How Are Transformations Used in Computer Graphics?

• Object construction using assemblies/hierarchy of parts à la Sketchpad’s masters and instances; leaves contain primitives

• Aid to realism
  – objects, camera use realistic motion

• Help form “object hypothesis”
  – kinesthetic feedback as user manipulates objects or synthetic camera

• Synthetic camera/viewing
  – definition
  – normalization

• Note: Play with transformation exploratory
  – transformation game, hierarchy applet, math applets
  – http://localhost/~rgb/illus/cs123applets.html

Problem: How to move an object

• We know how to move objects in real life
• How can we move them on a computer?
• Goal: move the house from “here” to “there”

Cartesian Coordinate Spaces

• Examples: one, two and three dimensional real coordinate spaces

• Real numbers: between any two real numbers on an axis there exists another real number

• Compare with the computer screen, a positive integer coordinate system
Vectors & Vector Space (1/3)

- Coordinate systems can tell how far objects have moved, but what about relationships between the objects?
- Consider all locations in relationship to one central reference point, called the origin

A vector tells you which direction to go with respect to the origin, and length of the trip.

Notation: column \([x, y]\), or for typographical reasons, row \([x, y]\)

-for example, the vector pointing to the center of the car is \([10, 2]\)

Vectors & Vector Space (2/3)

- You may have seen vectors used in a physics class. For example, vector \(F\) below represents force applied to brick resting on ground

- Break down force into components to see how much is actually pushing the box to the right \((F_x)\) vs. how much is pushing down \((F_y)\)

- Vectors are used extensively in computer graphics to
  - represent positions of vertices of objects
  - determine orientation of a surface in space (“surface normal”)
  - create impression of light interacting with solid and transparent objects (e.g., vectors from light source to surface)

Let’s use vector and matrix notation...

Vectors & Vector Space (3/3)

- House in a completely unstructured space (bounded by a rectangle)

- House in a coordinate space

- House in a vector space

2D Object Definition (1/3)

Lines and Polylines

- Lines drawn between ordered points to create more complex forms called polylines

- Same first and last point make closed polyline or polygon
- Can intersect itself
  - if it does not, called simple polygon

Convex vs. Concave Polygons

Convex: For every pair of points in the polygon, the line between them is fully contained in the polygon.

Concave: Not convex. So some two points in the polygon are joined by a line not fully contained in the polygon.
2D Object Definition (2/3)

Special polygons

- triangle
- square
- rectangle

Circles

- Consist of all points equidistant from one predetermined point (the center)
- \( r \) (radius) = \( c \), where \( c \) is a constant
- On a Cartesian grid with center of circle at origin equation is \( r^2 = x^2 + y^2 \)

2D Object Definition (3/3)

Circle as polygon

- Informally: a polygon with >15 sides

(Aligned) ellipses

- A circle scaled either along the x or y axis
- Example: height, on y-axis, remains 3 while length, on x-axis, changes from 3 to 6

2D to 3D Object Definition

Vertices in motion ("Generative object description")

- Line is drawn by tracing path of a point as it moves (one dimensional entity)
- Square drawn by tracing vertices of a line as it moves perpendicularly to itself (two dimensional entity)

- Cube drawn by tracing paths of vertices of a square as it moves perpendicularly to itself (three-dimensional entity)
- Circle drawn by swinging a point at a fixed length around a center point

Polygons into polyhedra

Moving Objects with Vectors

Vector addition in \( \mathbb{R}^1 \)

- Familiar addition of real numbers
  \( P'' = P' + P' \)


Vector addition in \( \mathbb{R}^2 \)

- The x and y parts of vectors can be added using addition of real numbers along each of the axes (component-wise addition)

  \[ P + P' = \begin{bmatrix} x' + x \\ y' + y \end{bmatrix} \]

- Result, \( P + P' \), plotted in \( \mathbb{R}^2 \) is the new vector
**Adding Vectors Visually**

- \( P' \) added to \( P \), using the parallelogram rule:
  - Take vector from the origin to \( P' \); reposition it so that its starting point is at the end-point of vector \( P \); define \( P + P' \) as the end of the new vector.

