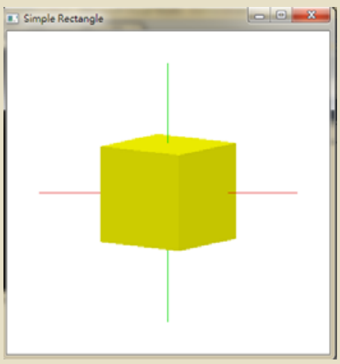
INTRODUCTION TO COMPUTER GRAPHICS

Vector, Matrix & Transformations

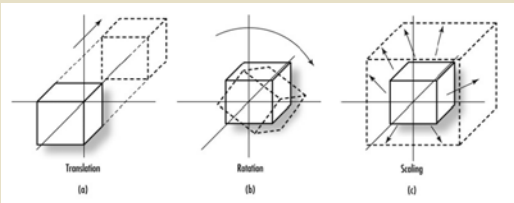
Goal

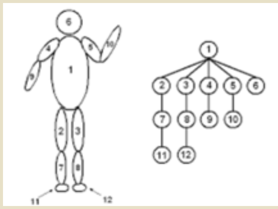
- Translation
- Rotation
- Scaling



Transformations

- Why use transformations?
  - Create object in convenient coordinates
  - Reuse basic shape multiple times
  - Hierarchical modeling
  - Virtual cameras





Translation

$$T(t_x, t_y, t_z)\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation

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

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

$$R_z(\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}$$

The call glRotate(Θ, 1, 0, 0) generates R_x as follows:

$$R_x(\theta)\begin{bmatrix}x\\ y\\ z\end{bmatrix}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\theta & -\sin\theta & 0\\ 0 & \sin\theta & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}$$

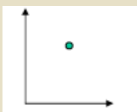
Scaling

$$S(s_x,s_y,s_z)\begin{bmatrix}x\\ y\\ z\end{bmatrix}=\begin{bmatrix}s_x & 0 & 0 & 0\\ 0 & s_y & 0 & 0\\ 0 & 0 & s_z & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}$$

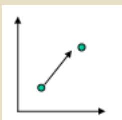
LINEAR ALGEBRA BASIC

Point v.s Vector

- A 3D point p = [x y z]
  - Represents a location with respect to some coordinate system
- A 3D vector v = [x y z]
  - Represents a displacement from a position
    - Magnitude
    - Direction
      - X-axis(1,0,0)、Y-axis(0,1,0)、Z-axis(0,0,1)



Point



vector

Unit Vector

- The Euclidean distance of u from the origin is:
  - $||u|| = \text{sqrt}(x^2+y^2+z^2)$
  - denoted by $||u||$
- if $||u|| = 1$, then u is a unit vector,
- Normalization:

$$v = \frac{u}{||u||}$$ ----- unit vector

Vector Spaces

- Consists of a set of elements, called vectors
- Two operations are defined on them
  - Addition
  - Multiplication

Vector Addition

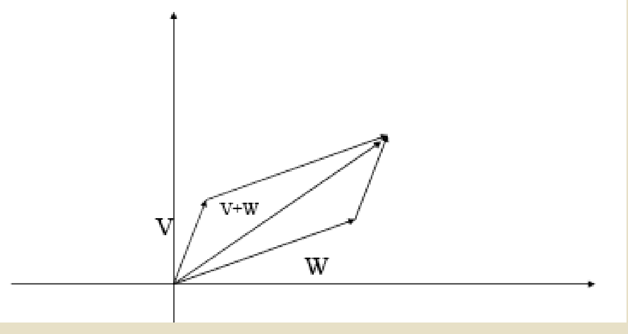
Given V = [X Y Z] and W = [A B C]

- V+W = [X+A Y+B Z+C]

- Properties of Vector addition:
  - 交換律 Commutative law: V+W=W+V
  - 結合律 Associative law : (U+V)+W = U+(V+W)
  - Additive Identity: V+0 = V
  - Additive Inverse: V+W = 0, W=-V

