CSCI-4530/6530 Advanced Computer Graphics

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

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

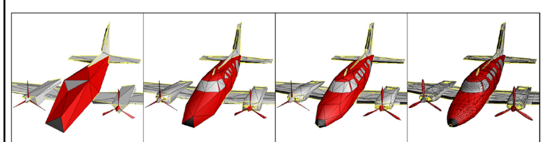
Luxo Jr.

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 …

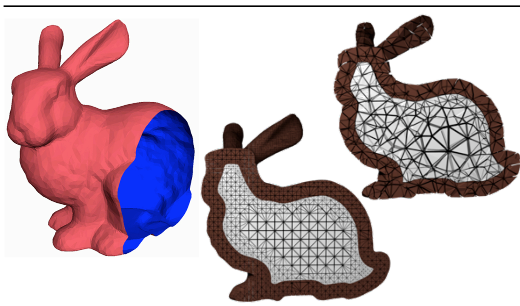
Mesh Simplification



(a) Base mesh M^0 (150 faces) (b) Mesh M^{175} (500 faces) (c) Mesh M^{425} (1,000 faces) (d) Original $\hat{M}=M^n$ (13,546 faces)

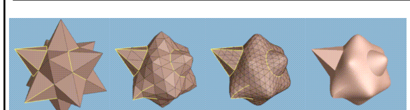
Hoppe "Progressive Meshes" SIGGRAPH 1996

Mesh Generation & Volumetric Modeling

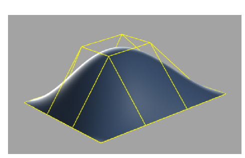


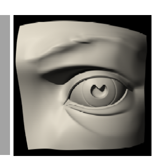
Cutler et al., "Simplification and Improvement of Tetrahedral Models for Simulation" 2004

Modeling – Subdivision Surfaces



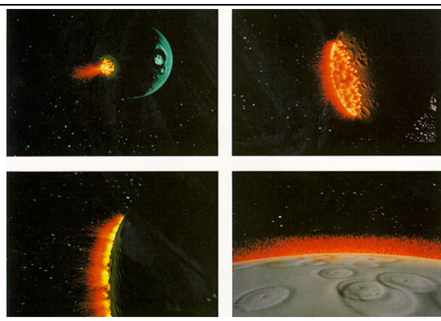
Hoppe et al., "Piecewise Smooth Surface Reconstruction" 1994





Geri's Game
Pixar 1997

Particle Systems

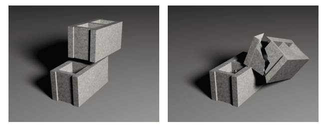


Star Trek: The Wrath of Khan 1982

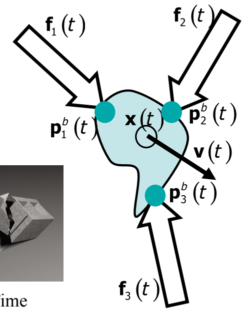
7

Physical Simulation

- Rigid Body Dynamics
- Collision Detection
- Fracture
- Deformation

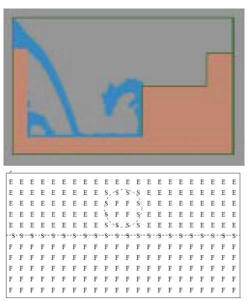


Müller et al., "Stable Real-Time Deformations" 2002



8

Fluid Dynamics



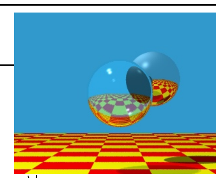
Foster & Mataxas, 1996

"Visual Simulation of Smoke"
Fedkiw, Stam & Jensen
SIGGRAPH 2001

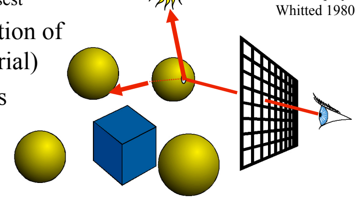
9

Ray Casting/Tracing

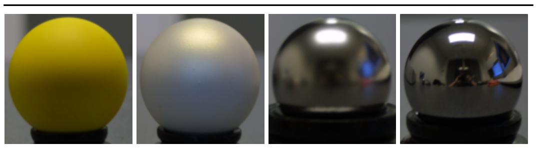
- For every pixel construct a ray from the eye
  - For every object in the scene
    - Find intersection with the ray
    - Keep the closest
- Shade (interaction of light and material)
- Secondary rays (shadows, reflection, refraction)



