Computer Graphics and Virtual Reality

Math

Basic Graphical Mathematics

- **Scalars**
  - Real Numbers
- **Vectors**
  - Direction and Magnitude
- **Matrixes**
  - 2 dimensional array of numbers
- **Points are a position in space(x,y,z)**
  - Measured in a coordinate frame

Basic Graphical Mathematics

- Three views on related concepts
  1. The mathematical view
     - Scalars, points, vectors as member of mathematical sets
     - Variety of abstract spaces and axioms for representing and manipulating these sets
     - (Linear) vector space, affine space, Euclidean space
  2. The geometric view
     - Mapping between the mathematical model and our perceived concept of space
     - Includes points as locations in space
     - Has referential properties (deixis)
  3. Computer science view
     - See concepts as abstract data types (ADTs), a set of operations on data
     - use of geometric ADTs for points, vectors,...

Vectors

- Vector is a directed line segment
- Vectors have a direction and magnitude
  - **Vectors do not have a position!**
- One view is as a displacement between two points

$$V = P_2 - P_1$$

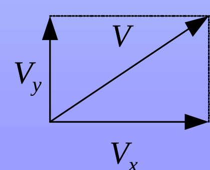
$$V = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

Vector Components

- All vectors can be broken down into their dimensional components vectors
- 3D Vector:

$$V = (V_x, V_y, V_z)$$

- 2D Vector:



Magnitude of a Vector

- Magnitude is “length” of vector

$$|V| = \sqrt{V_x^{\,2} + V_y^{\,2} + V_z^{\,2}}$$

Mathematical Vector Spaces

- (Linear) Vector Space
  - Scalars, Vectors
  - Scalar-Vector Multiplication
  - Vector-Vector Addition
  - Vector-Vector Multiplication
    - Dot Product (Vector-Vector Multiplication to Scalar)
    - Cross Product (Vector-Vector Multiplication to Vector)
- Affine Space
  - Scalars, Vectors, Points
  - Vector-Point Addition

Scalar-Vector Multiplication

- Multiply vector *V* by scalar *a*:

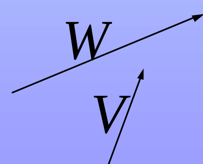
$$aV = (aV_x, aV_y, aV_z)$$

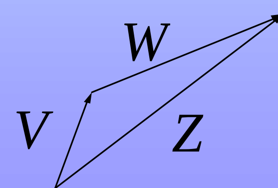
Vector-Vector Addition

- Add vectors V and W:

$$Z = V + W$$

$$= (V_x + W_x, V_y + W_y, V_z + W_z)$$





Vector-Point Addition

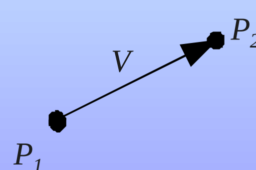
- Add vector V to P1:

$$P_2 = P_1 + V$$

$$P_{2x} = P_{1x} + V_x$$

$$P_{2y} = P_{1y} + V_y$$

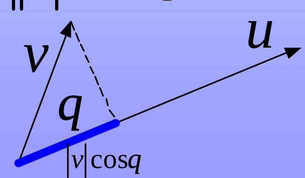
$$P_{2z} = P_{1z} + V_z$$



Vector-Vector Multiplication - Scalar Product

- Aka Dot Product
- Vector – Vector to Scalar Multiplication
- Product of two parallel components of the two vectors
- Vector – Vector multiplication producing a scalar

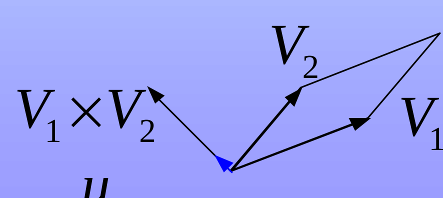
$$u \cdot v = |u||v| \cos q$$



Vector-Vector Multiplication - Cross Product

- Vector – Vector to Vector Multiplication
- Produces Normal Vector
  - Perpendicular to plane formed by the two vectors
  - Magnitude equal to area of parallelogram formed by the two vectors
  - U is unit vector (magnitude 1) that is perpendicular to plane

$$V_1 \times V_2 = u|V_1||V_2| \sin q$$



Vector-Vector Multiplication continued

- Formulas for calculation of dot and cross products in 3D Cartesian space (i.e. orthonormal base vectors):
  - $V \cdot U = u_x v_x + u_y v_y + u_z v_z$
  - $V \times U = (u_y v_z - u_z v_y, u_z v_x - u_x v_z, u_x v_y - u_y v_z)$

Matrixes

- Rectangular Array of Quantities
- Row *(m)* by Column *(n)*

$$A = \begin{bmatrix} a_{11} \; a_{12} \ldots a_{1n} \\ a_{21} \; a_{22} \ldots a_{2n} \\ a_{31} \; a_{32} \ldots a_{3n} \end{bmatrix}$$

Scalar Multiplication of a Matrix

$$
aA = \begin{bmatrix} a\mathrm{a}_{11} & a\mathrm{a}_{12} & a\mathrm{a}_{13} \\ a\mathrm{a}_{21} & a\mathrm{a}_{22} & a\mathrm{a}_{23} \\ a\mathrm{a}_{31} & a\mathrm{a}_{32} & a\mathrm{a}_{33} \end{bmatrix}
$$

Matrix Addition

- Defined only for matrixes of the same number of rows *m* and columns *n*

$$
A + B = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & a_{13} + b_{13} \\ a_{21} + b_{21} & a_{22} + b_{22} & a_{23} + b_{23} \\ a_{31} + b_{31} & a_{32} + b_{32} & a_{33} + b_{33} \end{bmatrix}
$$