Vector Addition

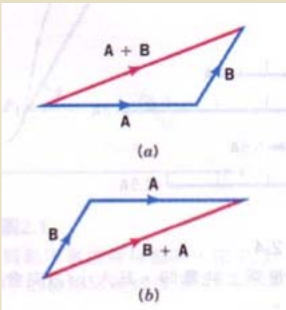
- Parallelogram Rule (平行四邊形)
  - To visualize what a vector addition is doing, here is a 2D example:

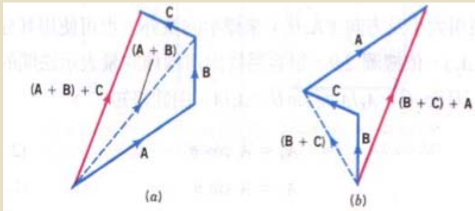


Vector Addition

Commutative law: A+B=B+A

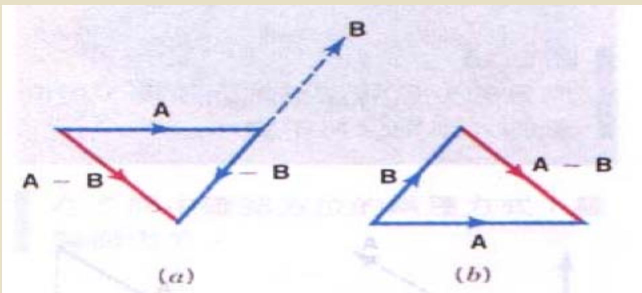
Associative law : (A+B)+C = A+(B+C)





Vector Addition

- Vector Subtraction is a special case (向量的減法可視為加法的特例)
  - A-B=A+(-B)



Vector Multiplication

Given V = [X Y Z] and a Scalar(純量) s and t

- ➤ sV = [sX sY sZ]

- Properties of Vector multiplication:
  - 結合律 Associative: (st)V = s(tV)
  - 純量分配律 Scalar Distribution: (s+t)V = sV+tV
  - 向量分配律 Vector Distribution: s(V+W) = sV+sW
  - Multiplicative Identity: 1V = V

Dot Product & Distances

Given u = $[x_1\ y_1\ z_1]$ and v = $[x_2\ y_2\ z_2]$

- 內積: $v \bullet u = x_1x_2 + y_1y_2 + z_1z_2$

- The Euclidean distance of u from the origin is:
  - sqrt($x_1^2+y_1^2+z_1^2$)
  - denoted by | | u | |
  - ➤ | | u | | = sqrt(u•u)
- The Euclidean distance between u and v is:
  - sqrt($(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2$)
  - denoted by | | u-v | |

Dot Product

Properties:

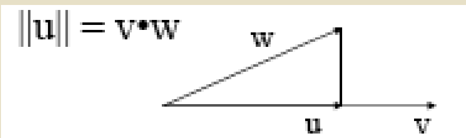
- Given a vector u, v, w and scalar s
  - The result of a dot product is a SCALAR value
  - Commutative: v•w = w•v
  - Non-degenerate: v•v=0 , only when v=0
  - Bilinear: v • (u + sw) = v•u + s(v•w)

Angles and Projection

Alternative view of the dot product:

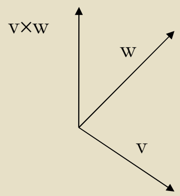
- v•w=||v|| ||w|| cos(θ)
  - where θ is the angle between v and w

- If we perpendicularly project w onto v
  - If v is a unit vector (||v|| = 1)
  - Then the projected vector u : ||u|| = v•w



Cross Product

- The cross product of v and w: v x w
  - is a VECTOR, perpendicular to the plane defined by v and w
  - ||v x w||=||v|| ||w|| sinθ
    - θ is the angle between v and w
  - vxw=-(wxv)

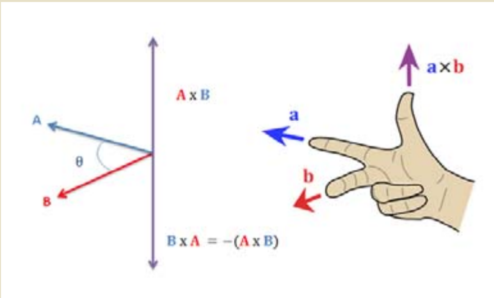


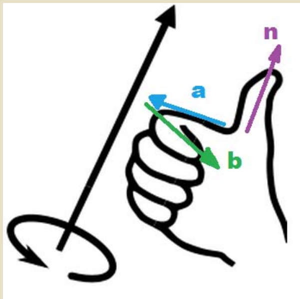
Uses of the Determinant?

