Vectors & Matrices

CSE167: Computer Graphics
Instructor: Steve Rotenberg
UCSD, Fall 2006
Project 1

- Make a program that renders a simple 3D object (like a cube). It should render several copies of the same object with different positions/rotations.
- Create a ‘Model’ class that stores an array of triangles and has a ‘Draw()’ function. The Model should have a ‘CreateBox(float,float,float)’ function that initializes it to a box. You can also make other shapes if you’d like.
- Use an object oriented approach that will allow you to re-use the Model for other projects and add new features easily as the course goes on.
- The goal of project 1 is to get familiar with the C++ compiler and OpenGL (or Java, Direct3D…)
- Due Thursday, October 5, 5:00 pm
- More details will be on the web page
class Vertex {
    Vector3 Position;
    Vector3 Color;
public:
    void Draw();
};

class Triangle {
    Vertex Vert[3];
public:
    void Draw();
};

class Model {
    int NumTris;
    Triangle *Tri;
    void Init(int num) { delete Tri; Tri=new Triangle[num]; NumTris=num; }
public:
    Model() { NumTris=0; Tri=0; }
    ~Model() { delete Tri; }
    void CreateBox(float x, float y, float z);
    void CreateTeapot();
    void Draw();
};
Software Architecture

- Object oriented
- Make objects for things that should be objects
- Avoid global data & functions
- Encapsulate information
- Provide useful interfaces
- Put different objects in different files
- Keep lower level classes as generic as possible
Vectors
Coordinate Systems

- Right handed coordinate system
Vector Arithmetic

\[ \mathbf{a} = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix} \]
\[ \mathbf{b} = \begin{bmatrix} b_x & b_y & b_z \end{bmatrix} \]
\[ \mathbf{a} + \mathbf{b} = \begin{bmatrix} a_x + b_x & a_y + b_y & a_z + b_z \end{bmatrix} \]
\[ \mathbf{a} - \mathbf{b} = \begin{bmatrix} a_x - b_x & a_y - b_y & a_z - b_z \end{bmatrix} \]
\[ -\mathbf{a} = \begin{bmatrix} -a_x & -a_y & -a_z \end{bmatrix} \]
\[ s\mathbf{a} = \begin{bmatrix} sa_x & sa_y & sa_z \end{bmatrix} \]
Vector Algebra

\((a + b) + c = a + (b + c)\)  \hspace{1cm} \text{Associativity}

\(a + b = b + a\)  \hspace{1cm} \text{Commutativity}

\(0 + a = a\)  \hspace{1cm} \text{Zero identity}

\(a + (-a) = 0\)  \hspace{1cm} \text{Additive inverse}

\((s + t)a = sa + ta\)  \hspace{1cm} \text{Distributivity}

\(s(a + b) = sa + sb\)  \hspace{1cm} \text{Distributivity}

\(1a = a\)  \hspace{1cm} \text{Multiplicative identity}
Vector Magnitude

- The magnitude (length) of a vector is:

\[ |\mathbf{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2} \]

- A vector with length=1.0 is called a *unit vector*

- We can also *normalize* a vector to make it a unit vector:

\[
\frac{\mathbf{v}}{|\mathbf{v}|}
\]
Vector Magnitude Properties

\[ sa = |s||a| \]

\[ |a + b| \leq |a| + |b| \]

Triangle inequality
Dot Product

\[ \mathbf{a} \cdot \mathbf{b} = \sum a_i b_i \]

\[ \mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y + a_z b_z \]

\[ \mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta \]
Dot Product

\[ \mathbf{a} \cdot \mathbf{b} = \sum a_i b_i \]

\[ \mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y + a_z b_z \]

\[ \mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta \]

\[ \mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b} \]

\[ \mathbf{a} \cdot \mathbf{b} = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \]
Dot Product Properties

\[(a + b) \cdot c = a \cdot b + a \cdot c\]
\[(sa) \cdot b = s(a \cdot b)\]
\[a \cdot b = b \cdot a\]