Matrix Addition Properties

- Commutative
  - A+B = B+A
- Associative
  - A+(B+C) = (A+B)+C

Matrix Multiplication

- Let A be a *m* x *n* matrix
- Let B be a *n* x *q* matrix
- There for C is a *m* x *q* matrix where:

$$C = AB \Rightarrow c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Matrix Multiplication Example

$$\begin{bmatrix} 0 & -1 \\ 5 & 7 \\ -2 & 8 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 0*1+-1*3 & 0*2+-1*4 \\ 5*1+7*3 & 5*2+7*4 \\ -2*1+8*3 & -2*2+8*4 \end{bmatrix} = \begin{bmatrix} -3 & -4 \\ 26 & 38 \\ 22 & 28 \end{bmatrix}$$

Matrix Multiplication Properties

- Associative
  - A(BC) = (AB)C
- Not Commutative
  - AB does not equal BA

Matrix Transpose

- A^T is interchange of rows and columns
- $(AB)^T = B^T A^T$

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix} \qquad \begin{bmatrix} A & B & C \end{bmatrix}^T = \begin{bmatrix} A \\ B \\ C \end{bmatrix}$$

Identity Matrix

- Square Matrix *I* with 1's along the diagonal and 0's elsewhere
- Multiplication of Identify matrix has no effect: I.e. *a*=*Ia*

$$I = \begin{bmatrix} 1\,0\,0 \\ 0\,1\,0 \\ 0\,0\,1 \end{bmatrix}$$

Inverse Matrix

- Suppose we have a matrix multiplication of a square matrix A such that:
  - $q = Ap$
- Do we have a square matrix B such that:
  - $p = Bq$
  - $p=BAp=Ip=p \;\; => BA=I$
- If so, B is the inverse of A and A is *nonsingular*
- The inverse of A is A^{-1}

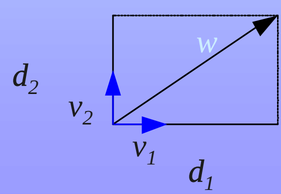
Vector Representation

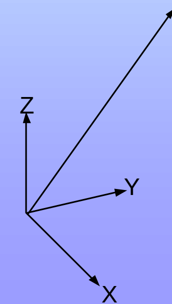
- Any 3D vector can be represented by $V=(V_x, V_y, V_z)$
  - Which is equivalent to: $v= v_x+ v_y+ v_z$
- Any vector can be represented by scaling another vector with the same direction : $V=aV_{st}$

Vector Representation Continued

- Any vector can be represented by a set of scaling values or displacements along with three vectors along the coordinate frame axis's:

$$w = d_1 v_1 + d_2 v_2 + d_3 v_3$$





Vector Representation in Matrix Form

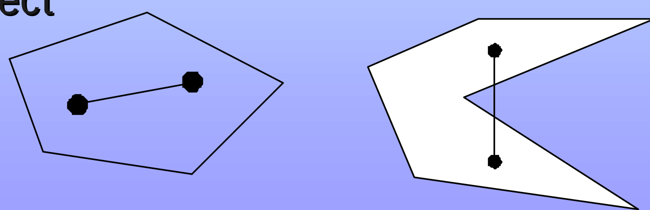
$$w = d_1 v_1 + d_2 v_2 + d_3 v_3$$

Is equivalent to:

$$w = a^T \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad \text{Where} \quad a = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

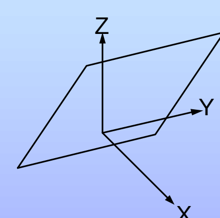
Convex Objects

- Convex Object is one where a line drawn between any two points within the object does not go outside the object



Planes

- A “flat” surface in 3D space
- Minimum of three points needed to define a plane
- Three points define a plane
  - Unless they lie in a straight line
- Four or more points may or may not lie in the same plane



Coordinate Systems

- **Definition**
  - Origin
  - Three Axes (x, y, z)
- **Right-Hand Coordinate Systems**
  - Wrap right hand from x axis to y axis
  - Thumb then points in the positive z direction

Frame

- **Origin Point P_0**
- **Basis Vectors v_1, v_2, v_3**
- **Thus**
  - Unique Vectors

$$w = a_1 v_1 + a_2 v_2 + a_3 v_3$$

  - Unique Points

$$P = P_0 + n_1 v_1 + n_2 v_2 + n_3 v_3$$

Familiar Frame

$$a = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\text{where } \mathrm{P}_0 = (0,0,0) \text{ and}$$

$$\mathrm{v}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, v_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, v_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Further Representation of Points and Vectors

- How do we distinguish between points and vectors represented as:

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \text{where } \mathrm{P} = \mathrm{P}_0 + \mathrm{x}\mathrm{v}_1 + \mathrm{y}\mathrm{v}_2 + \mathrm{z}\mathrm{v}_3$$

$$a = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} \quad \text{where } \mathrm{w} = d_1\mathrm{v}_1 + d_2\mathrm{v}_2 + d_3\mathrm{v}_3$$

Homogeneous representation of vectors and points

- Use of 4x1 matrixes given a frame (v_1, v_2, v_3, P_0)

$$P = a_1 v_1 + a_2 v_2 + a_3 v_3 + P_0 \iff P = (v_1, v_2, v_3, P_0)\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ 1 \end{bmatrix}$$

objects of interest (points, vectors)

$$w = d_1 v_1 + d_1 v_2 + d_1 v_3 \iff w = (v_1, v_2, v_3, P_0)\begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ 0 \end{bmatrix}$$

reference frame

Homogeneous representation arithmetic

The difference of two points is a vector:

$$(a_1, a_2, a_3, 1) - (b_1, b_2, b_3, 1) = (a_1 - b_1, a_1 - b_2, a_1 - b_3, 0)$$

The sum of a point and a vector is a point:

$$(a_1, a_2, a_3, 1) + (b_1, b_2, b_3, 0) = (a_1 + b_1, a_1 + b_2, a_1 + b_3, 1)$$