- Linear Independence of columns in a matrix
- Cross Product
  - Given 2 vectors $v=[v_1\ v_2\ v_3]$, $w=[w_1\ w_2\ w_3]$, the cross product is defined to be the determinant of

$$\begin{vmatrix} i & j & k \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{vmatrix} = \begin{pmatrix} v_2w_3 - v_3w_2 \\ v_3w_1 - v_1w_3 \\ v_1w_2 - v_2w_1 \end{pmatrix}$$

Right hand rule



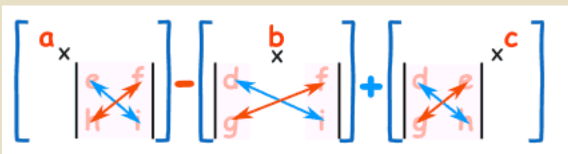


Determinant of a Matrix

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$\det A = |A| = \sum_{i=1}^{n} (-1)^{1+i} A_{1i}$$

|A| = a(ei - fh) - b(di - fg) + c(dh - eg)



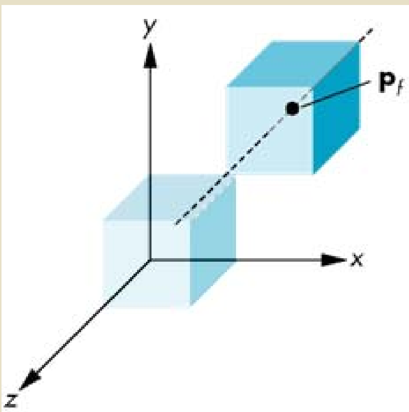
http://www.mathsisfun.com/algebra/matrix-determinant.html

$$C = \begin{bmatrix} 6 & 1 & 1 \\ 4 & -2 & 5 \\ 2 & 8 & 7 \end{bmatrix}$$

$$\begin{aligned} |\mathbf{C}| &= 6\times(-2\times7 - 5\times8) - 1\times(4\times7 - 5\times2) + 1\times(4\times8 - (-2)\times2) \\ &= 6\times(-54) - 1\times(18) + 1\times(36) \\ &= \mathbf{-306} \end{aligned}$$

Translation

- **void glTranslatef(tx , ty , tz);**



Translation

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

$$T(t_x, t_y, t_z)\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Properties of Translation

$$\boxed{T(t_x, t_y, t_z)\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}}$$

$$T(0,0,0)\,\mathbf{v} = \mathbf{v}$$

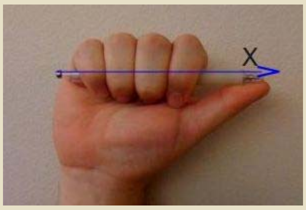
$$T(s_x, s_y, s_z)\,T(t_x, t_y, t_z)\mathbf{v} = T(s_x + t_x, s_y + t_y, s_z + t_z)\,\mathbf{v}$$

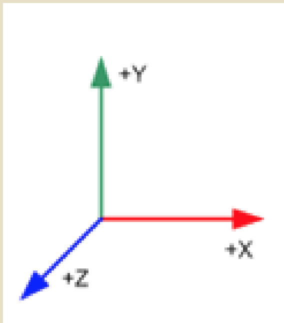
$$T(s_x, s_y, s_z)\,T(t_x, t_y, t_z)\mathbf{v} = T(t_x, t_y, t_z)\,T(s_x, s_y, s_z)\mathbf{v}$$

$$T^{-1}(t_x, t_y, t_z)\,\mathbf{v} = T(-t_x, -t_y, -t_z)\,\mathbf{v}$$

Rotation Matrix Direction

- OpenGL is right hand rule (counter clockwise)





http://www.youtube.com/watch?v=GbhTCg0FCr4

Rotations (3D)