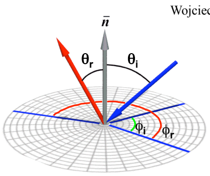
"An Improved Illumination Model for Shaded Display"
Whitted 1980

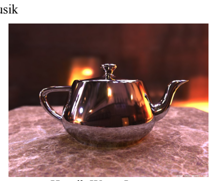


Appearance Models



Wojciech Matusik

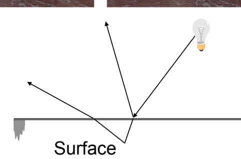




Henrik Wann Jensen

11

Subsurface Scattering



Jensen et al., "A Practical Model for Subsurface Light Transport" 2001

12

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?

13

CSCI 4530/6530 Advanced Computer Graphics / CSCI 4973 Introduction to Visualization

CSCI 4530/6530 Advanced Computer Graphics	CSCI 4973 Introduction to Visualization
Established course traditional, technical lectures instructor provides most of the content lots of in class discussion read 2 research papers a week	New course will be different than Fall 2010 offering instructor provides some of the content students provide some of the content lots of in class discussion some in class work time read 1 research paper a week
Structured individual homeworks lots of programming flexibility only in extra credit	Design-your-own homeworks design/art/creativity/thinking/revision/ presentation is focus some programming for implementation some fiddling with visualization toolkits individual & group work required
5 week final project teams of 2 encouraged topic of your choice lots of graphics-related programming	
4 units of credit (3 for grad version) Counts as a "CS option" for CS majors Huge time commitment	2 units of credit Counts only as "Free Elective" for CS majors (Probably) an unreasonable time commitment expected for a 2 credit course
Prior graphics experience recommended	Passion for visual perfection recommended

14

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 ☺

15

Introductions – Who are you?

- name
- year/degree
- graphics background (if any)
- research/job interests, future plans
- something fun, interesting, or unusual about yourself

16

Outline

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

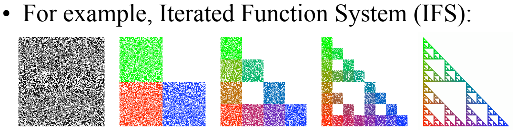
17

What is a Transformation?

- Maps points (x, y) in one coordinate system to points (x', y') in another coordinate system

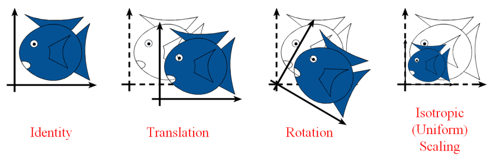
$$x' = ax + by + c$$
$$y' = dx + ey + f$$

- For example, Iterated Function System (IFS):



18

Simple Transformations

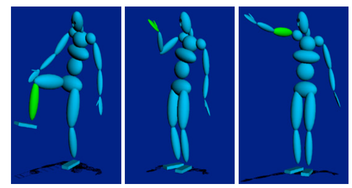


- Can be combined
- Are these operations invertible?

Yes, except scale = 0

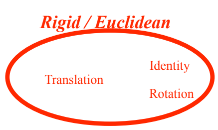
Transformations are used to:

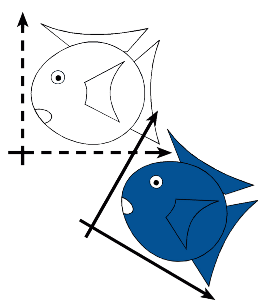
- Position objects in a scene
- Change the shape of objects
- Create multiple copies of objects
- Projection for virtual cameras
- Describe animations



Rigid-Body / Euclidean Transforms

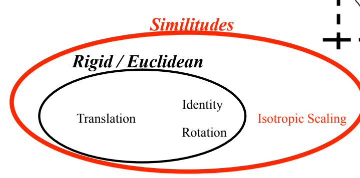
- Preserves distances
- Preserves angles