Commutativity

\[|a \cdot b| \leq |a||b|\]

Cauchy-Schwartz inequality
Example: Angle Between Vectors

- How do you find the angle $\theta$ between vectors $\mathbf{a}$ and $\mathbf{b}$?
Example: Angle Between Vectors

\[
a \cdot b = |a||b| \cos \theta
\]

\[
\cos \theta = \left( \frac{a \cdot b}{|a||b|} \right)
\]

\[
\theta = \cos^{-1} \left( \frac{a \cdot b}{|a||b|} \right)
\]
Dot Products with General Vectors

- The dot product is a scalar value that tells us something about the relationship between two vectors
  - If $a \cdot b > 0$ then $\theta < 90^\circ$
  - If $a \cdot b < 0$ then $\theta > 90^\circ$
  - If $a \cdot b = 0$ then $\theta = 90^\circ$ (or one or more of the vectors is degenerate (0,0,0))
If $|\mathbf{u}|=1.0$ then $\mathbf{a} \cdot \mathbf{u}$ is the length of the projection of $\mathbf{a}$ onto $\mathbf{u}$.
Example: Distance to Plane

- A plane is described by a point \( p \) on the plane and a unit normal \( n \). Find the distance from point \( x \) to the plane.
Example: Distance to Plane

The distance is the length of the projection of $\mathbf{x} - \mathbf{p}$ onto $\mathbf{n}$:

$$dist = (\mathbf{x} - \mathbf{p}) \cdot \mathbf{n}$$
Dot Products with Unit Vectors

\[ a \cdot b = 0 \]

\[ 0 < a \cdot b < 1 \]

\[ a \cdot b = 1 \]

\[ -1 < a \cdot b < 0 \]

\[ a \cdot b = -1 \]

\[ |a| = |b| = 1.0 \]

\[ a \cdot b = \cos(\theta) \]
Cross Product

\[ \mathbf{a} \times \mathbf{b} = \begin{vmatrix} i & j & k \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix} \]

\[ \mathbf{a} \times \mathbf{b} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix} \]
Properties of the Cross Product

\( \mathbf{a} \times \mathbf{b} \) is a vector perpendicular to both \( \mathbf{a} \) and \( \mathbf{b} \), in the direction defined by the right hand rule.

\[
|\mathbf{a} \times \mathbf{b}| = |\mathbf{a}| |\mathbf{b}| \sin \theta
\]

\[
|\mathbf{a} \times \mathbf{b}| = \text{area of parallelogram } \mathbf{ab}
\]

\[
|\mathbf{a} \times \mathbf{b}| = 0 \text{ if } \mathbf{a} \text{ and } \mathbf{b} \text{ are parallel}
\]
Example: Normal of a Triangle

- Find the unit length normal of the triangle defined by 3D points \( \textbf{a}, \textbf{b}, \) and \( \textbf{c} \)
Example: Normal of a Triangle

\[ n^* = (b - a) \times (c - a) \]

\[ n = \frac{n^*}{\|n^*\|} \]
Example: Area of a Triangle

- Find the area of the triangle defined by 3D points \(a, b, \text{ and } c\)
Example: Area of a Triangle

\[ \text{area} = \frac{1}{2} \left| (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a}) \right| \]
Example: Alignment to Target