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

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

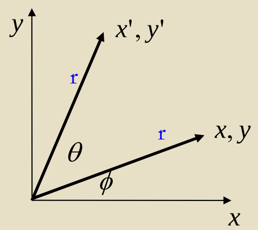
$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotations 2D

- So in matrix notation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Rotations (2D)



$$x = r\cos\phi$$
$$y = r\sin\phi$$
$$x' = r\cos(\phi+\theta)$$
$$y' = r\sin(\phi+\theta)$$

$$\cos(\phi+\theta) = \cos\phi\cos\theta - \sin\phi\sin\theta$$
$$\sin(\phi+\theta) = \cos\phi\sin\theta + \sin\phi\cos\theta$$

$$x' = (r\cos\phi)\cos\theta - (r\sin\phi)\sin\theta$$
$$y' = (r\cos\phi)\sin\theta + (r\sin\phi)\cos\theta$$

$$\boxed{\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \end{aligned}}$$

Rotations 2D

- So in matrix notation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Rotations (3D)

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

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

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Properties of Rotations

$$R_a(0) = I$$

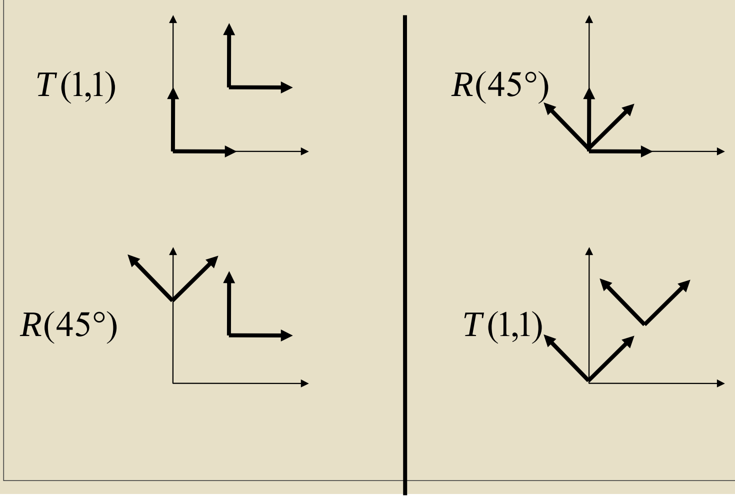
$$R_a(\theta)R_a(\phi) = R_a(\phi + \theta)$$

$$R_a(\theta)R_a(\phi) = R_a(\phi)R_a(\theta)$$

$$R_a^{-1}(\theta) = R_a(-\theta) = R_a^{T}(\theta)$$

$$R_a(\theta)R_b(\phi) \neq R_b(\phi)R_a(\theta)$$ order matters!

Combining Translation & Rotation



Combining Translation & Rotation

$$\mathbf{v}' = \mathbf{v} + T$$

$$\mathbf{v}'' = R\mathbf{v}'$$

$$\mathbf{v}'' = R(\mathbf{v} + T)$$

$$\mathbf{v}'' = R\mathbf{v} + RT$$

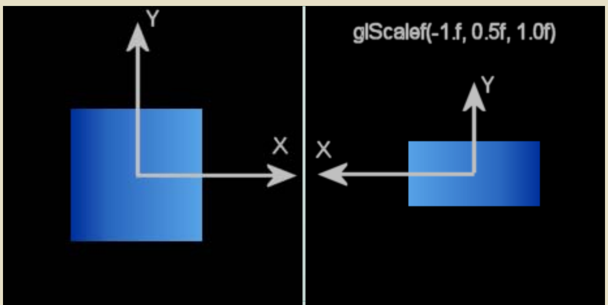
$$\mathbf{v}' = R\mathbf{v}$$

$$\mathbf{v}'' = \mathbf{v}' + T$$

$$\mathbf{v}'' = R\mathbf{v} + T$$

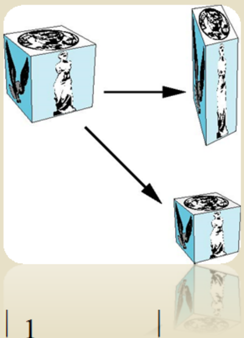
Scale

- void glScalef(x , y , z);



glScale(-1.f, 0.5f, 1.f)