- Or, equivalently, add \( P' \) to \( P \).

**Scalar Multiplication (1/2)**

- On \( \mathbb{R}^1 \), familiar multiplication rules.
- On \( \mathbb{R}^2 \) also:

  \[
  2P = (P + P)
  \]

**Scalar Multiplication (2/2)**

**Linear Dependence**

- Set of all scalar multiples of a vector is a line through the origin.
- Two vectors are *linearly dependent* when one is a multiple of the other.

**Basis vectors of the plane**

- The *unit vectors* (i.e., whose length is one) on the \( x \) and \( y \)-axes are called the *standard basis vectors* of the plane.
- The collection of all scalar multiples of a vector gives the first coordinate axis.
- The collection of all scalar multiples of another vector gives the second coordinate axis.

**Non-orthogonal Basis Vectors**

- \( \begin{bmatrix} 1 \\ 0 \end{bmatrix} \) and \( \begin{bmatrix} 0 \\ 1 \end{bmatrix} \) are perpendicular. Necessary?
- Question rephrased: can we make any vector from scalar multiples of random vectors \( \begin{bmatrix} a \\ b \end{bmatrix} \) and \( \begin{bmatrix} c \\ d \end{bmatrix} \)?
- Have \( \begin{bmatrix} a \\ b \end{bmatrix} = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha a + \beta b \\ \alpha d + \beta c \end{bmatrix} \)
  - Know all variables except for \( \alpha \) and \( \beta \) and have two equations, so can deduce both values.
- Note! Doesn’t work if candidate basis vectors are linearly dependent.

- Geometric description:

  | OK | OK | not OK | not OK |
Algebraic Properties of Vectors

- Commutative (vector) \( P + Q = Q + P \)
- Associative (vector) \((P + Q) + R = P + (Q + R)\)
- Additive identity: There is a vector 0 such that, for all \( P \), \((P + 0) = P = (0 + P)\)
- Additive inverse: For any \( P \) there is a vector \(-P\) such that \( P + (-P) = 0\)
- Distributive (vector) \( r(P + Q) = rP + rQ\)
- Distributive (scalar) \((r + s)P = rP + rQ\)
- Associative (scalar) \( r(sP) = (rs)P\)
- Multiplicative identity: For any \( P, 1 \in \mathbb{R}, 1P = P\)

The Dot Product

Uses of the dot product

- Define length of a vector
- Normalize vectors (generate vectors whose length is 1, called unit vectors)
- Measure angles between vectors
- Determine if two vectors are perpendicular
- Find length of projection of a vector onto a coordinate axis (as in force example from before)

F = \( F_x \) \( F_y \)

Finding the Length of a Vector

- The dot product of a vector with itself, \((P \cdot P)\), is the square of the length of the vector:
  \( P \cdot P = x^2 + y^2 \)
- We define the norm of a vector (i.e., its length) to be \( |P| = \sqrt{P \cdot P} \)
- Thus \((P \cdot P) \geq 0\) for all \( P \), with equality if and only if \( P = 0\)
- \( W \) is called a unit vector if \(|W| = 1\)

Rule for Dot Product

- Also known as scalar product, or inner product. The result is a scalar (i.e., a number, not a vector).
- Defined as \( P \cdot P' = \begin{bmatrix} x \end{bmatrix} \begin{bmatrix} x' \end{bmatrix} = xx' + yy' \)
- Example: for \( P = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \) and \( P' = \begin{bmatrix} 2 \\ 4 \end{bmatrix} \)
  \( P \cdot P' = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix} = (3 \cdot 2) + (2 \cdot 4) = 6 + 8 = 14 \)
- Not the same as component-wise multiplication of real numbers
Finding the Angle Between Two Vectors

