CSCI-4530/6530
Advanced Computer Graphics

http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S12/

Barb Cutler
cutler@cs.rpi.edu
MRC 331A

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

Geri’s Game
Pixar 1997

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

Foster & Mataxas, 1996

Appearance Models

Wojciech Matusik

Subsurface Scattering

Jensen et al., “A Practical Model for Subsurface Light Transport” 2001
**Syllabus & Course Website**

http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S12/

- Which version should I register for?
  - CSCI 6530: 3 units of graduate credit, class ends at 3:20
  - CSCI 4530: 4 units of undergraduate credit, class ends at 3:50
  (same lectures, assignments, quizzes, & grading criteria)

- 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.

- Other Questions?

**Participation/Laptops in Class Policy**

- Use of laptops for reference during paper discussion and general note-taking is allowed

- **Participation is 15% of your grade:**
  So, if your focus is mostly on your laptop and you rarely speak up in class, you will get a zero for participation

- If you are likely to be distracted by your laptop (email, web-surfing, games), close the lid 😊

**Outline**

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

**Introductions – Who are you?**

- name
- year/degree
- graphics background (if any)
- research/job interests, future plans
- 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):

<table>
<thead>
<tr>
<th>CSCI 4530/6530 Advanced Computer Graphics</th>
<th>CSCI 4973 Introduction to Visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Established course</td>
<td>New course</td>
</tr>
<tr>
<td>traditional, technical lectures</td>
<td>will be different than Fall 2010 offering</td>
</tr>
<tr>
<td>instructor provides most of the content</td>
<td>instructor provides some of the content</td>
</tr>
<tr>
<td>lots of in class discussion</td>
<td>students provide some of the content</td>
</tr>
<tr>
<td>read 2 research papers a week</td>
<td>lots of in class discussion</td>
</tr>
<tr>
<td>Structured individual homeworks</td>
<td>some in class work time</td>
</tr>
<tr>
<td>lots of programming</td>
<td></td>
</tr>
<tr>
<td>flexibility only in extra credit</td>
<td></td>
</tr>
<tr>
<td>5 week final project</td>
<td></td>
</tr>
<tr>
<td>teams of 2 encouraged</td>
<td></td>
</tr>
<tr>
<td>topic of your choice</td>
<td></td>
</tr>
<tr>
<td>lots of graphics-related programming</td>
<td></td>
</tr>
<tr>
<td>4 units of credit (3 for grad version)</td>
<td></td>
</tr>
<tr>
<td>Counts as a &quot;CS option&quot; for CS majors</td>
<td></td>
</tr>
<tr>
<td>Huge time commitment</td>
<td></td>
</tr>
<tr>
<td>Prior graphics experience recommended</td>
<td>Passion for visual perfection recommended</td>
</tr>
</tbody>
</table>

13 14 15 16 17 18
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

- \( L(p + q) = L(p) + L(q) \)
- \( L(ap) = a \cdot L(p) \)
Affine Transformations

- preserves parallel lines

```
Affine Transformations
```

Projective Transformations

- preserves lines

```
Projective Transformations
```

General (Free-Form) Transformation

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

```
General (Free-Form) Transformation
```

How are Transforms Represented?

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

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

\[ p' = M p + t \]

Homogeneous 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

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

\[
\begin{bmatrix}
  x' \\
  y' \\
  w'
\end{bmatrix}
= \begin{bmatrix}
  m & n & o & p
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  w
\end{bmatrix}
\]

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

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

Affine formulation

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

Homogeneous formulation

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

\[ p' = M p + t \]

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)

Scale \((s_x, s_y, s_z)\)

• Isotropic (uniform) scaling: \( s_x = s_y = s_z \)

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

Translate \((t_x, t_y, t_z)\)

• Why bother with the extra dimension?

Because now translations can be encoded in the matrix!

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

Rotation

• About z axis

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

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

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

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: \(p' = T(S(p)) = TS(p)\)

Caution: matrix multiplication is NOT commutative!

Non-commutative Composition

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

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

\[
\begin{pmatrix}
  x \\
y \\
0 \\
1
\end{pmatrix} =
\begin{pmatrix}
  1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
0 \\
1
\end{pmatrix}
\]

Simple Perspective Projection

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

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

Alternate Perspective Projection

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

\[
\begin{pmatrix}
x' \\
y' \\
z'
\end{pmatrix} =
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1/d & 0
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
(z+d) \\
1
\end{pmatrix}
\]

In the limit, as \(d \to \infty\)

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

is simply an orthographic projection
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

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 \((x_k, y_k)\)

Increasing the number of iterations

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!
- Due ASAP (start it today!)
- ¼ of the points of the other HWs (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

- OpenGL is a “state machine”
- OpenGL has lots of finicky setup & execution function calls
  - omitting a function call, swapping the order of 2 function calls, or passing the “wrong” argument, can result in a blank screen, nothing happens/changes, craziness happens, bus error, seg fault, etc.
- Often there’s more than one way to do things
  - often one way is much faster than another
- What is possible depends on your hardware

OpenGL Basics: Array Buffer

- Some useful commands:
  - Store data in points array
  - Initial setup:
  - `glBindBuffer(GL_ARRAY_BUFFER, points_VBO);`
  - `glBufferData(GL_ARRAY_BUFFER, ..., points);`
  - `glColor3f(0,0,0);`
  - `glPointSize(1);`
  - `glEnableClientState(GL_VERTEX_ARRAY);`
  - `glVertexAttribPointer(...);`
  - `glDrawArrays(GL_POINTS, ...);`
  - `glDisableClientState(GL_VERTEX_ARRAY);`

OpenGL Basics: Index Vertex Buffers

- Some useful commands:
  - Store data in verts & faces arrays:
  - `glBindBuffer(GL_ARRAY_BUFFER, cube_verts_VBO);`
  - `glBufferData(GL_ARRAY_BUFFER, cube_verts, ...);`
  - `glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, cube_face_indices_VBO);`
  - `glBufferData(GL_ELEMENT_ARRAY_BUFFER, cube_face_indices, GL_STATIC_DRAW);`
  - `glEnableClientState(GL_VERTEX_ARRAY);`
  - `glVertexPointer(... BUFFER_OFFSET(0));`
  - `glEnableClientState(GL_NORMAL_ARRAY);`
  - `glNormalPointer(... BUFFER_OFFSET(12));`
  - `glEnableClientState(GL_COLOR_ARRAY);`
  - `glColorPointer(... BUFFER_OFFSET(24));`
  - `glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, cube_face_indices_VBO);`
  - `glDrawElements(GL_QUADS, ...);`
  - `glDisableClientState(GL_NORMAL_ARRAY);`
  - `glDisableClientState(GL_COLOR_ARRAY);`

OpenGL Basics: Transformations

- Useful commands:
  - `glMatrixMode(GL_MODELVIEW);`
  - `glPushMatrix();`
  - `glPopMatrix();`
  - `glMultMatrixf(…);`
Questions?

Image by Henrik Wann Jensen

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