Scaling



$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x\, x \\ s_y\, y \\ s_z\, z \end{bmatrix}$$

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

$$S^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & \frac{1}{s_z} \end{bmatrix}$$

Uniform scaling ***iff*** $s_x = s_y = s_z$

Inverse Transform

$$T^{-1}(t_x, t_y, t_z)\,\mathbf{v} = T(-t_x, -t_y, -t_z)\,\mathbf{v}$$

$$S^{-1}(s_x, s_y, s_z)\,\mathbf{v} = S(\frac{1}{s_x}, \frac{1}{s_y}, \frac{1}{s_z})\,\mathbf{v}$$

$$R_a^{-1}(\theta) = R_a(-\theta) = R_a^{T}(\theta)$$

OpenGL implementation

- Code review

Homogeneous Coordinates

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \text{ can be represented as } \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix} \qquad \text{e.g., } \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

where $x = \dfrac{X}{w}, \quad y = \dfrac{Y}{w}, \quad z = \dfrac{Z}{w}$

Translation & Rotation

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

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

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

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation Revisited

$$T(t_x, t_y, t_z)\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation Revisited

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

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

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The call glRotate(Θ, 1, 0, 0) generates R_x as follows:

$$R_x(\theta)\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation Revisited

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

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

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The call glRotate(Θ, 0, 1, 0) generates R_y as follows:

$$R_y(\theta)\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation Revisited

The call glRotate(Θ, 0, 0, 1) generates R_z as follows:

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

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

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\theta)\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Scaling Revisited

$$S(s_x, s_y, s_z)\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Combining Transformations

$$\mathbf{v}' = S\mathbf{v}$$
$$\mathbf{v}'' = R\mathbf{v}' = RS\mathbf{v}$$
$$\mathbf{v}''' = T\mathbf{v}'' = TR\mathbf{v}' = TRS\mathbf{v}$$
$$\mathbf{v}''' = M\mathbf{v}$$

where $M = TRS$

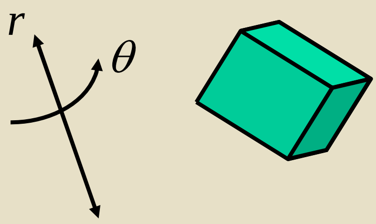
```
glLoadIdentity();
glScalef(1.0f, 2.0f, 1.0f);
glRotatef(angle, 0.0f, 0.0f, 1.0f);
glTranslatef(0.0f, 0.0f -5.0f);
```

