Computer Graphics - Week 2

Questions about Last Week?
**Administrivia**

**Books**
- Reading list has been forwarded to the bookstore, but ...
  - No success yet :-(
- Alternatives:
  - www.amazon.com
  - www.barnesnoble.com

**Minor adjustments to the schedule**
- Final assignment due on April 28

**Regularly check the class home page at**
- www.cs.columbia.edu/graphics

---

**Programming Environment**

**Gateway lab**
- SGI workstations with OpenGL and GLUT
- Account charge: $35
- Currently limited business hours ... should improve

**ACIS lab**
- HP workstation running OpenGL and GLUT
- No extra account charge

**Windows on PCs**
- Visual C++ Version 5 (6)
Overview of Week 2

Overview of the Raster Graphics Pipeline

Transformation
- Geometric transformations
- Homogeneous coordinates
- Composing transformations
- Hierarchical modeling

Raster Graphics Pipeline
Raster Graphics Pipeline: Geometry Stage

- **Modeling Transformation**
  - Create the scene: position objects in a common coordinate system
- **Viewing Transformation**
  - Setup and position camera
- **Lighting**
  - Setup lights
- **Clipping**
  - Remove portions of the scene outside of the view volume
- **Perspective & Viewport Mapping**
  - Transition from 3D to 2D
  - Map the image into screen / window coordinates

Raster Graphics Pipeline: Image Generation

- **Setup**
  - Calculate parameters for rasterization, e.g. line slopes, color gradients etc.
  - Convert floating point positions into fixed point raster coordinates
- **Rasterization**
  - Generate coordinates and values (color, depth etc.) for pixel covered by an object
- **Pixel Processing a.k.a. fragment processing**
  - Process and store pixels produced by rasterization process
  - E.g. z-buffer, texture mapping, alpha-blending or stenciling
Geometric Transformations: Overview

- Position, orient and size objects
  - Translation
  - Rotation
  - Scaling

Cartesian Coordinates

- "Common" rectilinear coordinate system
  - Usually in 1D, 2D and 3D
  - $n$ orthogonal, real number lines

- Right-handed coordinate system
  - Looking down the positive $z$-axis, counter-clockwise rotation will turn the $x$-axis into the $y$-axis.
Geometric Transformations: Translation

- Preserves lengths (isometric)
- Preserves angles (conformal)

\[ x' = x + a \]
\[ y' = y + b \]

Preserves lengths (isometric)
Preserves angles (conformal)

Geometric Transformations: Rotation

- Preserves lengths (isometric)
- Preserves angles (conformal)

\[ x' = x \cos \alpha - y \sin \alpha \]
\[ y' = x \sin \alpha + y \cos \alpha \]
Rigid Body Transformations

- **Rigid Body Transformations:**
  Transformations that preserve lengths and angles, i.e. that are isometric and conformal.

- Translation and rotation are rigid body transformations.

Geometric Transformations: Scaling

- Does not preserve lengths
- Preserves angles only for uniform scaling
Geometric Transformations: Shear

- Does not preserve lengths or angles

\[ x' = x + ay \]
\[ y' = y \]

Geometric Transformations: Matrix Form

- **Rotation**
  \[
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} = \begin{pmatrix}
  \cos \alpha & -\sin \alpha \\
  \sin \alpha & \cos \alpha
  \end{pmatrix}
  \begin{pmatrix}
  x \\
  y
  \end{pmatrix}
  \Rightarrow p' = R \cdot p
  
- **Shear**
  \[
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} = \begin{pmatrix}
  1 & a \\
  0 & 1
  \end{pmatrix}
  \begin{pmatrix}
  x \\
  y
  \end{pmatrix}
  \Rightarrow p' = S_{H_x} \cdot p
  
- **Scaling**
  \[
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} = \begin{pmatrix}
  a & 0 \\
  0 & b
  \end{pmatrix}
  \begin{pmatrix}
  x \\
  y
  \end{pmatrix}
  \Rightarrow p' = S \cdot p
  
- **Translation**
  \[
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} = \begin{pmatrix}
  a & 0 \\
  b & 1
  \end{pmatrix}
  \begin{pmatrix}
  x \\
  y
  \end{pmatrix}
  \Rightarrow p' = T \cdot p
Geometric Transformations

- Translations handled differently than other transformations
  \[ p' = R \cdot p \quad p' = S \cdot p \quad p' = SH_x \cdot p \quad p' = T + p \]

- Unelegant, cumbersome, clumsy!!
- Not possible to composite transformations into a single transformation
- Homogeneous coordinates solve this problem.