The sum of a vector and a vector is a vector:

$$(a_1, a_2, a_3, 0) + (b_1, b_2, b_3, 0) = (a_1 + b_1, a_1 + b_2, a_1 + b_3, 0)$$

Scaling a vector: $a * (b_1, b_2, b_3, 0) = (ab_1, ab_2, ab_3, 0)$

Linear combination of vectors is valid

Matrix / Vector Multiplication

$$p = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} \qquad p' = \begin{bmatrix} p'_1 \\ p'_2 \\ p'_3 \\ p'_4 \end{bmatrix} \qquad M = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

$$p' = Mp$$

$$\begin{bmatrix} p'_{11} = a_{11}p_{11} + a_{12}p_{21} + a_{13}p_{31} + a_{14}p_{41} \\ p'_{21} = a_{21}p_{11} + a_{22}p_{21} + a_{23}p_{31} + a_{24}p_{41} \\ p'_{31} = a_{31}p_{11} + a_{22}p_{21} + a_{33}p_{31} + a_{34}p_{41} \\ p'_{41} = a_{41}p_{11} + a_{22}p_{21} + a_{43}p_{31} + a_{44}p_{41} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{21} \\ p_{31} \\ p_{41} \end{bmatrix}$$

2D Transformations

Transformations

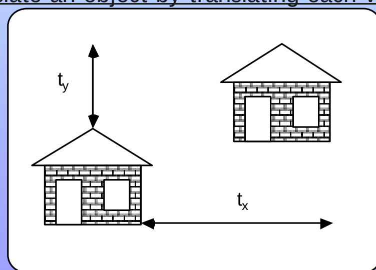
- What are they?
  - changing something to something else via rules
  - mathematics: mapping between values in a range set and domain set (function/relation)
  - geometric: translate, rotate, scale, shear,...
- Why are they important to graphics?
  - moving objects on screen / in space
  - mapping from model space to screen space
  - specifying parent/child relationships
  - ...

Coordinate Systems and Transformations

- Steps in Forming an Image
  - specify geometry (world coordinates)
  - specify camera (camera coordinates)
  - project (window coordinates)
  - map to viewport (screen coordinates)
- Each step uses transformations
- Every transformation is equivalent to a change in coordinate systems (frames)

Translations

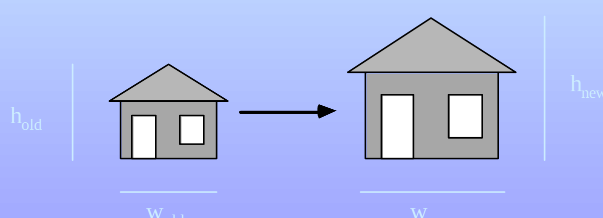
- Moving an object is called a translation. We translate a point by adding to the x and y coordinates, respectively, the amount the point should be shifted in the x and y directions. We translate an object by translating each vertex in the object.



$x_{new} = x_{old} + t_x$; $y_{new} = y_{old} + t_y$

Scaling

- Changing the size of an object is called a scale. We scale an object by scaling the x and y coordinates of each vertex in the object.

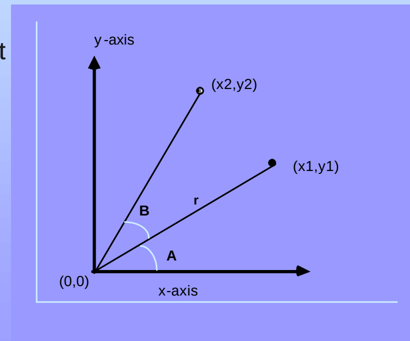


$s_x = w_{new} / w_{old}$ $s_y = h_{new} / h_{old}$

$x_{new} = s_x x_{old}$ $y_{new} = s_y y_{old}$

Rotation about the origin

- To rotate a line or polygon, we must rotate each of its vertices.
- We want to rotate point (x_1,y_1) to point (x_2,y_2) through angle **B**



From the illustration we know that:

$\sin(A + B) = y_2/r \qquad \cos(A + B) = x_2/r$

$\sin A = y_1/r \qquad \cos A = x_1/r$

Rotation about the origin (cont.)

From the double angle formulas: $\sin(A + B) = \sin A \cos B + \cos A \sin B$

Substituting: $y_2/r = (y_1/r)\cos B + (x_1/r)\sin B$

Therefore: $y_2 = y_1 \cos B + x_1 \sin B$

We have

$$\mathbf{x_2 = x_1 \cos B - y_1 \sin B}$$
$$\mathbf{y_2 = x_1 \sin B + y_1 \cos B}$$

Transformations as matrices

Scale:

$x_{new} = s_x x_{old}$

$y_{new} = s_y y_{old}$

$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x \cdot x \\ s_y \cdot y \end{bmatrix}$$

Rotation:

$x_2 = x_1 \cos q - y_1 \sin q$

$y_2 = x_1 \sin q + y_1 \cos q$

$$\begin{bmatrix} \cos q & -\sin q \\ \sin q & \cos q \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos q - y\sin q \\ x\sin q + y\cos q \end{bmatrix}$$

Translation:

$x_{new} = x_{old} + t_x$

$y_{new} = y_{old} + t_y$

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

Homogeneous Coordinates

- In order to represent a translation as a matrix multiplication operation we use 3 x 3 matrices and pad our points to become 3 x 1 matrices. This coordinate system (using three values to represent a 2D point) is called homogeneous coordinates.

$$P_{(x,y)} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad R_q = \begin{bmatrix} \cos q & -\sin q & 0 \\ \sin q & \cos q & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S_{x,y} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad T_{x,y} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Composite Transformations