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(N);             /* apply transformation N */
glMultMatrixf(M);             /* apply transformation M */
glMultMatrixf(L);             /* apply transformation L */
glBegin(GL_POINTS);
glVertex3f(v);                /* draw transformed vertex v */
glEnd();
```

Arbitrary rotations

Rotations about an arbitrary axis

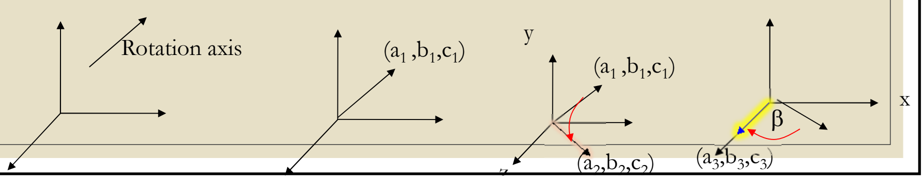
Rotate by θ around a unit axis r



Rotation about an arbitrary axis

$$rot_{axis}(\theta) = \begin{pmatrix} a' \\ b' \\ c' \end{pmatrix} = T^{-1}R_x^{-1}(\alpha)R_y^{-1}(\beta)R_z(\theta)R_y(\beta)R_x(\alpha)T\begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

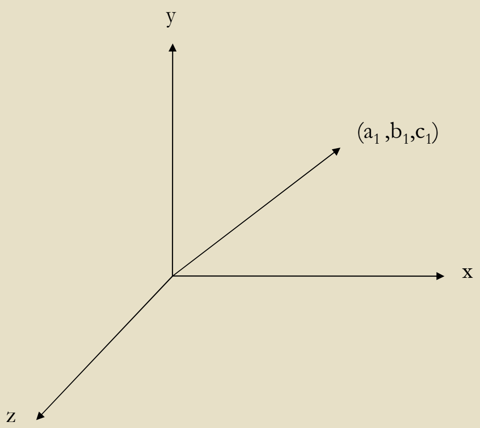
- Translate the space so that the origin of the unit vector is on the world origin
- Rotate such that the extremity of the vector now **lies in the xz plane** (x-axis rotation)
- Rotate such that the point **lies in the z-axis** (y-axis rotation)
- Perform the rotation around the z-axis
- Undo the previous transformations



Rotation about an arbitrary axis

- Step 1: Translate the space so that the origin of the unit vector is on the world origin

$$\begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} = T \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

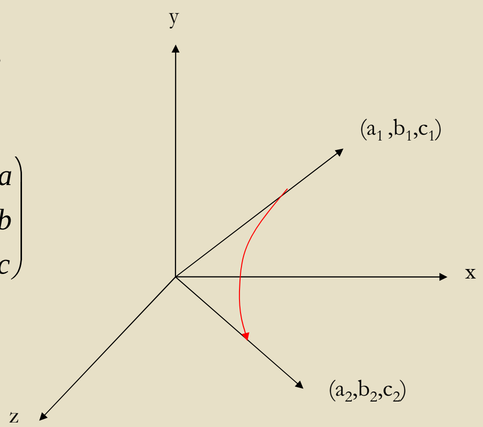


Rotation about an arbitrary axis

- Step 2

Rotate along x-axis

$$\begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} = R_x(\alpha) T \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$