Homogeneous Coordinates: Definition

- Every 2D point is defined by three numbers
  \[ p = (x, y) \equiv P = (X, Y, W) \]
  \[ x = X / W \quad y = Y / W \quad \text{for} \quad W \neq 0 \]

- \( P \) is called the homogeneous coordinates of \( p \).
  - \( W \) is the homogeneous component, or simply the \( W \) coordinate
  - \( p \) is in Cartesian coordinates

- Conventions
  - Cartesian coordinates: lower-case letters
  - Homogeneous coordinates: upper-case letters
  - Matrices and vectors: bold-face
Homogeneous Coordinates: Properties

- A point in Cartesian coordinates has many corresponding homogeneous representations
  - The Cartesian point \( p \) lies in the plane \( W = 1 \).
  - Different values of \( W \) trace out a line from the origin through \( p \)

\[
p = (x, y) \equiv P = (tX, tY, tW) \quad \text{with} \quad t \neq 0
\]

So far, we have excluded homogeneous coordinates with \( W = 0 \).
- Cartesian coordinates are infinite.

\[
\lim_{W \to 0} \left( \frac{x}{y} \right) = \lim_{W \to 0} \left( \frac{X}{W} \right) = \left( \frac{Y}{W} \right) = \left( \frac{\infty}{0} \right)
\]

- Look at the limit of \( x/y \):
  \[
  \lim_{W \to 0} \frac{x}{y} = \lim_{W \to 0} \frac{X}{W} = \frac{X}{Y} = \frac{x}{y}
  \]

- Although the individual coordinates are infinite, their ratio remains finite!
- Coordinates with \( W = 0 \) represent
  - Points at infinity in a given direction
  - Vectors
Homogeneous Coordinates: Application

So how does all that help with describing translations?

\[
\begin{pmatrix}
X' \\
Y' \\
W'
\end{pmatrix} =
\begin{pmatrix}
1 & 0 & a \\
0 & 1 & b \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
1
\end{pmatrix} =
\begin{pmatrix}
x + a \\
y + b \\
1
\end{pmatrix}
\Rightarrow x' = x + a \quad ; \quad y' = y + b
\]

\[P' = T \cdot P\]

Homogeneous Coordinates: Other Uses

As we will see later, homogeneous coordinates are also useful for
- Projective geometry, e.g. perspective viewing
- Clipping
- Modeling, e.g. non-uniform rational B-splines (NURBS)
Geometric Transformations: Matrix Form w/ Homogeneous Coordinates

- **Rotation**
  \[
  \begin{pmatrix}
  X' \\
  Y' \\
  W'
  \end{pmatrix} = \begin{pmatrix}
  \cos \alpha & -\sin \alpha & 0 \\
  \sin \alpha & \cos \alpha & 0 \\
  0 & 0 & 1
  \end{pmatrix} \begin{pmatrix}
  x \\
  y \\
  1
  \end{pmatrix}
  \]
  \[\Rightarrow P' = R \cdot P\]

- **Shear**
  \[
  \begin{pmatrix}
  X' \\
  Y' \\
  W'
  \end{pmatrix} = \begin{pmatrix}
  1 & a & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
  \end{pmatrix} \begin{pmatrix}
  x \\
  y \\
  1
  \end{pmatrix}
  \]
  \[\Rightarrow P' = SH_x \cdot P\]

- **Scaling**
  \[
  \begin{pmatrix}
  X' \\
  Y' \\
  W'
  \end{pmatrix} = \begin{pmatrix}
  a & 0 & 0 \\
  0 & b & 0 \\
  0 & 0 & 1
  \end{pmatrix} \begin{pmatrix}
  x \\
  y \\
  1
  \end{pmatrix}
  \]
  \[\Rightarrow P' = S \cdot P\]

- **Translation**
  \[
  \begin{pmatrix}
  X' \\
  Y' \\
  W'
  \end{pmatrix} = \begin{pmatrix}
  1 & 0 & a \\
  0 & 1 & b \\
  0 & 0 & 1
  \end{pmatrix} \begin{pmatrix}
  x \\
  y \\
  1
  \end{pmatrix}
  \]
  \[\Rightarrow P' = T \cdot P\]

