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

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 & Matusik, 1996

Appearance Models

Wojtek 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/S14/

• 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)

• 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

• Lecture is intended to be discussion-intensive
• Laptops, tablet computers, smart phones, and other internet-connected devices are not allowed
  – Except during the discussion of the day's assigned paper: students may use their laptop/tablet to view an electronic version of the paper
  – Other exceptions to this policy are negotiable; please see the instructor in office hours

Introductions – Who are you?

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

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)

How are Transforms Represented?

\[
\begin{align*}
x' &= ax + by + c \\
y' &= dx + ey + f \\
\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} &= \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} + \begin{bmatrix} c \\ f \\ 1 \end{bmatrix} \\
p' &= Mp + t
\end{align*}
\]

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

\[
\begin{align*}
x' &= ax + by + c \\
y' &= dx + ey + f \\
z' &= iw + jk + l \\
w' &= mn + op + q \\
\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} &= \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}
\end{align*}
\]

Translation in homogeneous coordinates

\[
\begin{align*}
x' &= ax + by + c \\
y' &= dx + ey + f \\
\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} &= \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} + \begin{bmatrix} c \\ f \\ 1 \end{bmatrix} \\
p' &= Mp + t
\end{align*}
\]

Homogeneous Coordinates

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

\[
\begin{align*}
x' &= ax + by + c \\
y' &= dx + ey + f \\
z' &= iw + jk + l \\
w' &= mn + op + q \\
\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} &= \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}
\end{align*}
\]

• 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)

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

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

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

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

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

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

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

Rotation

- About z axis

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

ZRotate(\(\theta\))

Rotation

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

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

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)

How are transforms combined?

Scale then Translate

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

Caution: matrix multiplication is NOT commutative!

Non-commutative Composition

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

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

Orthographic vs. Perspective

• Orthographic

• Perspective
Simple Orthographic Projection

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

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

Simple Perspective Projection

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

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

Alternate Perspective Projection

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

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

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

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

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

Another IFS: The Dragon

3D IFS in OpenGL

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!)

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