- The dot product of two non-zero vectors is the product of their lengths and the cosine of the angle between them: \[ |P||P'|\cos(\theta - \phi) \]

More Uses of the Dot Product

Finding the length of a projection

- If \( W \) is a unit vector, then \( P \cdot W \) is the length of the projection of \( P \) onto the line containing \( W \)

Determining right angles

- Perpendicular vectors always have a dot product of 0 because the cosine of 90° is 0
- Example: \( P = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \) and \( P' = \begin{bmatrix} -3 \\ 4 \end{bmatrix} \)

A Non-Geometric Example (2/2)

- Let’s use a shorthand to represent the situation (assuming we can remember order of items and corresponding prices):

  \[ X' = \begin{bmatrix} 6 \\ 5 \\ 1 \\ 2 \end{bmatrix} \]

  \[ P = \begin{bmatrix} 0.20 \\ 0.93 \\ 0.64 \\ 1.20 \end{bmatrix} \]
  \[ P' = \begin{bmatrix} 0.65 \\ 0.95 \\ 0.75 \\ 1.40 \end{bmatrix} \]
  \[ P'' = \begin{bmatrix} 0.95 \\ 1.10 \\ 0.90 \end{bmatrix} \]

Let’s Go Shopping

- Need 6 apples, 5 cans of soup, 1 box of tissues, and 2 bags of chips.
- Stores A, B, and C (East Side Market, Bread & Circus, and Store24) have following unit prices respectively
  - 1 apple: \$0.20 \$0.65 \$0.95
  - 1 can soup: \$0.93 \$0.95 \$1.10
  - box tissues: \$0.64 \$0.75 \$0.90
  - bag chips \$1.20 \$1.40 \$3.50
What do I Pay?

Let’s calculate for each of the three stores.

- Store A:
  \[ \sum_{i=1}^{4} P_{Ai}Q_{i} = totalCost_{A} \]
  \[ = (0.20 \cdot 6) + (0.93 \cdot 5) + (0.64 \cdot 1) + (1.20 \cdot 2) \]
  \[ = 1.2 + 4.65 + 0.64 + 2.40 \]
  \[ = 8.89 \]

- Store B:
  \[ \sum_{i=1}^{4} P_{Bi}Q_{i} = totalCost_{B} \]
  \[ = 3.9 + 4.75 + 0.75 + 2.8 \]
  \[ = 12.2 \]

- Store C:
  \[ \sum_{i=1}^{4} P_{Ci}Q_{i} = totalCost_{C} \]
  \[ = 5.7 + 5.5 + 0.9 + 7 \]
  \[ = 19.1 \]

Using Matrix Notation

- We can express these sums more compactly:
  \[ P(All) = \begin{bmatrix} totalCost_{A} \\ totalCost_{B} \\ totalCost_{C} \end{bmatrix} = \begin{bmatrix} 0.20 & 0.93 & 0.64 & 1.20 \\ 0.65 & 0.95 & 0.75 & 1.40 \\ 0.95 & 1.10 & 0.90 & 3.50 \end{bmatrix} \]
  \[ \begin{bmatrix} 6 \\ 5 \\ 12 \end{bmatrix} \]

A vector is a way of writing a list, and a matrix a way of writing a list of lists.

- The totalCost vector is determined by row-column multiplication where row = price, column = quantities.