Homogeneous Coordinates: 3D Space

- **Applying the same concepts to 3D is straight-forward**
  - Cartesian coordinates \((x,y,z)\) correspond to homogeneous coordinates \((X,Y,Z,W)\).
  - The transformation matrices for the basic geometric transformations remain essentially the same:

  \[
  R_x = \begin{pmatrix}
  1 & 0 & 0 & 0 \\
  0 & \cos \alpha & -\sin \alpha & 0 \\
  0 & \sin \alpha & \cos \alpha & 0 \\
  0 & 0 & 0 & 1
  \end{pmatrix}
  \]
  \[
  R_y = \begin{pmatrix}
  \cos \alpha & 0 & \sin \alpha & 0 \\
  0 & 1 & 0 & 0 \\
  -\sin \alpha & 0 & \cos \alpha & 0 \\
  0 & 0 & 0 & 1
  \end{pmatrix}
  \]
  \[
  R_z = \begin{pmatrix}
  \cos \alpha & -\sin \alpha & 0 & 0 \\
  \sin \alpha & \cos \alpha & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
  \end{pmatrix}
  \]
  \[
  T = \begin{pmatrix}
  1 & 0 & 0 & a \\
  0 & 1 & 0 & b \\
  0 & 0 & 1 & c \\
  0 & 0 & 0 & 1
  \end{pmatrix}
  \]
  \[
  S = \begin{pmatrix}
  a & 0 & 0 & 0 \\
  0 & b & 0 & 0 \\
  0 & 0 & c & 0 \\
  0 & 0 & 0 & 1
  \end{pmatrix}
  \]
  \[
  SH_x = \begin{pmatrix}
  1 & a & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
  \end{pmatrix}
  \]
Types of Transformations

- **Linear Transformations**
  - Maps lines to lines (or point)
  - Origin is always mapped into the origin
  
  \[ x' = a_{11}x + a_{12}y + a_{13}z \]
  \[ y' = a_{21}x + a_{22}y + a_{23}z \]
  \[ z' = a_{31}x + a_{32}y + a_{33}z \]

- **Affine Transformations**
  - Preserves parallel lines
  - Origin does not always map to the origin
  
  \[ x' = a_{11}x + a_{12}y + a_{13}z + a_{14} \]
  \[ y' = a_{21}x + a_{22}y + a_{23}z + a_{24} \]
  \[ z' = a_{31}x + a_{32}y + a_{33}z + a_{34} \]

\[ \Rightarrow \]

\[
\begin{pmatrix}
  x' \\
  y' \\
  z'
\end{pmatrix} =
\begin{pmatrix}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  a_{31} & a_{32} & a_{33}
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix}
\]

\[
\begin{pmatrix}
  X' \\
  Y' \\
  Z' \\
  W'
\end{pmatrix} =
\begin{pmatrix}
  a_{11} & a_{12} & a_{13} & a_{14} \\
  a_{21} & a_{22} & a_{23} & a_{24} \\
  a_{31} & a_{32} & a_{33} & a_{34} \\
  a_{41} & a_{42} & a_{43} & a_{44}
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  z \\
  1
\end{pmatrix}
\]

Combining Transformations ... or "Why we went through all that trouble"

- **Compositing transformations means to multiply the individual transformation matrices**

\[ M = M_1 \cdot M_2 \]

\[ P' = M \cdot P \]

\[ = (M_1 \cdot M_2) \cdot P \]

\[ = M_1 \cdot (M_2 \cdot P) \]

- **Notes:**
  - Matrix multiplication is **not** commutative
  - However, it is associative
  - \( M_2 \) is applied first!
  - Consider this when setting up transformations in your programs!

- Multiplying transformation matrices applies only to homogeneous coordinates
Combining Transformations: Example

1) Translate (-3, -2)
2) Rotate (30)
3) Translate (6, 4)

1) Rotate (30)
2) Translate (-3, -2)
3) Translate (6, 4)

Compositing of Transformations: Finding a Transformation Matrix

How are unit vectors and origin affected?

\[
\begin{pmatrix}
  a & b & c & d \\
  e & f & g & h \\
  i & j & k & l \\
  m & n & o & p
\end{pmatrix}
\begin{pmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  a & b & c & d \\
  e & f & g & h \\
  i & j & k & l \\
  m & n & o & p
\end{pmatrix}
\]

x-axis
y-axis
z-axis
origin

Identity matrix
Compositing of Transformations:
Finding a Transformation Matrix - Example

What is the transformation?