- An object is at position $p$ with a unit length heading of $h$. We want to rotate it so that the heading is facing some target $t$. Find a unit axis $a$ and an angle $\theta$ to rotate around.
Example: Alignment to Target

\[ a = \frac{h \times (t-p)}{|h \times (t-p)|} \]

\[ \theta = \cos^{-1} \left( \frac{h \cdot (t-p)}{|(t-p)|} \right) \]
```cpp
class Vector3 {
public:
    Vector3() {x=0.0f; y=0.0f; z=0.0f;}
    Vector3(float x0,float y0,float z0) {x=x0; y=y0; z=z0;}
    void Set(float x0,float y0,float z0) {x=x0; y=y0; z=z0;}
    void Add(Vector3 &a) {x+=a.x; y+=a.y; z+=a.z;}
    void Add(Vector3 &a,Vector3 &b) {x=a.x+b.x; y=a.y+b.y; z=a.z+b.z;}
    void Subtract(Vector3 &a) {x-=a.x; y-=a.y; z-=a.z;}
    void Subtract(Vector3 &a,Vector3 &b) {x=a.x-b.x; y=a.y-b.y; z=a.z-b.z;}
    void Negate() {x=-x; y=-y; z=-z;}
    void Negate(Vector3 &a) {x=-a.x; y=-a.y; z=-a.z;}
    void Scale(float s) {x*=s; y*=s; z*=s;}
    void Scale(float s,Vector3 &a) {x=s*a.x; y=s*a.y; z=s*a.z;}
    float Dot(Vector3 &a) {return x*a.x+y*a.y+z*a.z;}
    void Cross(Vector3 &a,Vector3 &b){
        x=a.y*b.z-a.z*b.y;
        y=a.z*b.x-a.x*b.z;
        z=a.x*b.y-a.y*b.x;
    }
    float Magnitude() {return sqrtf(x*x+y*y+z*z);}
    void Normalize() {Scale(1.0f/Magnitude());}
    float x,y,z;
};
```
Matrices & Transformations
Let’s say we have a 3D model that has an array of position vectors describing its shape.

We will group all of the position vectors used to store the data in the model into a single array: \( \mathbf{v}_n \) where \( 0 \leq n \leq \text{NumVerts}-1 \).

Each vector \( \mathbf{v}_n \) has components \( v_{nx} \ v_{ny} \ v_{nz} \).
Let’s say that we want to move our 3D model from it’s current location to somewhere else…

In technical jargon, we call this a translation

We want to compute a new array of positions $v'_{n}$ representing the new location

Let’s say that vector $d$ represents the relative offset that we want to move our object by

We can simply use: $v'_{n} = v_{n} + d$

to get the new array of positions
Transformations

\[ \mathbf{v}'_n = \mathbf{v}_n + \mathbf{d} \]

- This translation represents a very simple example of an object *transformation*
- The result is that the entire object gets moved or *translated* by \( \mathbf{d} \)
- From now on, we will drop the \( _n \) subscript, and just write \[ \mathbf{v}' = \mathbf{v} + \mathbf{d} \]

remembering that in practice, this is actually a loop over several *different* \( \mathbf{v}_n \) vectors applying the *same* vector \( \mathbf{d} \) every time
Transformations

\[ \mathbf{v}' = \mathbf{v} + \mathbf{d} \]

- Always remember that this compact equation can be expanded out into

\[
\begin{bmatrix}
v'_x \\
v'_y \\
v'_z
\end{bmatrix} =
\begin{bmatrix}
v_x \\
v_y \\
v_z
\end{bmatrix} +
\begin{bmatrix}
d_x \\
d_y \\
d_z
\end{bmatrix}
\]

- Or into a system of linear equations:

\[
\begin{align*}
v'_x &= v_x + d_x \\
v'_y &= v_y + d_y \\
v'_z &= v_z + d_z
\end{align*}
\]
Rotation

- Now, let’s rotate the object in the xy plane by an angle $\theta$, as if we were spinning it around the z axis.

\[
\begin{align*}
\nu'_x &= \cos(\theta)\nu_x - \sin(\theta)\nu_y \\
\nu'_y &= \sin(\theta)\nu_x + \cos(\theta)\nu_y \\
\nu'_z &= \nu_z
\end{align*}
\]