- More generally, if two people went shopping (and were purchasing different quantities of the same items as above) we could express necessary multiplications as:

  \[ \begin{bmatrix} P_{A} \\ P_{B} \\ P_{C} \end{bmatrix} = \begin{bmatrix} 0.20 & 0.93 & 0.64 & 1.20 \\ 0.65 & 0.95 & 0.75 & 1.40 \\ 0.95 & 1.10 & 0.90 & 3.50 \end{bmatrix} \begin{bmatrix} 6 \\ 5 \\ 12 \end{bmatrix} \]

Basic Transformations of the Plane

- Vectors, and the operations of addition, scalar multiplication, and dot product will be used to
  - translate (move)
  - rotate
  - scale
  - reflect
  - shear

objects made on the computer

Translation

- Component-wise addition of vectors
  \[ P' = P + T \]
  where \[ P = \begin{bmatrix} x' \\ y' \end{bmatrix} \]
  \[ \begin{bmatrix} dx \\ dy \end{bmatrix} \]
  and
  \[ x' = x + dx \]
  \[ y' = y + dy \]

- To move polygons: just translate vertices (vectors) and then redraw lines between them

- Preserves lengths (isometric)

- Preserves angles (conformal)

Note: House shifts position relative to origin
### Scaling
- Component-wise scalar multiplication of vectors $P' = SP$ where $P = \begin{bmatrix} x \\ y \end{bmatrix}$, and
  - $x' = s_x x$
  - $y' = s_y y$
- Does not preserve lengths
- Does not preserve angles (except when scaling is uniform)

Note: House shifts position relative to origin

### Rotation
- Rotation of vectors through an angle $\theta$
  - $P' = R_{\theta}P$ where $P = \begin{bmatrix} x \\ y \end{bmatrix}$ and
  - $x' = x \cos \theta - y \sin \theta$
  - $y' = x \sin \theta + y \cos \theta$
- Proof is by double angle formula
- Preserves lengths and angles

### Sets of Linear Equations and Matrices
- To translate, scale, and rotate vectors we need a function to give a new value of $x$, and a function to give a new value of $y$
- Examples:
  - for rotation
    - $x' = x \cos \theta - y \sin \theta$
    - $y' = (x \sin \theta + y \cos \theta)$
  - for scaling
    - $x' = s_x x$
    - $y' = s_y y$
- These two, but not translation, are of the form
  - $x' = ax + by$
  - $y' = cx + dy$
- A transformation given by such a system of linear equations is called a linear transformation and is represented by a matrix: $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$

### Types of Transformations
- Projective $\supset$ affine $\supset$ linear
- Linear: acts on a line to yield either another line or a point. The vector $(0, 0)$ is always transformed to $(0, 0)$.
- Affine: preserves parallel lines. The vector $(0, 0)$ is not always transformed to $(0, 0)$.
- Projective: parallel lines not necessarily preserved, but lines are sent to lines or points (not curves)
Some Important Matrices

- For rotation
  \[ x' = x \cos \theta - y \sin \theta \]
  \[ y' = x \sin \theta + y \cos \theta \]
  and the rotation matrix is
  \[ R_\theta = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \]

- For scaling
  \[ x' = s_x x \]
  \[ y' = s_y y \]
  and the scaling matrix is
  \[ S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \]

- Note that translation is not of the form
  \[ x' = ax + by \]
  \[ y' = cx + dy \]
  – at this point we cannot write it as a matrix
  – it is an affine, but not a linear, transformation

Two Other Matrices

- For reflection (across the \( y \)-axis)
  \[ x' = -x \]
  \[ y' = y \]
  and the \( y \)-axis reflection matrix is
  \[ R_{E_y} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \]

- For shearing (along the \( x \)-axis)
  \[ x' = x + ay \]
  and the matrix is
  \[ S_{H_x} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \]

Matrix-Vector Multiplication

- As in shopping example
  \[ X' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = MX \]
  where \( M = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \), \( X = \begin{bmatrix} x \\ y \end{bmatrix} \), \( X' = \begin{bmatrix} x' \\ y' \end{bmatrix} \)

  The new vector is the dot product of each row of the matrix with the column vector. Thus, the 1st entry of the transformed vector is the dot product of the 1st row of the matrix with the original vector.