Suppose we wished to perform multiple transformations on a point:

$$P_2 = T_{3,1}P_1$$

$$P_3 = S_{2,2}P_2$$

$$P_4 = R_{30}P_3$$

$$M = R_{30}S_{2,2}T_{3,1}$$

$$P_4 = MP_1$$

Remember:
- Matrix multiplication is associative, not commutative!
- Transform matrices must be pre-multiplied
- The first transformation you want to perform will be at the far right, just before the point

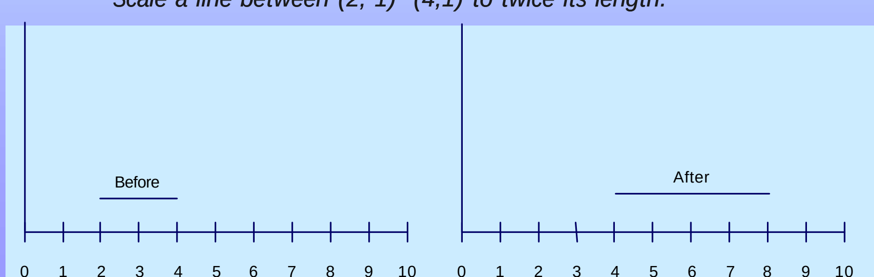
Composite Transformations - Scaling

Given our three basic transformations we can create other transformations.

Scaling with a fixed point

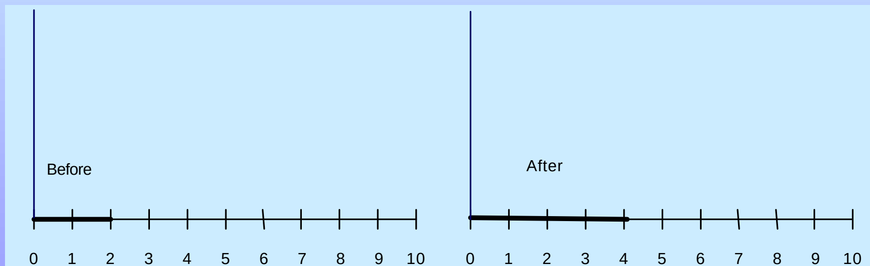
A problem with the scale transformation is that it also moves the object being scaled.

Scale a line between (2, 1) (4,1) to twice its length.



Composite Transforms - Scaling (cont.)

- If we scale a line between (0,0) & (2,0) to twice its length, the left-hand endpoint does not move.



(0,0) is known as a ***fixed point*** for the basic scaling transformation. We can use composite transformations to create a scale transformation with different fixed points.

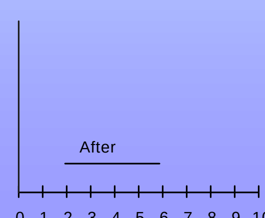
Fixed Point Scaling

Scale by 2 with fixed point = (2,1)

- Translate the point (2,1) to the origin
- Scale by 2
- Translate origin to point (2,1)

$$\underbrace{\begin{bmatrix}1&0&2\\0&1&1\\0&0&1\end{bmatrix}}_{T_{2,1}}\underbrace{\begin{bmatrix}2&0&0\\0&1&0\\0&0&1\end{bmatrix}}_{S_{2,1}}\underbrace{\begin{bmatrix}1&0&-2\\0&1&-1\\0&0&1\end{bmatrix}}_{T_{-2,-1}}=\underbrace{\begin{bmatrix}2&0&-2\\0&1&0\\0&0&1\end{bmatrix}}_{C}$$

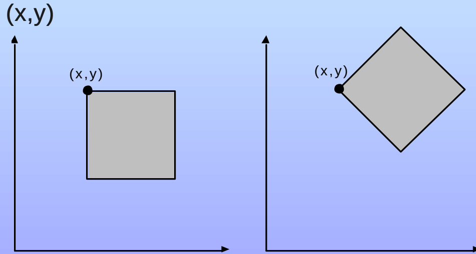
$$\underbrace{\begin{bmatrix}2&0&-2\\0&1&0\\0&0&1\end{bmatrix}}_{C}\begin{bmatrix}2\\1\\1\end{bmatrix}=\begin{bmatrix}2\\1\\1\end{bmatrix} \qquad \underbrace{\begin{bmatrix}2&0&-2\\0&1&0\\0&0&1\end{bmatrix}}_{C}\begin{bmatrix}4\\1\\1\end{bmatrix}=\begin{bmatrix}6\\1\\1\end{bmatrix}$$



Rotation about a Fixed Point

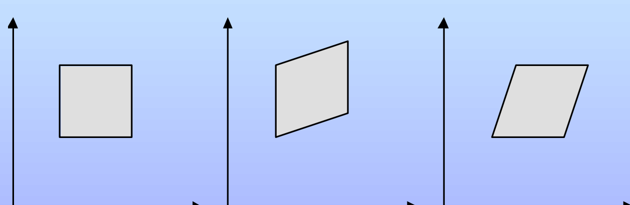
Rotation Of θ Degrees About Point (x,y)

- Translate (x,y) to origin
- Rotate
- Translate origin to (x,y)



$$C = \underbrace{\begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}}_{T_{x,y}} \underbrace{\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R_\theta} \underbrace{\begin{bmatrix} 1 & 0 & -x \\ 0 & 1 & -y \\ 0 & 0 & 1 \end{bmatrix}}_{T_{-x,-y}}$$

Shears



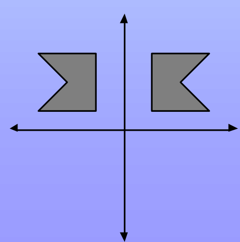
Original Data y Shear x Shear

$$\begin{bmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & b & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Reflections

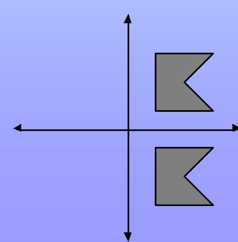
Reflection about the y-axis

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Reflection about the x-axis

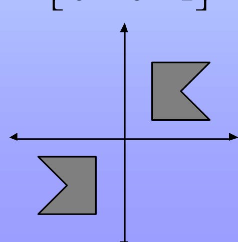
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



More Reflections

Reflection about the origin

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Reflection about the line y=x

?

