CSCI-4530/6530
Advanced Computer Graphics

http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S23/

Barb Cutler
cutler@cs.rpi.edu

*Luxo Jr.*, Pixar Animation Studios, 1986
Luxo Jr., Pixar Animation Studios, 1986

Topics for the Semester

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

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

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

• procedural modeling
• texture synthesis
• non-photorealistic rendering
• 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”
Fedkiw, Stam & Jensen
SIGGRAPH 2001

Foster & Mataxas, 1996
Ray Casting/Tracing

- For every pixel
  - Construct a ray from the eye
  - For every object in the scene
    - Find intersection with the ray
    - Keep the closest
- Shade (interaction of light and material)
- Secondary rays (shadows, reflection, refraction)

Appearance Models

[Images of different materials and surfaces, with annotations on lighting angles and materials by Wojciech Matusik and Henrik Wann Jensen]
Subsurface Scattering

Jensen et al.,
“A Practical Model for Subsurface Light Transport”
SIGGRAPH 2001

Syllabus & Course Website

http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S23/

• Which version should I register for?
  CSCI 6530 : 4 units of graduate credit
  CSCI 4530 : 4 units of undergraduate credit

• This is an intensive course aimed at graduate students and undergraduates interested in graphics research, involving significant reading & programming each week. Taking this course in a 5 course / overload semester is discouraged.
Grades

http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S23/

• This course counts as “communications intensive” for undergraduates. As such, you must satisfactorily complete all readings, presentations, project reports to pass the course.

• As this is an elective (not required) course, I expect to grade this course: “A”, “A-”, “B+”, “B”, “B-”, or “F”
  – Don’t expect C or D level work to “pass”
  – I don’t want to give any “F”s

Lecture Attendance/Participation

http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S23/

• Lecture will be discussion-intensive
  – We will discuss research papers
  – We will do worksheets in groups of 2 or 3

• You are expected to regularly attend and participate during in person lectures
  – Recorded lectures from a prior term will be recorded & posted on the calendar.
  – If illness or other appropriate absence force you to miss more than 2 lectures throughout the term, a formal excused absence will be required.
Outline

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

*Rigid / Euclidean*

Translation  Identity  Rotation
Similitudes / Similarity Transforms

- Preserves angles

Rigid / Euclidean
- Translation
- Identity
- Rotation
- Isotropic Scaling

Similitudes

Linear Transformations

Scaling
Reflection
Shear

Similitudes
Rigid / Euclidean
- Translation
- Identity
- Rotation
- Isotropic Scaling

Linear
- Scaling
- Reflection
- Shear

\[ L(p + q) = L(p) + L(q) \]
\[ L(ap) = a L(p) \]
Affine Transformations

• preserves parallel lines

Projective Transformations

• preserves lines
General (Free-Form) Transformation

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

Sederberg and Parry, Siggraph 1986

Outline

- Course Overview
- Classes of Transformations
  - Representing Transformations
- Combining Transformations
- Orthographic & Perspective Projections
- Example: Iterated Function Systems (IFS)
How are Transforms Represented?

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

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

\[ p' = Mp + t \]

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

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

\[ p' = Mp \]
Translation in homogeneous coordinates

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

Affine formulation

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

Homogeneous formulation

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

\[ p' = Mp + t \]

Homogeneous Coordinates

• Most of the time \( w = 1 \), and we can ignore it

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

• If we multiply a homogeneous coordinate by an affine matrix, \( w \) is unchanged
Homogeneous Visualization

- Divide by w to normalize (homogenize)
- \( W = 0? \)  *Point at infinity (direction)*

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

Translate \((tx, ty, tz)\)

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

\[
\begin{pmatrix}
x' \\
y' \\
z' \\
1
\end{pmatrix} =
\begin{pmatrix}
1 & 0 & 0 & tx \\
0 & 1 & 0 & ty \\
0 & 0 & 1 & tz \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix}
\]
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**

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

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

\[
\begin{bmatrix}
  x' \\
y' \\
z' \\
1
\end{bmatrix} =
\begin{bmatrix}
  k_xk_x(1-c)+c & k_xk_y(1-c)-k_zs & k_xk_z(1-c)+k_y s & 0 \\
k_yk_x(1-c)+k_zs & k_yk_y(1-c)+c & k_yk_z(1-c)-k_zs & 0 \\
k_zk_x(1-c)-k_ys & k_zk_y(1-c)-k_zs & k_zk_z(1-c)+c & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

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)

How are transforms combined?