Origin: (0,0) goes to (8,4)
X-axis: (1,0) goes to (-0.707, -0.707)
Y-axis: (0,1) goes to (-0.707, 0.707)

\[
\begin{bmatrix}
a & b & c \\
d & e & f \\
g & h & i \\
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
= \begin{bmatrix}
-\sqrt{0.5} & -\sqrt{0.5} & 8 \\
-\sqrt{0.5} & +\sqrt{0.5} & -
\end{bmatrix}
\begin{bmatrix}
2 & 6 & 6 & 4 \\
1 & 1 & 3 & 5 \\
1 & 1 & 1 & 1 \\
\end{bmatrix}
= \begin{bmatrix}
8 - 3\sqrt{0.5} & 8 - 7\sqrt{0.5} & 8 - 9\sqrt{0.5} & 8 - 9\sqrt{0.5} \\
4 - \sqrt{0.5} & 4 - 5\sqrt{0.5} & 4 - 3\sqrt{0.5} & 4 + \sqrt{0.5} \\
1 & 1 & 1 & 1 \\
\end{bmatrix}
= \begin{bmatrix}
5.88 & 3.05 & 1.64 & 1.64 \\
3.29 & 0.46 & 1.88 & 4.71 \\
1 & 1 & 1 & 1 \\
\end{bmatrix}
\]
Geometric Transformations: Transforming Coordinate the System

- Transformations can also be considered as change of coordinate system
  - Origin and axes are remapped

- Object positions can be specified in either coordinate system.
  - Object's lower-left corner is either (2, 1) or (6, 3.8).

An Aside: Vector and Matrix Notation

- We will be using column format for vectors
  - Vector is multiplied from the left with the matrix $M_C$
  - Matrices are applied right to left

- Vectors can also be represented in row format
  - Vector is multiplied from the right with the matrix $M_R$
  - $M_R = M_C^T$ ... of course you remember that from linear algebra
  - Matrices are applied left to right

\[
\begin{pmatrix} X' \\ Y' \\ Z' \\ W' \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix} = M \cdot P \quad P^T (X' \ Y' \ Z' \ W') = (X \ Y \ Z \ W) \begin{pmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{pmatrix} = P^T \cdot M^T
\]
Putting it all to work:
Hierarchical Modeling

Hierarchical Modeling: Motivation

► Scenes
  ● defined in world coordinate system
  ● composed from different models
  ● Example: City

► Models
  ● defined in model coordinate system
  ● built from different parts
  ● Example: House, Roads, Trees, ...

► Parts
  ● defined in another model coordinates system
  ● built from yet more parts
  ● Example: Doors, Windows, Walls, ...
**Hierarchical Modeling: Scene Graph**

- Scene graph is a directed acyclic graph (DAG)
  - Objects defined in local coordinates
  - Transformation nodes to transform objects into world coordinates
  - Group nodes to create collection of objects
  - Attribute nodes to define object properties, e.g. material
  - Camera nodes to define viewing parameters

- Used in modern, high-level APIs
  - PHIGS
  - OpenInventor
  - VRML
  - Java3D

---

**Scene Graph: Example**

![Scene Graph Diagram](image-url)
Scene Graph Concepts: Instantiation

Objects can be used multiple times, i.e. they can be instantiated

- Groups can also be instantiated
- Example:
  - Object O2 is instantiated twice
  - Group G3 is instantiated twice (must be defined before use)
  - It is placed into the root coordinate system in different places defined by T1 and T2

Scene Graph Concepts: Transformations

Scene graph (implicitly) defines multiple coordinate systems

- Transformations are composited along given path
- Transformations only affect their child nodes
  - General concept: Any node has effect for its children.
  - Exception: VRML and OpenInventor
- Example:
  - Object O7 is transformed by T4*T5 into the root coordinate system
  - Object O6 is only affected by T4
Hierarchical Modeling: Transformations

- **Remember:**
  It is important in what order transformations are applied!
  - When applying a new transformation to an existing transformation, new matrix must by pre-multiplied.
  - When using row-vectors, matrices must be post-multiplied.