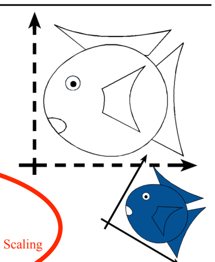




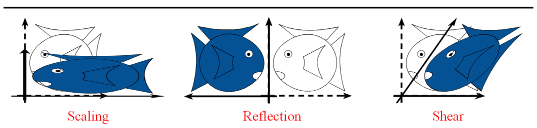
Similitudes / Similarity Transforms

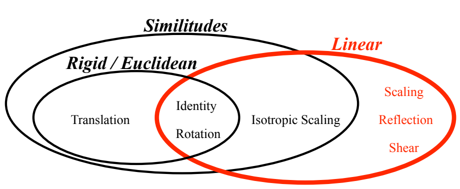
- Preserves angles





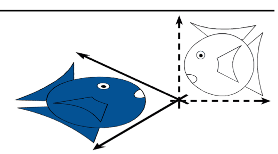
Linear Transformations

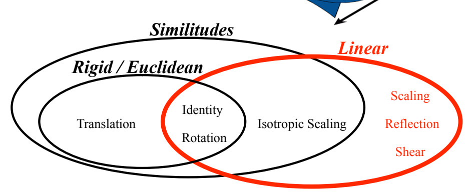




Linear Transformations

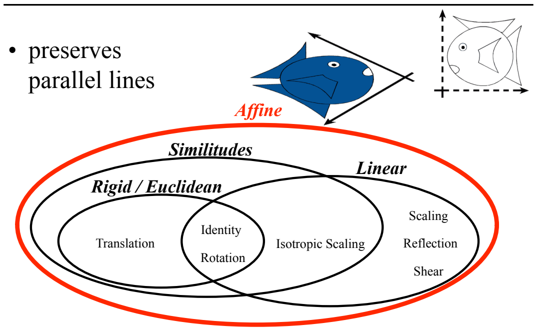
- $L(p + q) = L(p) + L(q)$
- $L(ap) = a\ L(p)$





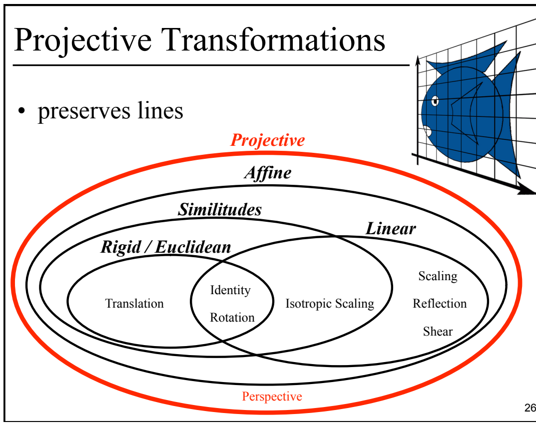
Affine Transformations

- preserves parallel lines



Projective Transformations

- preserves lines



General (Free-Form) Transformation

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

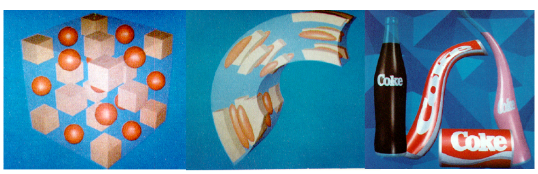


Fig 1. Undeformed Plastic

Fig 2. Deformed Plastic

Sederberg and Parry, Siggraph 1986

Outline

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

How are Transforms Represented?

$$x' = ax + by + c$$
$$y' = dx + ey + f$$

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

$$p' \;=\; M\,p \;+\; t$$

Homogeneous 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

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

$$p' = Mp + t$$

Homogeneous formulation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$p' = Mp$$

31

Homogeneous Coordinates

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

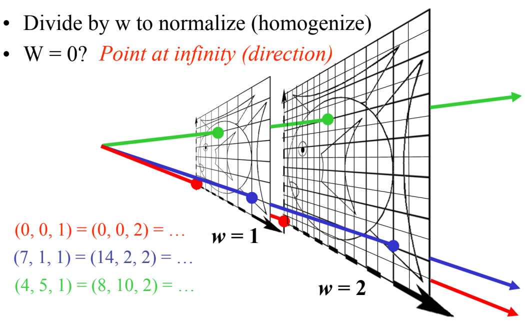
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- If we multiply a homogeneous coordinate by an *affine matrix*, w is unchanged

32

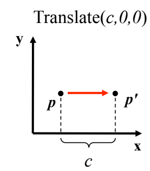
Homogeneous Visualization

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



33

Translate (tx, ty, tz)

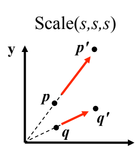


- Why bother with the extra dimension?
  Because now translations can be encoded in the matrix!