  Example: scaling the vector \( \begin{bmatrix} 3 \\ 3 \end{bmatrix} \) by 7 in the \( x \) direction and .5 in the \( y \) direction

  \[ \begin{bmatrix} 7 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix} = (7 \cdot 3) + (0 \cdot 3) = 21 + 0 = \begin{bmatrix} 21 \\ 0 \end{bmatrix} \]

In general:

\[
\begin{bmatrix}
  a_1 & a_2 & a_3 & \ldots & a_n \\
  b_1 & b_2 & b_3 & \ldots & b_n \\
  c_1 & c_2 & c_3 & \ldots & c_n \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  m_1 & m_2 & m_3 & \ldots & m_n \\
\end{bmatrix} \begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  \vdots \\
  x_n \\
\end{bmatrix}
\]

\[ = \begin{bmatrix}
  (a_1x_1) + (a_2x_2) + (a_3x_3) + \ldots + (a_nx_n) \\
  (b_1x_1) + (b_2x_2) + (b_3x_3) + \ldots + (b_nx_n) \\
  (c_1x_1) + (c_2x_2) + (c_3x_3) + \ldots + (c_nx_n) \\
  \vdots \\
  (m_1x_1) + (m_2x_2) + (m_3x_3) + \ldots + (m_nx_n) \\
\end{bmatrix} \]

\[ \text{can also express as:} \]

\[ \sum_{i=1}^{n} a_i x_i \]

\[ = \begin{bmatrix}
  a \cdot x \\
  b \cdot x \\
  \vdots \\
  m \cdot x \\
\end{bmatrix} \]

\[ \sum_{i=1}^{n} b_i x_i \\
\sum_{i=1}^{n} m_i x_i \]

\[ = \begin{bmatrix}
  a \\
  b \\
  \vdots \\
  m \\
\end{bmatrix} \begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_n \\
\end{bmatrix} \]

\[ = MX \]
Finding a Matrix
We can use the effect of the transformation on the basis vectors $[1,0]$ and $[0,1]$ to find the coefficients that make up the transformation matrix:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} a \\ c \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} b \\ d \end{bmatrix}$$

• Example: find the matrices that will accomplish the transformations below

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} a & c \\ b & d \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Matrix Composition (1/2)
• Rule for composing matrices

For $M_1 = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $M_2 = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$, and $X = \begin{bmatrix} x \\ y \end{bmatrix}$,

$$M_1(M_2(X)) = (M_1M_2)(X)$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ae + bf & bg + bh \\ ce + df & cf + dh \end{bmatrix}$$

Remembering the formula:

$$\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ \vdots & \vdots \\ a_n & b_n \end{bmatrix} \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \beta_1 & \beta_2 & \cdots & \beta_n \\ \Gamma_1 & \Gamma_2 & \cdots & \Gamma_n \end{bmatrix} = \begin{bmatrix} \alpha_1 \beta_1 + \cdots + \alpha_n \beta_n \\ \alpha_1 \Gamma_1 + \cdots + \alpha_n \Gamma_n \\ \vdots & \vdots & \ddots & \vdots \\ \beta_1 \Gamma_1 + \cdots + \beta_n \Gamma_n \end{bmatrix}$$

• Element at position (i,j) of result is the dot product of row i of first matrix with column j of second

Matrix Composition (2/2)

Combining Transformations
The problem with Scaling and Rotating
• The house shifts position relative to the origin. Not what you would expect when rotating or scaling an object.

More natural solution
• Move house to origin, then scale, (and/or rotate), then move house back to original position
• Matrices performed in a sequence can be composed into a single matrix
Homogenous Coordinates

- Translation, scaling and rotation are expressed (non-homogeneously) as:

  - Translation:
    \[ P' = P + T \]
  
  - Scale:
    \[ P' = S \cdot P \]
  
  - Rotation:
    \[ P' = R \cdot P \]

- Composition is difficult to express, since translation not expressed as a matrix multiplication.