- Note: a positive rotation will rotate the object counterclockwise when the rotation axis (z) is pointing towards the observer.
Rotation

\[
\begin{align*}
    v'_x &= \cos(\theta)v_x - \sin(\theta)v_y \\
    v'_y &= \sin(\theta)v_x + \cos(\theta)v_y \\
    v'_z &= v_z
\end{align*}
\]

- We can expand this to:
  \[
  \begin{align*}
    v'_x &= \cos(\theta)v_x - \sin(\theta)v_y + 0v_z \\
    v'_y &= \sin(\theta)v_x + \cos(\theta)v_y + 0v_z \\
    v'_z &= 0v_x + 0v_y + 1v_z
  \end{align*}
  \]

- And rewrite it as a matrix equation:
  \[
  \begin{bmatrix}
    v'_x \\
    v'_y \\
    v'_z
  \end{bmatrix} =
  \begin{bmatrix}
    \cos \theta & -\sin \theta & 0 \\
    \sin \theta & \cos \theta & 0 \\
    0 & 0 & 1
  \end{bmatrix}
  \begin{bmatrix}
    v_x \\
    v_y \\
    v_z
  \end{bmatrix}
  \]

- Or just:
  \[
  v' = M \cdot v
  \]
We can represent a z-axis rotation transformation in matrix form as:

\[
\begin{bmatrix}
v'_x \\
v'_y \\
v'_z
\end{bmatrix} =
\begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
v_x \\
v_y \\
v_z
\end{bmatrix}
\]

or more compactly as:

\[v' = M \cdot v\]

where

\[
M = R_z(\theta) =
\begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\]
We can also define rotation matrices for the x, y, and z axes:

\[
R_x(\theta) = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos \theta & -\sin \theta \\
0 & \sin \theta & \cos \theta \\
\end{bmatrix}
\]

\[
R_y(\theta) = \begin{bmatrix}
\cos \theta & 0 & \sin \theta \\
0 & 1 & 0 \\
-\sin \theta & 0 & \cos \theta \\
\end{bmatrix}
\]

\[
R_z(\theta) = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\]
Linear Transformations

- Like translation, rotation is an example of a linear transformation.
- True, the rotation contains sin()’s and cos()’s, but those ultimately just end up as constants in the actual linear equation.
- We can generalize our matrix in the previous example to be:

\[
v' = M \cdot v \quad \text{where} \quad M = \begin{bmatrix}
a_1 & b_1 & c_1 \\
a_2 & b_2 & c_2 \\
a_3 & b_3 & c_3
\end{bmatrix}
\]
Linear Equation

- A general linear equation of 1 variable is:

\[ f(v) = av + d \]

where \( a \) and \( d \) are constants

- A general linear equation of 3 variables is:

\[ f(v_x, v_y, v_z) = f(v) = av_x + bv_y + cv_z + d \]

- Note: there are no nonlinear terms like \( v_x v_y, v_x^2, \sin(v_x) \)…
System of Linear Equations

- Now let’s look at 3 linear equations of 3 variables \( v_x, v_y, \) and \( v_z \)

\[
\begin{align*}
v'_x &= a_1 v_x + b_1 v_y + c_1 v_z + d_1 \\
v'_y &= a_2 v_x + b_2 v_y + c_2 v_z + d_2 \\
v'_z &= a_3 v_x + b_3 v_y + c_3 v_z + d_3
\end{align*}
\]

- Note that all of the \( a_n, b_n, c_n, \) and \( d_n \) are constants (12 in total)
Matrix Notation

\[
\begin{align*}
\dot{v}_x &= a_1 v_x + b_1 v_y + c_1 v_z + d_1 \\
\dot{v}_y &= a_2 v_x + b_2 v_y + c_2 v_z + d_2 \\
\dot{v}_z &= a_3 v_x + b_3 v_y + c_3 v_z + d_3 \\
\end{align*}
\]

