CSCI-4972/6963
Advanced Computer Graphics
http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S08/

Professor 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
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” Foster & Mataxas, 1996

“Visual Simulation of Smoke” Foster & Mataxas, 1996

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

Henrik Wann Jensen

Syllabus & Course Website

http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S08/

• Which version should I register for?
  - CSCI 6963
    • 3 units of graduate credit
  - CSCI 4972
    • 4 units of undergraduate credit
  - same lectures, assignments, quizzes, & grading criteria
• Other Questions?

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

Outline

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

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):

Simple Transformations

• Can be combined
• Are these operations invertible?
  Yes, except scale = 0
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

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

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?

\[
x' = ax + by + c \\
y' = dx + ey + f
\]

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

\[
p' = M p + t
\]

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

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

\[
p' = M p
\]

Translation in homogeneous coordinates

\[
x' = ax + by + c \\
y' = dx + ey + f
\]

Affine formulation

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

\[
p' = M p + t
\]

Homogeneous formulation

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

\[
p' = M p
\]
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)

\[
\begin{array}{c}
(0, 0, 1) = (0, 0, 2) = \\
(7, 1, 1) = (14, 2, 2) = \\
(4, 5, 1) = (8, 10, 2) = \\
\end{array}
\]

- $W = 0$?

Translate ($t_x$, $t_y$, $t_z$)

- 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 ($s_x$, $s_y$, $s_z$)

- Isotropic (uniform) scaling: $s_x = s_y = s_z$

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

Rotation

- About z axis

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

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

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

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

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

Simple Perspective Projection

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

$$
\begin{pmatrix}
 x_p \\
 y_p \\
 z_p
\end{pmatrix} =
\begin{pmatrix}
\frac{d-x}{z/d} \\
\frac{d-y}{z/d} \\
\frac{d-z}{z/d}
\end{pmatrix}
$$

Alternate Perspective Projection

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

$$
\begin{pmatrix}
 x_p \\
 y_p \\
 z_p
\end{pmatrix} =
\begin{pmatrix}
\frac{d-x}{z/d+1} \\
\frac{d-y}{z/d+1} \\
\frac{d-z}{z/d+1}
\end{pmatrix}
$$

In the limit, as $d \to \infty$

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

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

```plaintext
glDisable(GL_LIGHTING);
glBegin(GL_POINTS);
COLOR3F(0.0,0.0,0.0);
glVertex3f(...);
glEnd();
```

- lighting should be disabled...

OpenGL Basics: GL_QUADS

```plaintext
glEnable(GL_LIGHTING);
glBegin(GL_QUADS);
COLOR3F(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:
  ```plaintext
  glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glPopMatrix();
glMultMatrixf(...);
  ```

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