CSCI-4530/6530
Advanced Computer Graphics
http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S09/

Barb Cutler
cutler@cs.rpi.edu
MRC 309A

Topics for the Semester

• Meshes
  – representation
  – simplification
  – subdivision surfaces
  – generation
  – volumetric modeling

• Simulation
  – particle systems
  – rigid body, deformation, cloth, wind/water flows
  – collision detection
  – weathering

• Rendering
  – ray tracing
  – appearance models
  – shadows
  – local vs. global illumination
  – radiosity, photon mapping, subsurface scattering, etc.

• procedural modeling
• texture synthesis
• hardware & more …

Mesh Simplification

Hoppe “Progressive Meshes” SIGGRAPH 1996

Mesh Generation & Volumetric Modeling

Cutler et al., “Simplification and Improvement of Tetrahedral Models for Simulation” 2004

Modeling – Subdivision Surfaces

Hoppe et al., “Piecewise Smooth Surface Reconstruction” 1994

Geri’s Game
Pixar 1997

Luxo Jr.
Pixar Animation Studios, 1986
Particle Systems

Star Trek: The Wrath of Khan 1982

Physical Simulation

• Rigid Body Dynamics
• Collision Detection
• Fracture
• Deformation

Müller et al., “Stable Real-Time Deformations” 2002

Fluid Dynamics

“Visual Simulation of Smoke” Fedkiw, Stam & Jensen SIGGRAPH 2001

Ray Casting

• For every pixel construct a ray from the eye
  – For every object in the scene
    • Find intersection with the ray
    • Keep the closest

Ray Tracing

• Shade (interaction of light and material)
• Secondary rays (shadows, reflection, refraction)

“An Improved Illumination Model for Shaded Display” Whitted 1980

Subsurface Scattering

Jensen et al., “A Practical Model for Subsurface Light Transport” 2001
**Appearance Models**

Introductions – Who are you?
- name
- year/degree
- graphics background (if any)
- research/job interests
- why you are taking this class
- something fun, interesting, or unusual about yourself

What is a Transformation?
- Maps points \((x, y)\) in one coordinate system to points \((x', y')\) in another coordinate system
  \[
  x' = ax + by + c \\
  y' = dx + ey + f
  \]
- For example, Iterated Function System (IFS):

Outline
- Course Overview
- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Orthographic & Perspective Projections
- Example: Iterated Function Systems (IFS)
- OpenGL Basics

Simple Transformations
- Can be combined
- Are these operations invertible?
  \textit{Yes, except scale = 0}

Syllabus & Course Website

http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S09/
- Which version should I register for?
  - CSCI 6530
    - 3 units of graduate credit
  - CSCI 4530
    - 4 units of undergraduate credit
  - same lectures, assignments, quizzes, & grading criteria
- Other Questions?
Transformations are used to:

- Position objects in a scene
- Change the shape of objects
- Create multiple copies of objects
- Projection for virtual cameras
- Describe animations

Rigid-Body / Euclidean Transforms

- Preserves distances
- Preserves angles

Similitudes / Similarity Transforms

- Preserves angles

Linear Transformations

- $L(p + q) = L(p) + L(q)$
- $L(ap) = aL(p)$

Affine Transformations

- Preserves parallel lines
Projective Transformations

- preserves lines

General (Free-Form) Transformation

- Does not preserve lines
- Not as pervasive, computationally more involved

Outline

- Course Overview
- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Orthographic & Perspective Projections
- Example: Iterated Function Systems (IFS)
- OpenGL Basics

How are Transforms Represented?

\[
\begin{align*}
x' &= ax + by + c \\
y' &= dx + ey + f
\end{align*}
\]

\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c \\ f \end{bmatrix}
\]

\[
p' = M p + t
\]

Homogeneous Coordinates

- Add an extra dimension
  - in 2D, we use 3 x 3 matrices
  - in 3D, we use 4 x 4 matrices
- Each point has an extra value, w

\[
\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}
\]

\[
p' = M p
\]

Translation in homogeneous coordinates

Affine formulation

\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c \\ f \end{bmatrix}
\]

\[
p' = M p + t
\]

Homogeneous formulation

\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
\]

\[
p' = M p
\]
Homogeneous Coordinates

- Most of the time $w = 1$, and we can ignore it

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- If we multiply a homogeneous coordinate by an affine matrix, $w$ is unchanged

Homogeneous Visualization

- Divide by $w$ to normalize (homogenize)
- $W = 0$?

\[
\begin{align*}
(0, 0, 1) &= (0, 0, 2) = \ldots \\
(7, 1, 1) &= (14, 2, 2) = \ldots \\
(4, 5, 1) &= (8, 10, 2) = \ldots
\end{align*}
\]

Point at infinity (direction)

Translate ($tx$, $ty$, $tz$)

- Why bother with the extra dimension? Because now translations can be encoded in the matrix!

\[
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
\]

Scale ($sx$, $sy$, $sz$)

- Isotropic (uniform) scaling: $sx = sy = sz$

\[
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
\]

Rotation

- About $z$ axis

\[
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
\]

$Z\text{Rotate}(\theta)$

Rotation

- About $(k_x, k_y, k_z)$, a unit vector on an arbitrary axis (Rodrigues Formula)

\[
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} kk(1-c)+c & kk(1-c)-ks & kk(1-c) \\ kk(1-c)+ks & kk(1-c)+c + ks & 0 \\ kk(1-c)-ks & kk(1-c)-ks & kk(1-c)+c \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
\]

where $c = \cos \theta \& s = \sin \theta$
Storage

- Often, \( w \) is not stored (always 1)
- Needs careful handling of direction vs. point
  - Mathematically, the simplest is to encode directions with \( w = 0 \)
  - In terms of storage, using a 3-component array for both direction and points is more efficient
  - Which requires to have special operation routines for points vs. directions

Outline

- Course Overview
- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Orthographic & Perspective Projections
- Example: Iterated Function Systems (IFS)
- OpenGL Basics

How are transforms combined?