Scale then Translate

Use matrix multiplication:  \( p' = T \left(S \cdot p\right) = TS \cdot p \)

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

Caution: matrix multiplication is NOT commutative!
Non-commutative Composition

Scale then Translate: \( p' = T( S p) = TS p \)

Translate then Scale: \( p' = S( T p) = ST p \)

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

\[
ST = \begin{pmatrix}
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 & 3 \\
0 & 1 & 1 \\
0 & 0 & 1
\end{pmatrix}
= \begin{pmatrix}
2 & 0 & 6 \\
0 & 2 & 2 \\
0 & 0 & 1
\end{pmatrix}
\]
Worksheet!

Write down the 3x3 matrix that transforms this set of 4 points:

A: (0,0)          B: (1,0)          C: (1,1)          D: (0,1)

to these new positions:

A’: (-1, 1)       B’: (-1, 0)       C’: (0, 0)        D’: (0, 1)

Show your work.

If you finish early…
Solve the problem using a different technique.

Outline

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

- Orthographic

- Perspective

Simple Orthographic Projection

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

\[
\begin{pmatrix}
    x \\
    y \\
    0 \\
    1
\end{pmatrix} = \begin{pmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 0 & 0 \\
    0 & 0 & 0 & 0 \\
    0 & 0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
    x \\
    y \\
    z \\
    1
\end{pmatrix}
\]
Simple Perspective Projection

• Project all points along the z axis to the \( z = d \) plane, eyepoint at the origin:

By similar triangles:
\[
x'/x = d/z \\
x' = (x*d)/z
\]

homogenize

\[
\begin{pmatrix}
x * d/z \\
y * d/z \\
d \\
1 
\end{pmatrix}
= \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix}
= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}
\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}
\]

Alternate Perspective Projection

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

By similar triangles:
\[
x'/x = d/(z+d) \\
x' = (x*d)/(z+d)
\]

homogenize

\[
\begin{pmatrix}
x * d / (z + d) \\
y * d / (z + d) \\
0 \\
1 
\end{pmatrix}
= \begin{pmatrix} x \\ y \\ 0 \\ (z + d)/d \end{pmatrix}
= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix}
\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}
\]
In the limit, as $d \to \infty$

this perspective projection matrix...

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

...is simply an orthographic projection

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

Outline

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

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
Assignment 0: OpenGL/Metal Warmup

• Get familiar with:
  – C++ environment
  – OpenGL / Metal
  – Transformations
  – simple Vector & Matrix classes

• Have Fun!
• Due ASAP (start it today!)
• ¼ of the points of the other HWs
  (but you should still do it and submit it!)
Questions?

Image by Henrik Wann Jensen

For Next Time:

- Read Hugues Hoppe “Progressive Meshes” SIGGRAPH 1996
- Everyone will a comment or question on the course Submitty discussion forum before 10am on Friday

We need 2 volunteers to be “Discussants”
Note: This is not a “presentation”. Don’t make slides!
Be sure to read blurb (& linked webpage) on course webpage about Assigned Readings & Discussants.

(a) Base mesh $M^0$ (150 faces)  (b) Mesh $M^{15}$ (500 faces)  (c) Mesh $M^{35}$ (1,000 faces)  (d) Original $\tilde{M}=M^0$ (13,546 faces)
Questions to think about:

• How do we represent meshes?
• How to automatically decide what parts of the mesh are important / worth preserving?
• Algorithm performance: memory, speed?
• What were the original target applications? Are those applications still valid? Are there other modern applications that can leverage this technique?