n_y n_z + sn_x \\ (1-c)n_x n_z + sn_y & (1-c)n_y n_z - sn_x & c+(1-c)n_z^2 \end{bmatrix} \quad (2)$$

θ is the angle of rotation (right-handed system) about the unit vector n=<n_x, n_y, n_z> and c=cos(θ) and s=sin(θ).

Scaling and shearing matrices are also used to make the animations of the flags more realistic as they move towards the screen.

Another application of shearing matrices is that of creating an animated flag using shearing matrices as shown below. The flag is broken into rectangular pieces which are animated by using shearing.



3D Wave 20

[ ] Reset elem. [✓] Continue previous

3D Wave | Camera | Cascade | Motion | Easing

[ ] X [✓] Y [✓] Z

Amplitude	30	20	%
Freq (horz)	2	1	cycles
Freq (vert)	1	1	cycles
Freq (face)	1	1	cycles
Freq (rad)	0	1	cycles
Phase	0	0	

[✓] Fix [ ] Mirror vert Delay in: 0

[ ] Mirror horz Decay out: 0

x=372.5 y=-2.0 Unknown Size

Figure 3 Creating a Flag (Created with SwishMax)

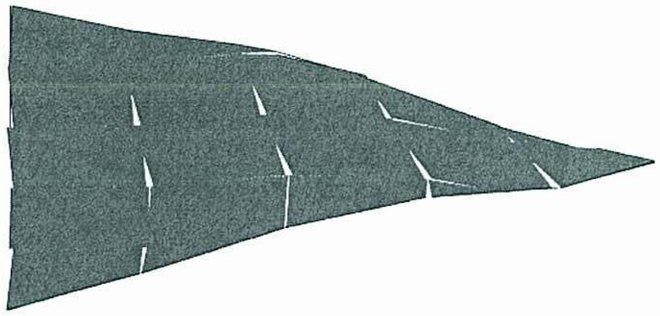


Figure 4 Skewed Rectangles (Created with SwishMax)

Rotating the individual pixels by an angle of 45 degrees creates a non 1-1 mapping between the original pixel grid and the new pixel grid-as illustrated below:

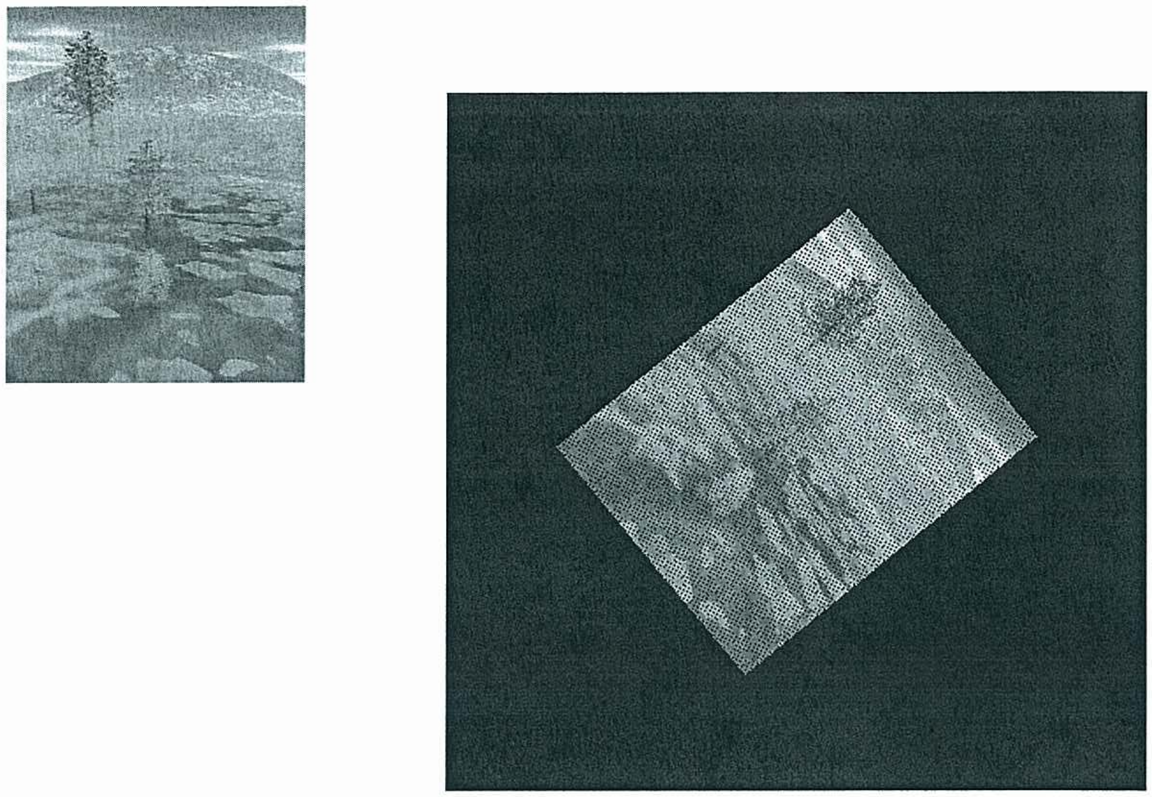


Figure 5 Rotations are not 1-1

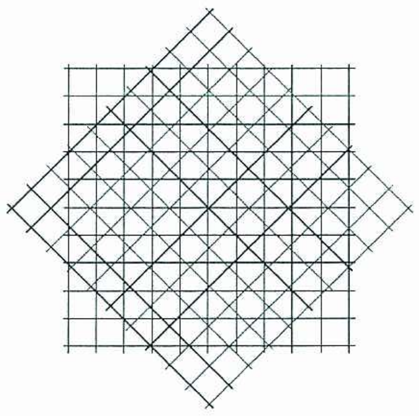
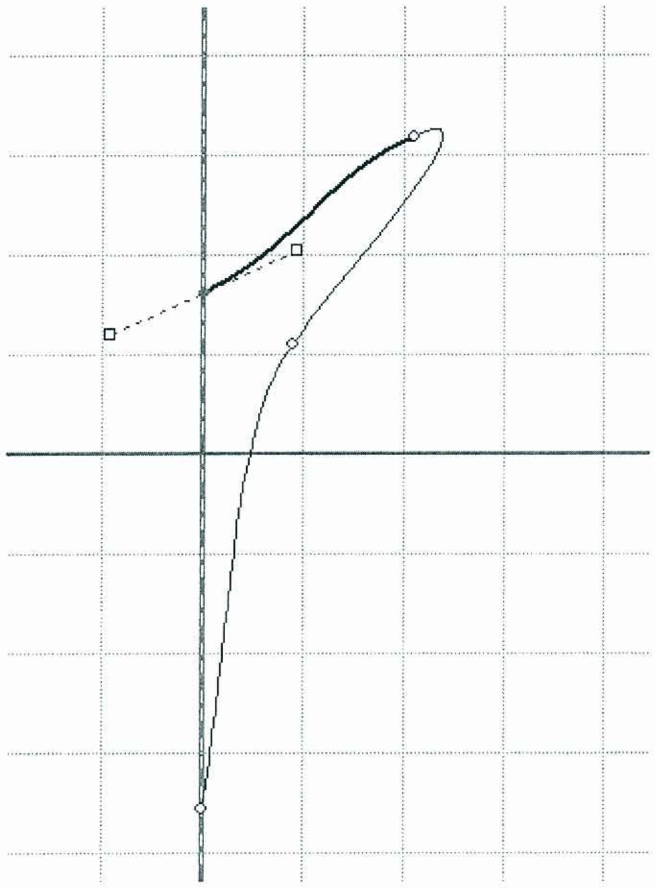


Figure 6 Showing Rotations are not 1-1

Such non 1-1 problems can be fixed by making passes over the rotated image and filling in missing pixels with the average of neighboring pixels.

Rotation matrices can also be used with Bezier curves to defined props such as golf tees.