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

34

Scale (sx, sy, sz)

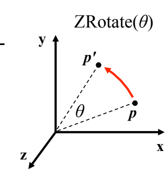


- Isotropic (uniform) scaling: $s_x = s_y = s_z$

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

35

Rotation



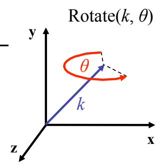
- About z axis

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

36

Rotation

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



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

where $c = \cos\theta$ & $s = \sin\theta$

Storage

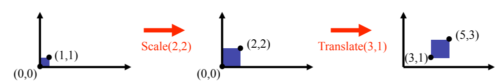
- Often, w is not stored (always 1)
- Needs careful handling of direction vs. point
  - Mathematically, the simplest is to encode directions with $w = 0$
  - In terms of storage, using a 3-component array for both direction and points is more efficient
  - Which requires to have special operation routines 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



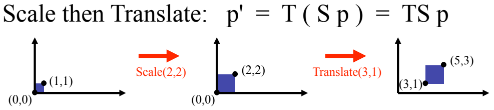
Use matrix multiplication: p' = T (S p) = TS p

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

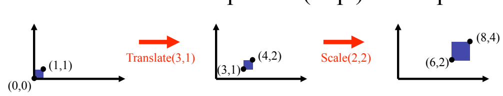
Caution: matrix multiplication is NOT commutative!

Non-commutative Composition

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



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



Non-commutative Composition

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

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

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

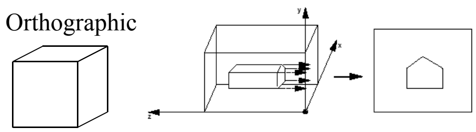
$$ST = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Outline

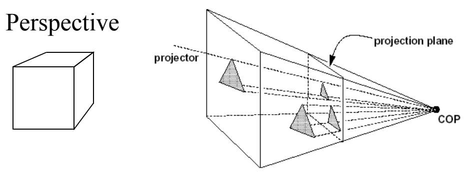
- Course Overview
- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Orthographic & Perspective Projections
- Example: Iterated Function Systems (IFS)
- OpenGL Basics

Orthographic vs. Perspective

- Orthographic

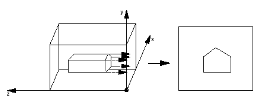


- Perspective



Simple Orthographic Projection

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

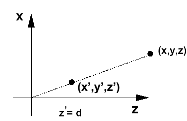


$$\begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

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



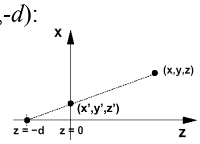
homogenize

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

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)



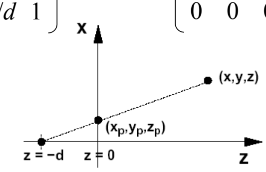
homogenize

$$\begin{pmatrix} x * d / (z + d) \\ y * d / (z + d) \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ (z + d) / d \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

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

this perspective projection matrix... ...is simply an orthographic projection

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



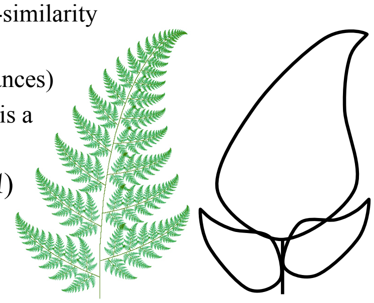
Outline

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

Iterated Function Systems (IFS)

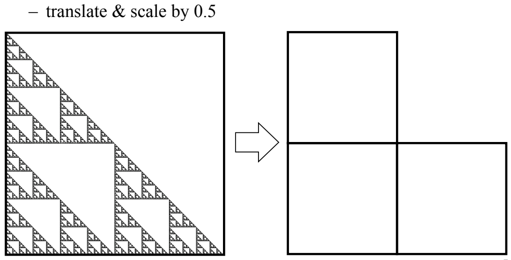
- Capture self-similarity
- Contraction (reduce distances)
- An attractor is a fixed point

$$A = \bigcup f_i(A)$$



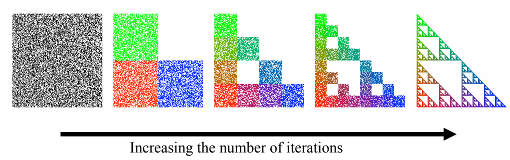
Example: Sierpinski Triangle

- Described by a set of n affine transformations
- In this case, $n = 3$
  - translate & scale by 0.5