Scale then Translate

\[
\begin{pmatrix}
1 & 0 & 3 \\
0 & 1 & 1 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
2 & 0 & 3 \\
0 & 2 & 1 \\
0 & 0 & 1
\end{pmatrix}
\]

Use matrix multiplication:

\[\begin{pmatrix}
1 & 0 & 3 \\
0 & 1 & 1 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 1
\end{pmatrix}
= \begin{pmatrix}
2 & 0 & 3 \\
0 & 2 & 1 \\
0 & 0 & 1
\end{pmatrix}\]

Caution: matrix multiplication is NOT commutative!

Non-commutative Composition

Scale then Translate:

\[
\begin{pmatrix}
1 & 0 & 3 \\
0 & 1 & 1 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
2 & 0 & 3 \\
0 & 2 & 1 \\
0 & 0 & 1
\end{pmatrix}
\]

Translate then Scale:

\[
\begin{pmatrix}
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 & 3 \\
0 & 1 & 1 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
2 & 0 & 6 \\
0 & 2 & 2 \\
0 & 0 & 1
\end{pmatrix}
\]

Outline

- Course Overview
- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Orthographic & Perspective Projections
- Example: Iterated Function Systems (IFS)
- OpenGL Basics
Orthographic vs. Perspective

- Orthographic
- Perspective

Simple Orthographic Projection

- Project all points along the $z$ axis to the $z = 0$ plane

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

Simple Perspective Projection

- Project all points along the $z$ axis to the $z = d$ plane, eyepoint at the origin:

\[
\begin{align*}
x' &= \frac{d \cdot x}{z/d} \\
y' &= \frac{d \cdot y}{z/d} \\
z' &= d
\end{align*}
\]

\[
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 1/d & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

Alternate Perspective Projection

- Project all points along the $z$ axis to the $z = 0$ plane, eyepoint at the $(0,0,-d)$:

\[
\begin{align*}
x' &= \frac{d \cdot x}{z + d} \\
y' &= \frac{d \cdot y}{z + d} \\
z' &= \frac{z + d}{d}
\end{align*}
\]

\[
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 1/d & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

In the limit, as $d \to \infty$...is simply an orthographic projection

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 1/d & 1
\end{bmatrix}
\]

Outline

- Course Overview
- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Orthographic & Perspective Projections
- Example: Iterated Function Systems (IFS)
- OpenGL Basics
Iterated Function Systems (IFS)

- Capture self-similarity
- Contraction (reduce distances)
- An attractor is a fixed point
  \[ A = \bigcup f_i(A) \]

Example: Sierpinski Triangle

- Described by a set of \( n \) affine transformations
- In this case, \( n = 3 \)
  - translate & scale by 0.5

Example: Sierpinski Triangle

```cpp
for "lots" of random input points \((x_0, y_0)\)
for \(j=0\) to \(\text{num\_iters}\)
    randomly pick one of the transformations
    \((x_{k+1}, y_{k+1}) = f_i(x_k, y_k)\)
display \((x_k, y_k)\)
```

Another IFS: The Dragon

3D IFS in OpenGL

- GL_POINTS
- GL_QUADS

Assignment 0: OpenGL Warmup

- Get familiar with:
  - C++ environment
  - OpenGL
  - Transformations
  - simple Vector & Matrix classes
- Have Fun!
- Will not be graded (but you should still do it and submit it!)
Outline

• Course Overview
• Classes of Transformations
• Representing Transformations
• Combining Transformations
• Orthographic & Perspective Projections
• Example: Iterated Function Systems (IFS)
• OpenGL Basics

OpenGL Basics: GL_POINTS

```c
glDisable(GL_LIGHTING);
glBegin(GL_POINTS);
  glColor3f(0.0,0.0,0.0);
  glVertex3f(...);
glEnd();
```

• lighting should be disabled...

OpenGL Basics: GL_QUADS

```c
glEnable(GL_LIGHTING);
glBegin(GL_QUADS);
  glNormal3f(...);
  glColor3f(1.0,0.0,0.0);
  glVertex3f(...);
  glVertex3f(...);
  glVertex3f(...);
  glVertex3f(...);
  glEnd();
```

• lighting should be enabled...
• an appropriate normal should be specified

OpenGL Basics: Transformations

• Useful commands:
  ```c
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  glPopMatrix();
  glMultMatrixf(...);
  ```

Questions?

For Next Time:

• Read Hugues Hoppe “Progressive Meshes” SIGGRAPH 1996
• Post a comment or question on the course WebCT/LMS discussion by 10am on Friday 1/15