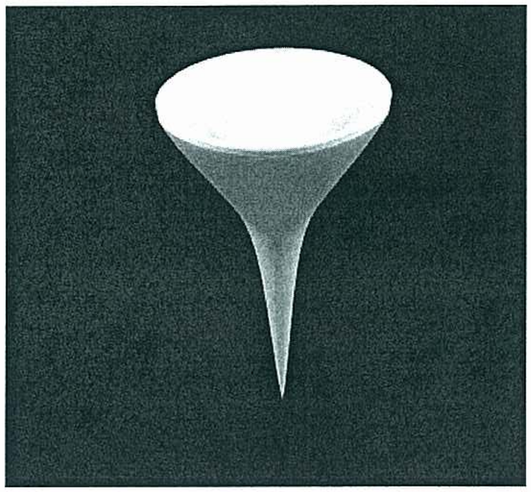


Figure 7 Creating a golf tee using rotations and Bezier curves (Swift 3d)

Rotations via Quaternions:

To rotate objects along paths and changing axes in R^3 and to create cameras one can using quaternions and spherical linear interpolation. Quaternions are defined as follows:

Define $q=q_0+iq_1+jq_2+kq_3=q_{0+}\mathbf{q}$ where

$$i^2=j^2=k^2=ijk=-1$$

$$ij=k=-ji$$

$$jk=i=-kj \qquad (3)$$

$$ki=j=-ik$$

Quaternion multiplication:

where

$$q=q_0+iq_1+jq_2+kq_3=q_0+\mathbf{q}$$

$$p=p_0+ip_1+jp_2+kp_3=p_0+\mathbf{p}$$

$$pq=p_0q_0-\mathbf{q}\bullet\mathbf{p}+p_0\mathbf{q}+q_0\mathbf{p}+\mathbf{p}\times\mathbf{q} \tag{4}$$

Given pure quaternions:

$$q=iq_1+jq_2+kq_3=\mathbf{q}$$

$$p=ip_1+jp_2+kp_3=\mathbf{p} \tag{5}$$

$$pq=-\mathbf{p}\bullet\mathbf{q}+\mathbf{p}\times\mathbf{q}$$

Rotations via Quaternions:[1], [2]

Given a unit quaternion

$$\cos(\frac{\theta}{2})+u\sin(\frac{\theta}{2}) \tag{6}$$

Where *u* in a unit vector in R^3

Defining

$$q^{-1}=q_0-iq_1-jq_2-kq_3=q_0-\mathbf{q} \tag{7}$$

and where $v\in R^3$ is represented as a pure quaternion $(0+v)$

Then

$$\mathbf{q}v\mathbf{q}^{-1} \tag{8}$$

is a right-handed rotation of *v* by an angle of θ

about the vector **q.**

Quaternion Math

Given quaternions p, q, and r:

$qv^{-1}q=(q_0+\mathbf{q})(0+\mathbf{v})(q_0+\mathbf{q})=$

$$0+(q\cdot v)q+q_0^2q+q_0(q\times v)-q_0(v\times q)$$
$$=(2q_0^2-1)v+2q\cdot v+2q_0^2(q\times v) \tag{9}$$

To perform a sequence of rotations:

$$r_2(r_1vr^{-1})r_2=(r_2r_1)v(r_1^{-1}r_2^{-1})=(r_2r_1)v(r_2r_1)^{-1} \tag{10}$$

To illustrate quaternion rotation one can construct a simple cube-computer graphics version of "Hello World." A simple cube is constructed using

- 8 points and
- 12 connectors (for the wireframe version)

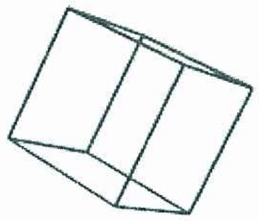


Figure 8- A Simple Cube (Constructed in Flash)

Spherical Linear Interpolation:

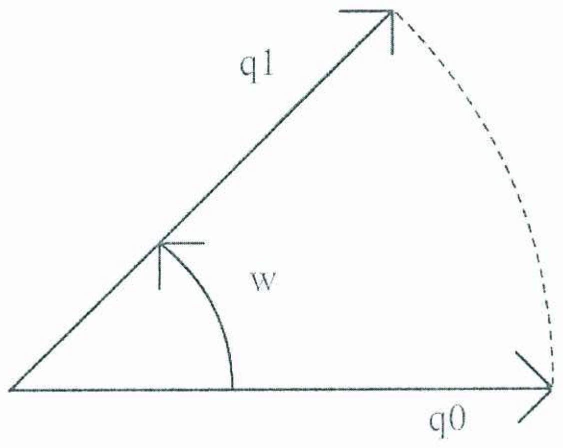


Figure 9-Sperical Linear Interpolation

$$slerp(q_0,q_1,t) = \frac{\sin((1-t)w)}{\sin(w)} q_0 + \frac{\sin(tw)}{\sin(w)} q_1 \qquad (11)$$

- Choose the signs of q_0 and q_1 such that $q_0 \cdot q_1 > 0$ to get the shortest rotational arc from q_0 and q_1.[1]
- If $\sin(w) \approx 0$ we could have numerical problems-can use simple linear interpolation.[1]

Back-Face Culling:

As at most three faces of a cube are visible to the computer screen at one time one can use back-face culling to decide whether to draw the given face on the screen. This is illustrated as follows:

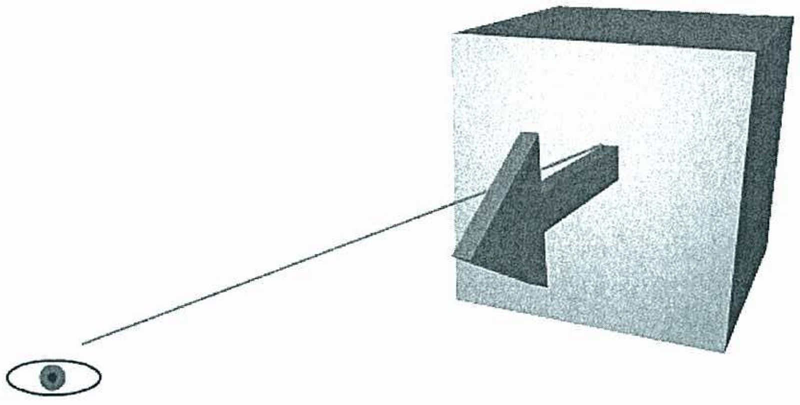


Figure 10-Back-Face Culling Using Dot Products and Normal Vectors

On a given face:

- Take three vertices-counterclockwise
- Create the normal vector

```
if ( $\overrightarrow{eye} \cdot \overrightarrow{surface\ normal} \geq 0$ )
  {render the face;}
else
  {do not render the face;}
```

Creating a Camera:

- Define a center point for the scene
- Rotate camera about this center point
- Translate objects about this center point
- Rotate objects in the opposite direction
- Translate back to the original location

$$T = \begin{bmatrix} 0 & 0 & 0 & \Delta x \\ 0 & 0 & 0 & \Delta y \\ 0 & 0 & 0 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (12)$$

Compute TRT^{-1} and translate into a rotation via a quaternion and a corresponding translation.

Other important topics I do not have room here to discuss are:

1. Projections-Mapping 3D objects onto a 2D screen and mapping textures and reflections onto surfaces.
2. Shadows-Created via various types of lights in combinations with maps.
3. Symmetry-Using symmetry often cuts down on production time taking into account the various symmetries of the models being created.
4. Interpolation-Animations are (generally) created by interpolating the various positional, rotational, and morphing aspects of objects at key frames.
5. Particles-Creating smokes, fog, clouds, and water effects.

Conclusions:

Using computer graphics it is possible to illustrate many applications of mathematics.

References:

[1] 3D Math Primer for Graphics and Game Development, Wordworth Publishing, Inc., 2002, by Dunn and Parberry.
[2] Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality, Princeton University Press, by Kuipers.