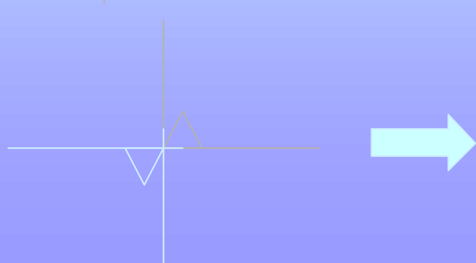
?

Transformations as a change in coordinate system

- All transformations we have looked at involve transforming points in a fixed coordinate system (CS).
- Can also think of them as a transformation of the CS itself

Transforming the CS - examples

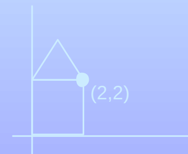
Translate(4,4)



Rotate(180°)

Why transform the CS?

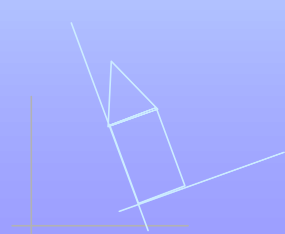
- **Objects often defined in a "natural" or "convenient" CS**



- **To draw objects transformed by T, we could:**
  - Transform each vertex by T, then draw
  - Or, draw vertices <u>in a transformed CS</u>

Drawing in transformed CS

- **Tell system once how to draw the object, then draw in a transformed CS to transform the object**



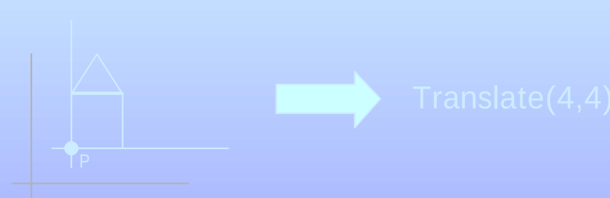
House drawn in a CS that's been translated, rotated, and scaled

$M = S_{x,y} R_d T_{x,y}$

Mapping between systems

- **Given:**
  - The vertices of an object in CS_2
  - A transformation matrix M that transforms CS_1 to CS_2
- **What are the coordinates of the object's vertices in CS_1?**

Mapping example



Point P is at (0,0) in the transformed CS (CS_2). Where is it in CS_1?

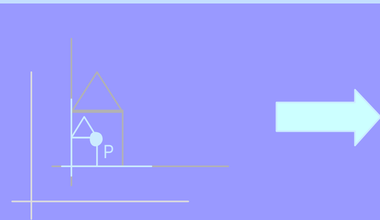
Answer: (4,4)

*Note: (4,4) = $T_{4,4}$ P

Mapping rule

- In general, if CS_1 is transformed by a matrix M to form CS_2, a point P in CS_2 is represented by MP in CS_1

Another example



Translate(4,4), then Scale(0.5, 0.5)

Where is P in CS_3?	(2,2)
Where is P in CS_2?	$S_{0.5,0.5}$ (2,2) = (1,1)
Where is P in CS_1?	$T_{4,4}$ (1,1) = (5,5)

*Note: to go directly from CS_3 to CS_1 we can calculate $T_{4,4}\ S_{0.5,0.5}$ (2,2) = (5,5)

General mapping rule

- If CS_1 is transformed consecutively by $M_1, M_2, \ldots, M_n$ to form CS_{n+1}, then a point P in CS_{n+1} is represented by
  $M_1\ M_2\ \ldots\ M_n$ P in CS_1.
- To form the composite transformation between CSs, you postmultiply each successive transformation matrix if you are using column vectors!!!

Transposes and concatenation

$$M = \begin{pmatrix} a1 & a2 & a3 \\ b1 & b2 & b3 \\ c1 & c2 & c3 \end{pmatrix} \qquad p' = M_2 M_1 p$$

$$M^T = \begin{pmatrix} a1 & b1 & c1 \\ a2 & b2 & c2 \\ a3 & b3 & c3 \end{pmatrix} \qquad p^{T'} = p^T M_1^T\ M_2^T$$

3D Transformations

Types of Affine Transformations:

- Want transformations which preserve geometry (lines, polygons, quadrics...)
  - (affine = line preserving)
- Translation
- Rotation
- Scaling
- Reflection
- Shear
- (Most 3D transformations can be expressed in these 5 transformations)

Affine Transformations

- Property:
  - Parallel lines remain parallel lines
  - Finite points map to finite points
- Non-Affine Transformation: Projection

Homogeneous Coordinates

- each vertex is a column vector

$$\vec{v} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

- w is usually 1.0
- all operations are matrix multiplications
- directions (directed line segments) can be represented with w = 0.0

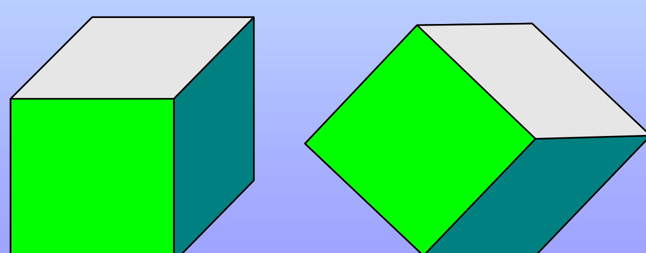
3D Transformations

- A vertex is transformed by 4 x 4 matrices
  - all affine operations are matrix multiplications
  - all matrices are stored column-major in OpenGL
  - matrices are always post-multiplied
  - product of matrix and vector is $\mathbf{M}\vec{v}$

$$\mathbf{M} = \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix}$$

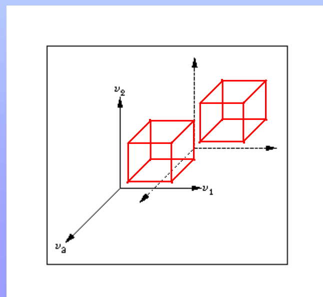
Transformations

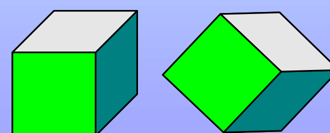
- Change of an object from one form to another