\[
\begin{bmatrix}
\dot{v}_x \\
\dot{v}_y \\
\dot{v}_z \\
\end{bmatrix}
= 
\begin{bmatrix}
a_1 & b_1 & c_1 \\
a_2 & b_2 & c_2 \\
a_3 & b_3 & c_3 \\
\end{bmatrix}
\begin{bmatrix}
v_x \\
v_y \\
v_z \\
\end{bmatrix}
+ 
\begin{bmatrix}
d_1 \\
d_2 \\
d_3 \\
\end{bmatrix}
\]

\[
v' = M \cdot v + d
\]
Translation

Let’s look at our translation transformation again:

\[
\begin{align*}
\mathbf{v}' &= \mathbf{v} + \mathbf{d} \\
\mathbf{v}'_x &= \mathbf{v}_x + d_x \\
\mathbf{v}'_y &= \mathbf{v}_y + d_y \\
\mathbf{v}'_z &= \mathbf{v}_z + d_z
\end{align*}
\]

If we really wanted to, we could rewrite our three translation equations as:

\[
\begin{align*}
\mathbf{v}'_x &= 1\mathbf{v}_x + 0\mathbf{v}_y + 0\mathbf{v}_z + d_x \\
\mathbf{v}'_y &= 0\mathbf{v}_x + 1\mathbf{v}_y + 0\mathbf{v}_z + d_y \\
\mathbf{v}'_z &= 0\mathbf{v}_x + 0\mathbf{v}_y + 1\mathbf{v}_z + d_z
\end{align*}
\]
Identity

- We can see that this is equal to a transformation by the identity matrix

\[
\begin{align*}
\nu_x' &= 1\nu_x + 0\nu_y + 0\nu_z + d_1 \\
\nu_y' &= 0\nu_x + 1\nu_y + 0\nu_z + d_2 \\
\nu_z' &= 0\nu_x + 0\nu_y + 1\nu_z + d_3
\end{align*}
\]

\[
\begin{bmatrix}
\nu_x' \\
\nu_y' \\
\nu_z'
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\nu_x \\
\nu_y \\
\nu_z
\end{bmatrix} +
\begin{bmatrix}
d_1 \\
d_2 \\
d_3
\end{bmatrix}
\]
Identity

Multiplication by the *identity matrix* does not affect the vector

\[
I = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\]

\[v = I \cdot v\]
We can apply a uniform scale to our object with the following transformation:

\[
\begin{bmatrix}
v'_x \\
v'_y \\
v'_z
\end{bmatrix} = \begin{bmatrix}
s & 0 & 0 \\
0 & s & 0 \\
0 & 0 & s
\end{bmatrix} \cdot \begin{bmatrix}
v_x \\
v_y \\
v_z
\end{bmatrix}
\]

- If \( s > 1 \), then the object will grow by a factor of \( s \) in each dimension.
- If \( 0 < s < 1 \), the object will shrink.
- If \( s < 0 \), the object will be reflected across all three dimensions, leading to an object that is ‘inside out.’
Non-Uniform Scaling

- We can also do a more general nonuniform scale, where each dimension has its own scale factor

\[
\begin{bmatrix}
 v'_x \\
v'_y \\
v'_z \\
\end{bmatrix} =
\begin{bmatrix}
 s_x & 0 & 0 \\
 0 & s_y & 0 \\
 0 & 0 & s_z \\
\end{bmatrix} \cdot
\begin{bmatrix}
 v_x \\
v_y \\
v_z \\
\end{bmatrix}
\]

which leads to the equations:

\[
\begin{align*}
v'_x &= s_x v_x \\
v'_y &= s_y v_y \\
v'_z &= s_z v_z \\
\end{align*}
\]
Multiple Transformations

- If we have a vector $\mathbf{v}$, and an x-axis rotation matrix $R_x$, we can generate a rotated vector $\mathbf{v'}$:

  \[ \mathbf{v'} = R_x(\theta) \cdot \mathbf{v} \]

