Topics for the Semester

- **Mesures**
  - 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

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

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

Cutler et al., “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

Foster & Matusik, 1996

“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)

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/S13/

• 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

• 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)
• 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

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

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?

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

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

\[ p' = Mp + t \]

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

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

\[ p' = Mp \]

Translation in homogeneous coordinates

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

Affine formulation

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

Homogeneous formulation

\[ p' = Mp + t \]

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 \((t_x, t_y, t_z)\)

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

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

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

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

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

Rotation

- About \( z \) axis

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

Rotation

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

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

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

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

Caution: matrix multiplication is NOT commutative!

Non-commutative Composition

Scale then Translate: \[ p' = T( Sp) = TS p \]

Translate then Scale: \[ p' = S(Tp) = ST p \]

Orthographic vs. Perspective

• Orthographic

• Perspective
Simple Orthographic Projection

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

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

Simple Perspective Projection

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

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

Alternate Perspective Projection

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

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

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)
- OpenGL Basics

Iterated Function Systems (IFS)

- Capture self-similarity
- Contraction (reduce distances)
- An attractor is a fixed point

\[
A = \bigcup f_t(A)
\]
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

Assignment 0: OpenGL Warmup

- Get familiar with:
  - C++ environment
  - OpenGL
  - Transformations
  - simple Vector & Matrix classes
- Have Fun!
- Due ASAP (start it today!)
- \( \frac{1}{4} \) 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

3D IFS in OpenGL

For “lots” of random input points \((x_0, y_0)\)
for \(j=0 \) to num_iters
randomly pick one of the transformations
\((x_{i+1}, y_{i+1}) = f_i (x_i, y_i)\)
display \((x_i, y_i)\)
Increasing the number of iterations
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 usually 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:

```c
/* store data in points array */
glGenBuffers(1, &points_VBO);
glBindBuffer(GL_ARRAY_BUFFER, points_VBO);
glBufferData(GL_ARRAY_BUFFER, ..., points);

glColor3f(0,0,0);
glPointSize(1);
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(...);
glEnableVertexAttribArray(...);
glVertexAttribPointer(...);
glDrawArrays(GL_POINTS, ...);

glDisableClientState(GL_VERTEX_ARRAY);
glDisableVertexAttribArray(...);
```

OpenGL Basics: Index Vertex Buffers

- Some useful commands:

```c
/* 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);
glDisableClientState(GL_VERTEX_ARRAY);
```

OpenGL Basics: Transformations

- Useful commands:

```c
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