- Homogeneous coordinates allow all three to be expressed homogeneously, using multiplication by 3 x 3 matrices.

\[
\begin{align*}
P_{2d}(x, y) &\rightarrow P_h(wx, wy, w), w \neq 0 \\
P_h(x', y', w), w \neq 0 \\
P_{2d}(x, y) &= P_{2d}(\frac{x}{w}, \frac{y}{w})
\end{align*}
\]

- W is 1 for affine transformations in graphics.

Homogenous Coordinate Transformations (1/2)

- For points written in homogeneous coordinates \[ \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \]

  - Translation, scaling and rotation are expressed homogeneously as:

    \[
    T(dx, dy) = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}
    \]

    \[
    S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}
    \]

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

Homogenous Coordinate Transformations (2/2)

- Consider

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

- The 2 x 2 submatrix columns:
  - are unit vectors (length=1)
  - are perpendicular (dot product = 0)
  - are vectors into which X-axis and Y-axis rotate

- The 2 x 2 submatrix rows:
  - are unit vectors
  - are perpendicular
  - rotate into X-axis and Y-axis

- These properties are “rigid body” and preserve lengths and angles.
Composition of 2D Transforms

- $R(\phi)$ rotates about the origin; to rotate about point $P_1$
  1. Translate $P_1$ to origin
  2. Rotate
  3. Translate origin back to $P_1$

$$\begin{bmatrix}
1 & 0 & x_1 \\
0 & 1 & y_1 \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
cos\phi & -sin\phi & 0 \\
sin\phi & cos\phi & 0 \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
1 & 0 & -x_1 \\
0 & 1 & -y_1 \\
0 & 0 & 1
\end{bmatrix}$$

Scale, Rotate, and Translate

- To scale, rotate and translate an arbitrary symbol about center $P_1$, place at $P_2$:
  1. Translate $P_1$ to origin
  2. Scale
  3. Rotate
  4. Translate origin to $P_2$

Note: these operations are not, in general, commutative because matrix multiplication isn't: i.e., in general

$$M_2M_1 \neq M_1M_2$$

Commutative and Non-Commutative Combinations of Transformations in 2D

- Commutative
  - translate, translate
  - scale, scale
  - rotate, rotate
  - scale uniformly, rotate

- Non-commutative
  - non-uniform scale, rotate
  - translate, scale
  - rotate, translate

3D Basic Transformations (1/2)

(right-handed coordinate system)

- Translation
  $$\begin{bmatrix}
1 & 0 & 0 & dx \\
0 & 1 & 0 & dy \\
0 & 0 & 1 & dz
\end{bmatrix}$$

- Scaling
  $$\begin{bmatrix}
s_x & 0 & 0 & 0 \\
0 & s_y & 0 & 0 \\
0 & 0 & s_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}$$
3D Basic Transformations (2/2)
(right-handed coordinate system)

- Rotation about X-axis
  \[
  \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & \cos \theta & -\sin \theta & 0 \\
  0 & \sin \theta & \cos \theta & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]

- Rotation about Y-axis
  \[
  \begin{bmatrix}
  \cos \theta & 0 & \sin \theta & 0 \\
  0 & 1 & 0 & 0 \\
  -\sin \theta & 0 & \cos \theta & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]

- Rotation about Z-axis
  \[
  \begin{bmatrix}
  \cos \theta & -\sin \theta & 0 & 0 \\
  \sin \theta & \cos \theta & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]

Homogeneous Coordinates

Some uses we’ll be seeing later

- Construction: putting sub-objects in their parents’ coordinate system
  - transforming primitives in their coordinate system
- View volume mapping
  - 2D: Window \rightarrow>VViewport mapping
  - 3D: normalization: mapping arbitrary view volume into canonical view volume along the z-axis
- Parallel (orthographic and oblique) and perspective projection
- Perspective transformation

So How Do We Use Transforms?
(1/2)