- If we wanted to then rotate that vector around the y-axis, we could simply:

  \[ \mathbf{v''} = R_y(\phi) \cdot \mathbf{v'} \]
  \[ \mathbf{v''} = R_y(\phi) \cdot (R_x(\theta) \cdot \mathbf{v}) \]
Multiple Transformations

- We can extend this to the concept of applying any sequence of transformations:

\[ \mathbf{v}' = \mathbf{M}_4 \cdot (\mathbf{M}_3 \cdot (\mathbf{M}_2 \cdot (\mathbf{M}_1 \cdot \mathbf{v}))) \]

- Because matrix algebra obeys the associative law, we can regroup this as:

\[ \mathbf{v}' = (\mathbf{M}_4 \cdot \mathbf{M}_3 \cdot \mathbf{M}_2 \cdot \mathbf{M}_1) \cdot \mathbf{v} \]

- This allows us to concatenate them into a single matrix:

\[ \mathbf{M}_{total} = \mathbf{M}_4 \cdot \mathbf{M}_3 \cdot \mathbf{M}_2 \cdot \mathbf{M}_1 \]

\[ \mathbf{v}' = \mathbf{M}_{total} \cdot \mathbf{v} \]

- Note: matrices do NOT obey the commutative law, so the order of multiplications is important.
Matrix Dot Matrix

\[ \mathbf{L} = \mathbf{M} \cdot \mathbf{N} \]

\[
\begin{bmatrix}
  l_{11} & l_{12} & l_{13} \\
  l_{21} & l_{22} & l_{23} \\
  l_{31} & l_{32} & l_{33}
\end{bmatrix}
= \begin{bmatrix}
  m_{11} & m_{12} & m_{13} \\
  m_{21} & m_{22} & m_{23} \\
  m_{31} & m_{32} & m_{33}
\end{bmatrix}
\cdot \begin{bmatrix}
  n_{11} & n_{12} & n_{13} \\
  n_{21} & n_{22} & n_{23} \\
  n_{31} & n_{32} & n_{33}
\end{bmatrix}
\]

\[ l_{12} = m_{11}n_{12} + m_{12}n_{22} + m_{13}n_{32} \]
3D Linear Transformations

\[ v'_x = a_1 v_x + b_1 v_y + c_1 v_z + d_1 \]
\[ v'_y = a_2 v_x + b_2 v_y + c_2 v_z + d_2 \]
\[ v'_z = a_3 v_x + b_3 v_y + c_3 v_z + d_3 \]

\[
\begin{bmatrix}
  v'_x \\
  v'_y \\
  v'_z
\end{bmatrix} =
\begin{bmatrix}
  a_1 & b_1 & c_1 \\
  a_2 & b_2 & c_2 \\
  a_3 & b_3 & c_3
\end{bmatrix}
\begin{bmatrix}
  v_x \\
  v_y \\
  v_z
\end{bmatrix} +
\begin{bmatrix}
  d_1 \\
  d_2 \\
  d_3
\end{bmatrix}
\]

\[ v' = M \cdot v + d \]
Multiple Rotations & Scales

- We can combine a sequence of rotations and scales into a single matrix.
- For example, we can combine a \( y \)-rotation, followed by a \( z \)-rotation, then a non-uniform scale, and finally an \( x \)-rotation:

\[
M = R_x(\gamma) \cdot S(s) \cdot R_z(\beta) \cdot R_y(\alpha)
\]

\[
v' = M \cdot v
\]
Multiple Translations

- We can also take advantage of the associative property of vector addition to combine a sequence of translations.

- For example, a translation along vector $t_1$ followed by a translation along $t_2$ and finally $t_3$ can be combined:

\[
d = t_1 + t_2 + t_3
\]

\[
v' = v + d
\]
Combining Transformations

- We see that we can combine a sequence of rotations and/or scales.
- We can also combine a sequence of translations.
- But what if we want to combine translations with rotations/scales?