Transformations

- Transformations can be thought of as changing an object within a frame or changing frames





Transformation Math

- Let *p* be a vector or point in homogeneous coordinates (4x1 Column Matrix)
- Let *M* be a 4x4 Transformation Matrix

$$p' = Mp$$

Matrix / Vector Multiplication

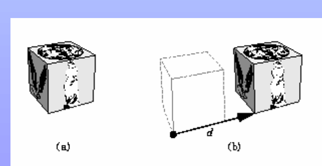
$$p = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} \quad p' = \begin{bmatrix} p'_1 \\ p'_2 \\ p'_3 \\ p'_4 \end{bmatrix} \quad M = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

$$p' = Mp$$

$$\begin{bmatrix} p'_{11} = a_{11}p_{11} + a_{12}p_{21} + a_{13}p_{31} + a_{14}p_{41} \\ p'_{21} = a_{21}p_{11} + a_{22}p_{21} + a_{23}p_{31} + a_{24}p_{41} \\ p'_{31} = a_{31}p_{11} + a_{22}p_{21} + a_{33}p_{31} + a_{34}p_{41} \\ p'_{41} = a_{41}p_{11} + a_{22}p_{21} + a_{43}p_{31} + a_{44}p_{41} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{21} \\ p_{31} \\ p_{41} \end{bmatrix}$$

Translation

- Move object some distance along a displacement vector δ
- Rigid-Body Transformation (object does not change shape)



Translation Math 1

$$p = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, p' = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}, d = \begin{bmatrix} a_x \\ a_y \\ a_z \\ 1 \end{bmatrix}$$

$$x' = x + a_x$$

$$y' = y + a_y$$

$$z' = z + a_z$$

Translation Math 2

$$p' = Tp \;\; where \;\; T = \begin{bmatrix} 1 & 0 & 0 & a_x \\ 0 & 1 & 0 & a_Y \\ 0 & 0 & 1 & a_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T(a_x, a_y, a_z)$$

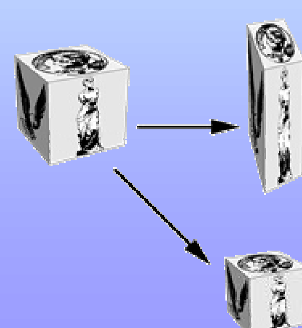
Translation Properties

- Can be reversed by:

$$T^{-1}(a_x, a_y, a_z) = T(-a_x, -a_y, -a_z)$$

Scaling

- Change the size by a given value
- Non-Rigid Body Transformation



Scaling Math 1

$$x' = b_x x$$
$$y' = b_y y$$
$$z' = b_z z$$

Scaling Math 2

$$p' = Sp \text{ where } S = \begin{bmatrix} b_x & 0 & 0 & 0 \\ 0 & b_y & 0 & 0 \\ 0 & 0 & b_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$S(b_x, b_y, b_z)$$

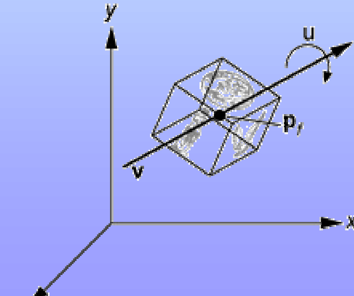
Scaling Properties

- Can be reversed by:

$$S^{-1}(b_x, b_y, b_z) = S(1/b_x, 1/b_y, 1/b_z)$$

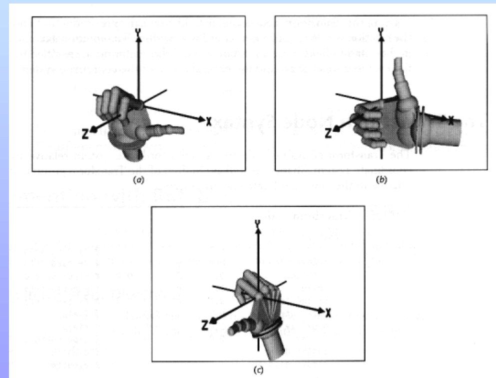
Rotation

- Rotate the object about an axis
- Rigid-Body Transformation (object does not change shape)



Rotation about the main axes

The *right-hand rule* to determine a positive/negative rotation angle around (a) the X axis, (b) the Y axis, and (c) the Z axis