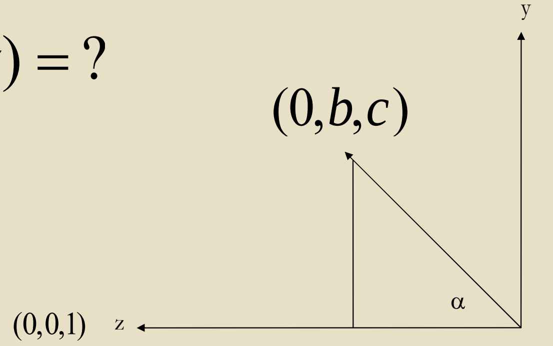
Let the vector **lies in the xz plane!**

Closer Look at Y-Z Plane

- Need to rotate α degrees around the x-axis

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

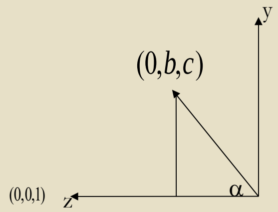
$$R_x(\alpha) = ?$$



Equations for α

|vxw||=||v|| ||w|| sinθ

v•w=||v|| ||w|| cos(θ) |



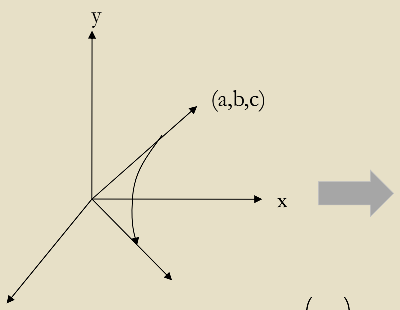
$$\sin(\alpha) = \frac{\|(0,0,1)\times(0,b,c)\|}{\|(0,0,1)\|\,\|(0,b,c)\|} = \frac{b}{sqrt(b^2+c^2)}$$

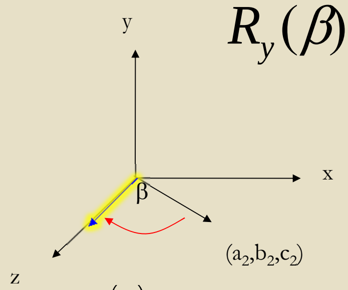
$$\cos(\alpha) = \frac{(0,0,1)\bullet(0,b,c)}{\|(0,0,1)\|\,\|(0,b,c)\|} = \frac{c}{sqrt(b^2+c^2)}$$

Rotation about the Y-axis

- Step3: Rotate along y axis

Using the same analysis as before, we need to rotate β degrees around the Y-axis



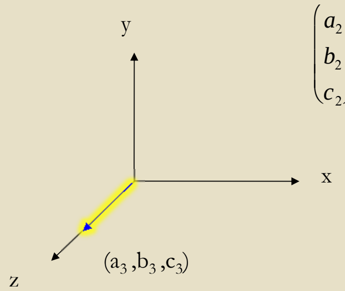


$$R_y(\beta) = ?$$

$$\begin{pmatrix} a_3 \\ b_3 \\ c_3 \end{pmatrix} = R_y(\beta) R_x(\alpha) T \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

Rotation about the Z-axis

- Now, it is aligned with the Z-axis, thus we can simply rotate θ degrees around the Z-axis.
- Then undo all the transformations we just did



$$\begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} = R_z(\theta) R_y(\beta) R_x(\alpha) T \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

Equation summary

$$rot_{axis}(\theta) = \begin{pmatrix} a' \\ b' \\ c' \end{pmatrix} = T^{-1}R_x^{-1}(\alpha)R_y^{-1}(\beta)R_z(\theta)R_y(\beta)R_x(\alpha)T\begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

Sample View—Transformation