- **Example**
  - First T5
  - Then T4
  - T4 * T5 * O7

Hierarchical Modeling: Implementation

- **Stack** is used to save and restore matrices (coordinate systems) for different levels in the hierarchy
  - Compute current matrix
  - Push onto the stack
  - Matrix at the top of the stack is the active matrix
  - When done with this object,
    - Pop matrix
    - Return to previous (parent) matrix
Hierarchical Modeling: Coordinate Systems

- **Object or Model Coordinates**
  - Local coordinate system of an object
  - Chosen to conveniently describe the object
  - Examples:
    - Sphere has its center at the origin
    - Bolts in an airplane are described in units of millimeters

- **World Coordinates**
  - Coordinate system to define the entire scene
  - Chosen to conveniently describe the scene (world)
  - Examples:
    - Origin defined at the Southern tip of Manhattan
    - The airplane is described in units of meters (or worse inches)

- **Many intermediate coordinate systems**

Hierarchical Modeling: Animation

- **Animations describe changes in a model over time**
  - Characters are moving across the scene by moving their legs
  - Cars are driving down the street (wheels are spinning)
  - Birds are flapping their wings while flying through the air

- **Each motion has to be described by transformation of the proper object in the proper coordinate system**
  - Mistakes can lead to surprising results
  - Characters lose their legs, cars rotate around their wheels, etc.

- **Transformations are parameterized in time**
  - For example, position as a function of time
  - $x(t) = x(t-1) + 5$
Hierarchical Modeling: Animation

- Modeling a robot with body, head, arms and grabber
  - Each is described in its own, local coordinates
  - Pieced together using a scene graph
  - Consider moving the robot, lifting an arm, rotating the grabber, ...

Hierarchical Modeling: Animation
OpenGL

OpenGL: Introduction

- **OpenGL is a graphics system implementing a state machine**
  - First, the current state is defined by issuing state commands, e.g. defining the current color or current transformation.
  - Then, objects are processed using the current state.

- **OpenGL is (mostly) an immediate mode API**
  - Commands take immediate effect and are not collected
  - Small amount of internal buffering for efficiency
OpenGL: Rendering Commands

- Vertex-based rendering
- Meaning of vertices is defined by the current rendering primitive
- Rendering primitive defined by a `glBegin()` - `glEnd()` pair
- All vertices are subject to current state, e.g. transformation matrix or lighting conditions

For example: Drawing a triangle
- `glBegin(GL_TRIANGLES)` ;
  - `glVertex3f (0.0, 0.0, 0.5)` ;
  - `glVertex3f (0.0, 1.0, 0.5)` ;
  - `glVertex3f (1.0, 1.0, 0.5)` ;
- `glEnd ()` ;

OpenGL: Matrix Representation

- OpenGL matrices are represented as one-dimensional array of 16 numbers, e.g. `GLfloat m[16]`
- OpenGL stores the matrix in column-major form:
  \[
  M = \begin{pmatrix}
  m_0 & m_4 & m_8 & m_{12} \\
  m_1 & m_5 & m_9 & m_{13} \\
  m_2 & m_6 & m_{10} & m_{14} \\
  m_3 & m_7 & m_{11} & m_{15}
  \end{pmatrix}
  \]

- Memory layout of C arrays is row-major, i.e. different from OpenGL
  - `M[i][j]` refers to i-th column and j-th row !!
**OpenGL: Matrix Commands**

- **glMatrixMode()**
  - OpenGL manages 3 matrices: ModelView, Projection and Texture
  - We will only use ModelView and Projection matrices

- **glLoadIdentity()**
  - Makes the active matrix an identity matrix

- **glLoadMatrix()**
  - Assigns values to the active matrix

- **glMultMatrix()**
  - Post-multiplies current matrix with specified matrix

**OpenGL: Matrix Stacks**

- **glPushMatrix()**
  - Push current matrix onto the active matrix stack

- **glPopMatrix()**
  - Pop the top matrix and make it the current matrix
OpenGL: Modeling Transformations

- **glTranslate()**
  - Translation of an object along a given vector

- **glRotate()**
  - Rotation of an object ccw around a given axis by a specified angle

- **glScale()**
  - Scale an object independently along x, y, and z axis

Summary

- **Raster Graphics Pipeline**
  - Overview and basic understanding of major components

- **Geometric transformations and hierarchical modeling**
  - Elementary transformations
  - Rigid body, linear, affine transformations
  - Homogeneous coordinates
  - Hierarchical modeling
  - Basic animations

- If you had any problems with this material ...  
  Brush up on your basic geometry and linear algebra !!
Homework

- Foley et al., Chapter 6
- Start reading OpenGL Programming Guide

Next week ...

- More on transformations and coordinates systems
  - Viewing transformations and camera models
- More on OpenGL
- First Assignment

\[
\text{VP} = \text{M}_{\text{proj}} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ -1/d \end{pmatrix} \Rightarrow \text{vp} = \begin{pmatrix} 0 \\ 0 \\ -d \end{pmatrix}
\]