Rotation Math 1

- Rotation is done about a given axis
- Example, Rotation around the Z axis is:

$$x' = x\cos q - y\sin q$$

$$y' = x\sin q - y\cos q$$

$$z' = z$$

Rotation Math 2

- Continuing the rotation about the Z axis:

$$p' = R_z p \ \ where$$

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

Rotation Math 3

- **Rotation about the x axis:**

$$p' = R_x p \ \ where$$

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

Rotation Math 4

- Rotation about the y axis:

$$p' = R_y p \ \ where$$

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

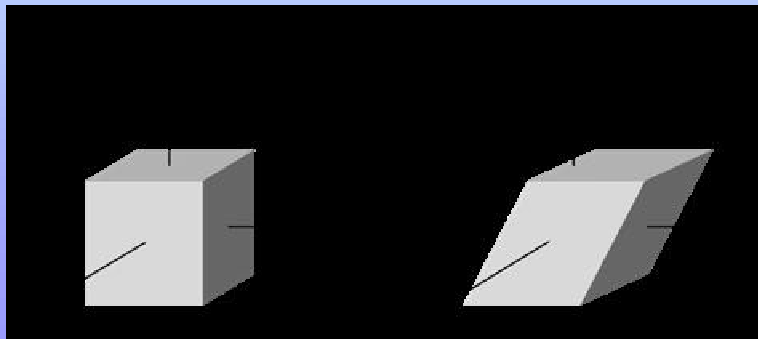
Rotation Properties

- Rotation order is very important!
- Can be reversed by:

$$R^{-1}(q) = R(-q)$$

Shear

- Change shape of object in an arbitrary direction



Shear Math 1

- Shear the object in the x direction by θ

$$x' = x + y \cot q$$

$$y' = y$$

$$z' = z$$

Shear Math 2

$$p' = H_x p \ \ where$$

$$H_x(q) = \begin{bmatrix} 1 & \cot q & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Shear Properties

- **Can be reversed by:**

$$H_x^{-1}(q) = H_x(-q)$$

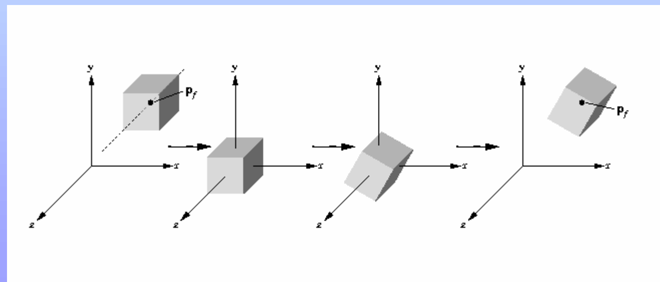
Topic >>
Use of Transformation

Use of Basic Transformation to Perform Complex Transforms

- **Transforms may be done in sequence**
- **Example to rotate an object about a point along the z axis:**
  - Translate Object such that point is on the origin
  - Rotate Object around the z axis
  - Translate Object such that the point is back to its original location

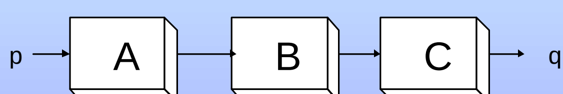
Example

- Rotation of a cube about a point & axis



Transformation Concatenation

$$q = (C(B(Ap)) \Rightarrow q = CBAp$$



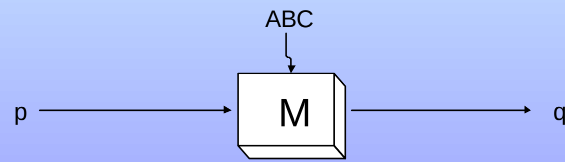
$$M = CBA$$

$$q = Mp$$

- Please Note Reversal of Matrixes!

  $q = CBAp$ *is not same as* $q = ABCp$

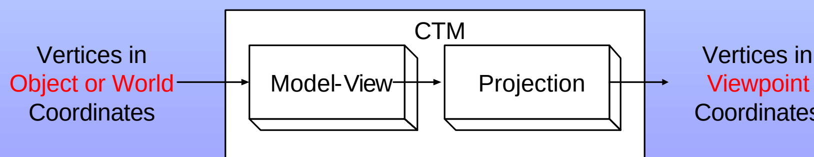
Pipeline Transformation



3D Transformations in OpenGL

OpenGL Issues

- OpenGL is a state machine
- Vertices are transformed using the CTM ("current transformation matrix"):



Transformations in OpenGL

- OpenGL makes it easy to do transformations to the CS, not the object
- Sequence of operations:
  - Set up a routine to draw the object in its "base" CS
  - Call transformation routines to transform the CS
  - Object drawn in transformed CS

Transformations in OpenGL

- Modeling
- Viewing
  - orient camera
  - projection
- Animation
- Map to screen

OpenGL Continued

- Steps:
  - Select Current Matrix
    - glMatrixMode(GL_MODELVIEW)
    - or: glMatrixMode(GL_PROJECTION)
    - or: glMatrixMode(GL_TEXTURE)
  - Set Identity – glLoadIdentity()
  - In reverse order (last loaded is first transformation)
    define the transformation matrices
  - Draw object

OpenGL Continued

- Two forms of transformation matrices:
- Predefined types (Multiply Current Matrix)
  - glTranslated, glTranslatef
  - glRotated, glRotatef
  - glScaled, glScalef
- Build your own matrix
  - glLoadMatrix – Replace current matrix
  - glMultMatrix – Multiple current matrix with new matrix

OpenGL Matrix

- void **glLoadMatrixd**(GLdouble **m*)
  void **glLoadMatrixf**(GLfloat **m*)
- **glLoadMatrix** replaces the current matrix with the one specified in *m*.
- *m* points to a 4x4 matrix of single- or double-precision floating-point values stored in column-major order. That is, the matrix is stored as follows:

$$\begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}$$

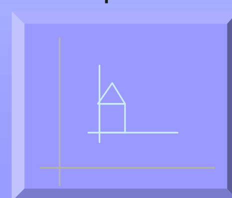
OpenGL transformation example

```
drawHouse(){
   glBegin(GL_LINE_LOOP);
        glVertex2i(…);
        glVertex2i(…);
        …
   glEnd();
}
```

Draws basic house

```
drawTransformedHouse(){
   glMatrixMode(GL_MODELVIEW);
   glTranslated(4.0, 4.0,
   0.0);
   glScaled(0.5, 0.5, 1.0);
   drawHouse();
}
```

Draws transformed house
(push for example car transform)



Notes on OpenGL code

`glMatrixMode(GL_MODELVIEW);`

- Which “current transformation matrix” am I modifying?