```
void RotateMatrix(float angle, GLfloat X, GLfloat Y, GLfloat Z){
MatrixReset();
GLfloat Cos = cos(angle*Math::PI/180);//角度轉弧度
GLfloat Sin = sin(angle*Math::PI/180);
glMultMatrixf(multiMatrix);}
```

$$\begin{bmatrix} \cos\theta + (1-\cos\theta)x^2 & (1-\cos\theta)xy - (\sin\theta)z & (1-\cos\theta)xz + (\sin\theta)y \\ (1-\cos\theta)yx + (\sin\theta)z & \cos\theta + (1-\cos\theta)y^2 & (1-\cos\theta)yz - (\sin\theta)x \\ (1-\cos\theta)zx - (\sin\theta)y & (1-\cos\theta)zy + (\sin\theta)x & \cos\theta + (1-\cos\theta)z^2 \end{bmatrix}$$

Transformation

```
void TranslateMatrix(GLfloat X, GLfloat Y, GLfloat Z)
{
   MatrixReset();
   multiMatrix[12]=X;
   multiMatrix[13]=Y;
   multiMatrix[14]=Z;
   glMultMatrixf(multiMatrix);
};
```

$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Sample View —Transformation

```
void ScaleMatrix(GLfloat X,GLfloat Y,GLfloat Z)
{
MatrixReset();
multiMatrix[0] = X;
multiMatrix[5] = Y;
multiMatrix[10] = Z;
glMultMatrixf(multiMatrix);
}
```

$$\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Deformations

Transformations that do not preserve shape

- Non-uniform scaling
- Shearing
- Tapering
- Twisting
- Bending

Shearing

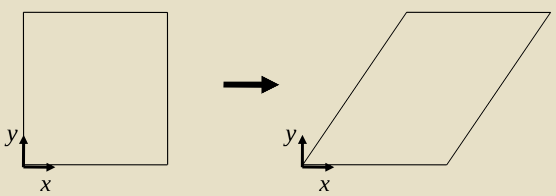
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & s_{xy} & s_{xz} & 0 \\ s_{yx} & 1 & s_{yz} & 0 \\ s_{zx} & s_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$s_{xy} = 1$

$s_{xz} = 0$

$s_{yx} = s_{yz} = 0$

$s_{zx} = s_{zy} = 0$



Tapering

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & f(x) & 0 & 0 \\ 0 & 0 & f(x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

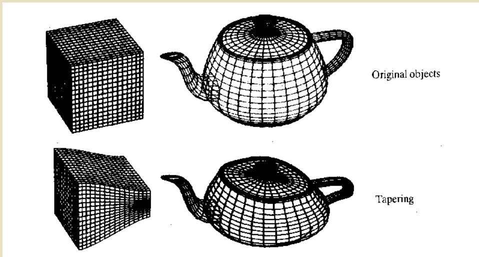
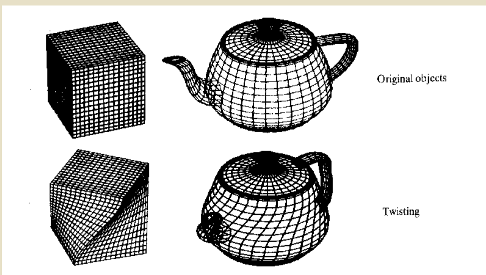


Image courtesy of Watt, 3D Computer Graphics

Twisting

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta(y)) & 0 & \sin(\theta(y)) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta(y)) & 0 & \cos(\theta(y)) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

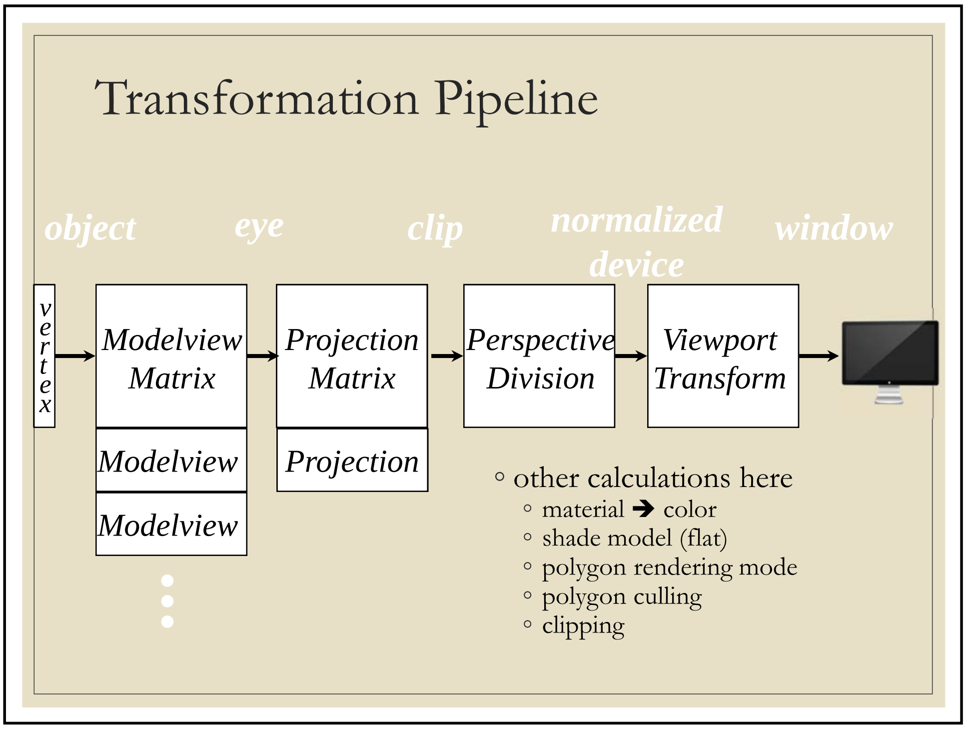


$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

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

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Image courtesy of Watt, 3D Computer Graphics


Transformation Pipeline
object
eye
clip
normalized device
window
vertex
Modelview Matrix
Projection Matrix
Perspective Division
Viewport Transform
Modelview
Projection
Modelview
◦ other calculations here
◦ material ➔ color
◦ shade model (flat)
◦ polygon rendering mode
◦ polygon culling
◦ clipping