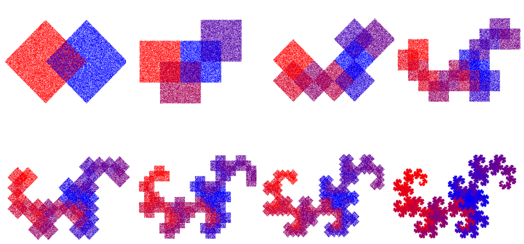


Example: Sierpinski Triangle

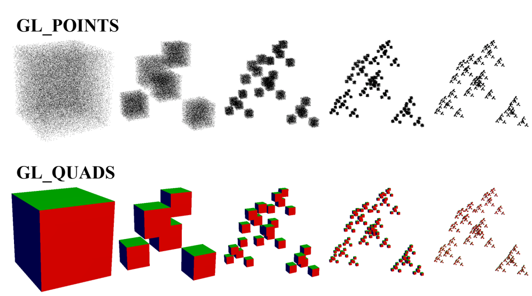
```
for "lots" of random input points (x0, y0)
   for j=0 to num_iters
      randomly pick one of the transformations
      (xk+1, yk+1) = fi (xk, yk)
   display (xk, yk)
```



Another IFS: The Dragon

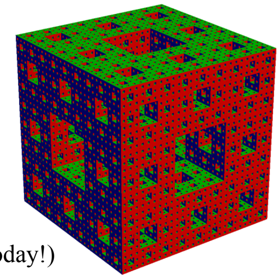


3D IFS in OpenGL



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!*)



55

Outline

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

56

OpenGL

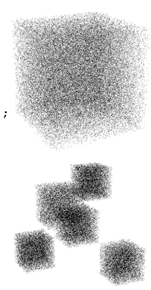
- OpenGL is a "state machine"
- OpenGL has lots of finicky setup & execution function calls
  - omitting a function call, swapping the order of 2 function calls, or passing the "wrong" argument, can result in a blank screen, nothing happens/changes, craziness happens, bus error, seg fault, etc.
- Often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

57

OpenGL Basics: Array Buffer

- **Some useful commands:**

```
/* 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(...);
```

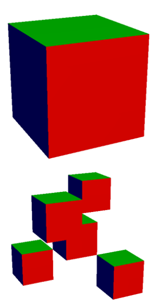


58

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);
glDisableClientState(GL_VERTEX_ARRAY);
```

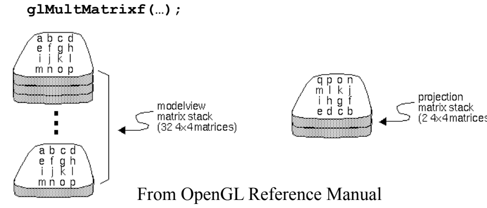


59

OpenGL Basics: Transformations

- Useful commands:

```
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glPopMatrix();
glMultMatrixf(…);
```



From OpenGL Reference Manual

60

Questions?

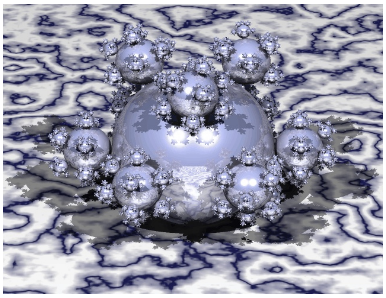
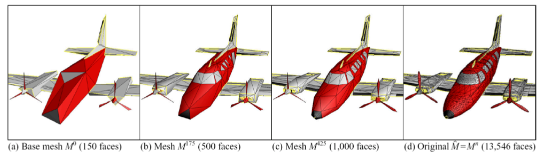


Image by Henrik Wann Jensen

61

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



(a) Base mesh M^0 (150 faces) (b) Mesh M^{175} (500 faces) (c) Mesh M^{425} (1,000 faces) (d) Original $\hat{M} = M^n$ (13,546 faces)

62