- 3D scenes are typically stored in a DAG called a scene graph (or scene tree) created by calls on a library
  - OpenInventor
  - Java3D
  - VRML
- Typical scene graph format
  - objects (cubes, sphere, cone, etc.) with basic defaults (located at the origin with unit area of volume) are stored as nodes in the graph
  - other things like attributes (color, texture map, etc.) and transformations are also nodes in the scene graph (labeled edges on page 1 are an abstraction) and get applied to different object nodes depending on their position in the tree
- For your assignments, you will deal with a much simpler scene graph format
  - attributes of each object will be stored as a component of the object node (no separate attribute node)
  - transform node will affect its subtree, but not siblings
  - transform node can only have one child
  - only leaf nodes are graphical objects
  - all internal nodes that are not transform nodes are group nodes

So How Do We Use Transforms?
(2/2)

- In the scene graph below, transformation t0 will affect all objects, but t2 will only affect obj2 and one instance of group3 (which includes an instance of obj3 and obj4)
  - t2 doesn’t affect obj1, other instance of group3
- Note that if you want to use multiple instances of a sub-tree, such as group3 above, you must define it before it’s used
  - this is so that it’s easier to implement
Hierarchical Transformation (1/2)

• Typically, transformation nodes contain at least a matrix that handles the transformation; additionally, it may contain individual transformation parameters
  – refer to scene graph hierarchy applet by Dave Karelitz (URL on p. 1)

• To determine the final composite transformation matrix (CTM) for an object node, you need to compose all parent transformations during prefix graph traversal
  – exact detail of how this is done varies from package to package, so be careful

Hierarchical Transformation (2/2)

• An example:

  \[ \begin{align*}
  \text{for o1, } & CTM = m_1 \\
  \text{for o2, } & CTM = m_2 \times m_3 \\
  \text{for o3, } & CTM = m_2 \times m_4 \times m_5 \\
  \text{for a vertex in o3, its position, size and orientation in the world (root) coordinate system is:} & \\
  & CTM \times \mathbf{x} = (m_2 \times m_4 \times m_5) \times \mathbf{x}
  \end{align*} \]

Addendum—Matrix Notation

• The application of matrices in the row vector notation is executed in the reverse order of application in the column vector notation:

  \[\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \leftrightarrow \begin{bmatrix} x \\ y \\ z \end{bmatrix}\]

  • Column format: vector follows transformation matrix.

  \[\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}\]

  • Row format: vector precedes matrix and is post-multiplied by it.

  \[\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}\]

  • By convention, we always use column vectors.

But, There’s a Problem...

• Notice that

  \[\begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} ax + dy + gz \\ bx + ey + hz \\ cx + fy + iz \end{bmatrix}\]

  while

  \[\begin{bmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{bmatrix}\]
Solution to Notational Problem

- In order for both types of notations to yield same result, matrix in row system must be transpose of matrix in column system.

\[
\begin{bmatrix}
a & b & c \\
d & e & f \\
g & h & i
\end{bmatrix}
\quad\text{in column notation} \leftrightarrow
\begin{bmatrix}
a & d & g \\
b & e & h \\
c & f & i
\end{bmatrix}
\quad\text{in row notation}
\]

- Transpose often indicated, for a matrix \( M \), by \( M^T \)

- Again, the two types of notation are equivalent:

\[
\begin{bmatrix}
x & y & z
\end{bmatrix}
\begin{bmatrix}
a & d & g \\
b & e & h \\
c & f & i
\end{bmatrix}
= \begin{bmatrix}
ax + by + cz & dx + ey + fz & gx + hy + iz
\end{bmatrix}
\leftrightarrow
\begin{bmatrix}
ax + by + cz \\
dx + ey + fz \\
gx + hy + iz
\end{bmatrix}
\begin{bmatrix}
a & b & c \\
d & e & f \\
g & h & i
\end{bmatrix}
\]

- Different texts and graphics packages use different notations. Be careful!