`glTranslated(4.0, 4.0, 0.0);`
`glScaled(0.5, 0.5, 1.0);`

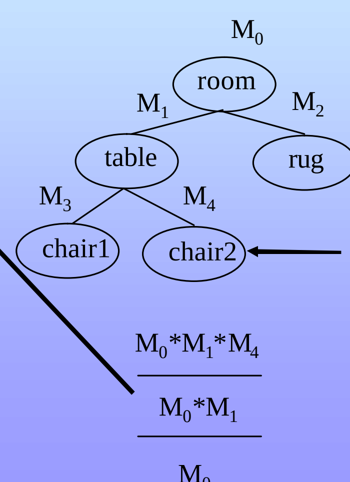
- Setting up the current tranformation matrix - the next vertices specified will be transformed by $T_{4,4}\,S_{0.5,0.5}$ in order to obtain their values in the original CS.
- Note three values for each - all OpenGL transformations are 3D
- Also **`glRotated(degrees,`** $\mathbf{vect}_x$, $\mathbf{vect}_y$, $\mathbf{vect}_z$**`)`** where vect is the vector about which the rotation is done - for 2D, vect = (0, 0, 1)

Composite transformations in OpenGL

- concept of matrix stacks
- supports hierarchical representations
- pushmatrix, popmatrix
- loadmatrix
- multmatrix

OpenGL matrix stack example

```
glLoadMatrixf(m0);
glPushMatrix();
glMultMatrixf(m1);
glPushMatrix();
glMultMatrixf(m4);
render chair2;
glPopMatrix();
glMultMatrixf(m3);
render chair1;
glPopMatrix();
render table;
glPopMatrix();
render room;
glPushMatrix();
glMultMatrixf(m2);
render rug;
```



OpenGL Matrix Management

- Matrix Stack (glPushMatrix(), glPopMatrix())
  - Push a **copy** of the current matrix onto the stack
    - Saves a copy of the current matrix but does not remove current matrix
  - Pop the matrix off the stack into the current matrix
    - Replaces current matrix
- Standard Code:

```
glPushMatrix();
glTranslatef(…);
glRotatef(…);
glScalef(…);
/* Draw Object */
glPopMatrix();
```

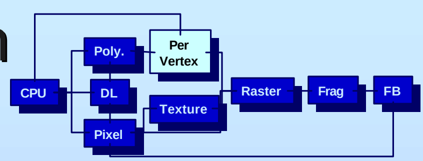
Specifying Transformations

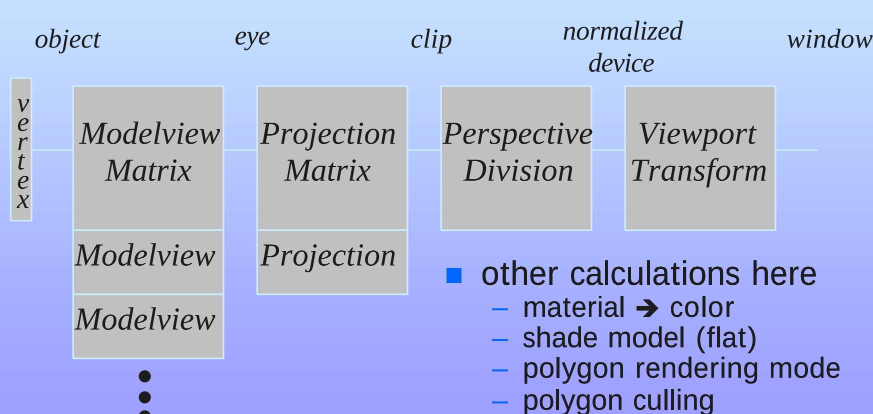
- Programmer has two styles of specifying transformations
  - specify matrices (`glLoadMatrix, glMultMatrix`)
  - specify operation (`glRotate, glOrtho`)
- Programmer does not have to remember the exact matrices
  - check appendix of Red Book (Programming Guide)

Programming Transformations

- Prior to rendering, view, locate, and orient:
  - eye/camera position
  - 3D geometry
- Manage the matrices
  - including matrix stack
- Combine (composite) transformations

Transformation Pipeline





- other calculations here
  - material → color
  - shade model (flat)
  - polygon rendering mode
  - polygon culling
  - clipping

Matrix Operations

- Specify current matrix stack

  `glMatrixMode( GL_MODELVIEW or GL_PROJECTION )`
- Other Matrix or stack operations

  `glLoadIdentity()`
  `glPushMatrix()`
  `glPopMatrix()`
- Viewport
  - usually same as window size
  - viewport aspect ratio should be same as projection transformation or resulting image may be distorted

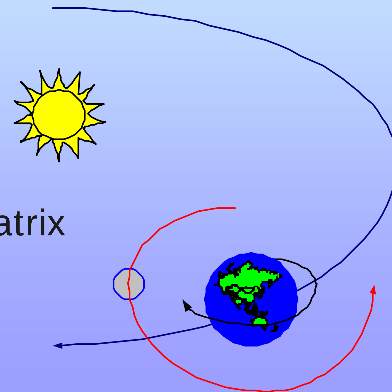
  `glViewport( x, y, width, height )`

Object Representation vs World Representation

- Object Coordinates (aka local coordinates)
- World Coordinates
- Manipulation of Objects in World
- Object Templates, Instances, Duplication
- Object Hierarchies
  - Object Coordinate Hierarchies
- Not all model formats support object coordinates
- Role of Object-Oriented Programming

Example of Transformation Use

- Planet orbiting a sun:
- Each Time Tick:
  - Clear Transform
  - Set Rotation Matrix
  - Set Transformation Matrix
  - Draw Object



Transformation examples

- Some tutor examples using transformations and the transformation stack

push for example