Homogeneous Coordinate Example

- Translation matrix with row notation:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
= \begin{bmatrix}
x + dx \\
y + dy \\
z + dz
\end{bmatrix}
\]

- Translation matrix with column notation:

\[
\begin{bmatrix}
1 & 0 & 0 & dx \\
0 & 1 & 0 & dy \\
0 & 0 & 1 & dz \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
= \begin{bmatrix}
x + dx \\
y + dy \\
z + dz
\end{bmatrix}
\]

Matrix Notation and Composition

- Application of matrices in row notation is reverse of application in column notation:

\[
\begin{bmatrix}
1 & 0 & 0 & dx \\
0 & 1 & 0 & dy \\
0 & 0 & 1 & dz \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
\quad\text{matrices applied right to left}
\]

\[
\begin{bmatrix}
s_x & 0 & 0 & 0 \\
0 & s_y & 0 & 0 \\
0 & 0 & s_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\cos\theta & -\sin\theta & 0 & 0 \\
\sin\theta & \cos\theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
\quad\text{matrices applied left to right}
\]

Matrix Representation in Memory

- In order to use matrices in your program, you need to establish a standard way of storing the matrices in memory
  - this means you need to decide on a way of ordering the elements in your matrix

- There are two common ways of doing this called row-major and column-major format
  - in row-major, the elements in a row are stored sequentially in memory from left to right, and the rows are stored top to bottom
  - in column-major, the elements in a column are stored sequentially in memory from top to bottom, and the columns are stored left to right

- Example: Given matrix

\[
\begin{bmatrix}
a & b & c \\
d & e & f \\
g & h & i
\end{bmatrix}
\]

  - row-major ordering: a, b, c, d, e, f, g, h, i
  - column-major ordering: a, d, g, b, e, h, c, f, i

- For your assignments, use column-major matrices
Finding a Matrix—Solutions (1/2)

- Problem (posed on p. 37): find matrices which perform transformations pictured below:

  a)
  b)

- Method 1: Use matrix forms to determine elements of the transformation matrix. For a) we can see that we need a scale matrix. We need to scale along the $y$-axis; in particular we need $y = 1.5$ to become $y = 3$, and $y = 2.5$ to become $y = 5$. The $x$ values do not change. So the matrix would be

$$
\begin{bmatrix}
  s_x & 0 \\
  0 & s_y
\end{bmatrix}
\begin{bmatrix}
  1 \\
  0
\end{bmatrix}
\begin{bmatrix}
  0 \\
  2
\end{bmatrix}
$$

- Method 2: Solve linear equations to find the matrix coefficients. We know that in general

$$
\begin{bmatrix}
  a & b \\
  c & d
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
= \begin{bmatrix}
  ax + by \\
  cx + dy
\end{bmatrix}
$$


Finding a Matrix—Solutions (2/2)

- We can substitute in a known vector and its transformed result to solve the equations. In this case, for instance, \( \begin{bmatrix} 2 \\ 1.5 \end{bmatrix} \) becomes \( \begin{bmatrix} 2 \\ 3 \end{bmatrix} \). Substituting these examples gives us

$$
\begin{bmatrix}
  a & b \\
  c & d
\end{bmatrix}
\begin{bmatrix}
  2 \\
  1.5
\end{bmatrix}
= \begin{bmatrix}
  a2 + b(1.5) \\
  c2 + d(1.5)
\end{bmatrix}
= \begin{bmatrix}
  2 \\
  3
\end{bmatrix}
$$

- Solve for two of the variables, say $a$ and $c$. Choose another known vector example, and substitute in to solve for the other two. Keep doing this until you get unambiguous results for $a$, $b$, $c$, and $d$.

- Works the easiest if points are of the form \( \begin{bmatrix} x \\ 0 \end{bmatrix} \) and \( \begin{bmatrix} 0 \\ y \end{bmatrix} \).