Computer Animation

CSE169: Computer Animation
Instructor: Steve Rotenberg
UCSD, Winter 2014
CSE169

Computer Animation Programming
Instructor: Steve Rotenberg (steve@graphics.ucsd.edu)
TA: Krishna Mullia (krishna@eng.ucsd.edu)
Lecture: EBU3 4140 (MW 5:00-6:20pm)
Office: EBU3 4150 (MW 4:00-5:00pm)
Lab: EBU3 basement
Discussion: TBD
Web page:
- http://graphics.ucsd.edu/courses/cse169_w14/index.html
Prerequisites

- CSE167 or equivalent introduction to computer graphics
- Familiarity with:
  - Vectors (dot products, cross products…)
  - Matrices (4x4 homogeneous transformations)
  - Polygon rendering
  - Basic lighting (normals, Gouraud, Phong…)
  - OpenGL, Direct3D, Java3D, or equivalent
  - C++ or Java
  - Object oriented programming
  - Basic physics
Undergraduate Computer Graphics at UCSD

- CSE 166: Image Processing
- CSE 167: Computer Graphics
- CSE 168: Rendering Algorithms
- CSE 169: Computer Animation
- CSE 125: Software Engineering (Game Project)
- Math 155B: Mathematics for Computer Graphics
Reading

- Papers
- Chapters
- Suggested book
  - 3D Computer Graphics: A Mathematical Introduction with OpenGL (Buss)
Programming Projects

- **Project 1: Due 1/22 (Week 3)**
  - Skeleton Hierarchy: Load a .skel file and display a 3D pose-able skeleton

- **Project 2: Due 2/3 (Week 5)**
  - Skin: Load .skin file and attach to the skeleton

- **Project 3: Due 2/19 (Week 7)**
  - Animation: Load .anim file and play back a key-framed animation on the skeleton

- **Project 4: Due 3/12 (Week 10)**
  - Choose one of the following
    - Cloth: Implement a simple cloth simulation
    - Inverse Kinematics: Implement an IK algorithm
    - Rigid Bodies: Implement a simple rigid body system with collisions
    - Choose your own project (but talk to me *first*)
Grading

- 15% Project 1
- 15% Project 2
- 15% Project 3
- 20% Project 4
- 15% Midterm
- 20% Final
Course Outline

1. 1/6: Introduction
2. 1/8: Skeletons
3. 1/13: Quaternions
4. 1/15: Skinning
5. 1/20: (Holiday)
6. 1/22: Facial Animation
7. 1/27: Channels & Keyframes
8. 1/29: Animation Blending
9. 2/3: Inverse Kinematics 1
10. 2/5: Inverse Kinematics 2
11. 2/10: TBD
12. 2/12: Midterm
13. 2/17: (Holiday)
14. 2/19: Particle Systems
15. 2/24: Cloth Simulation
16. 2/26: Collision Detection
17. 3/3: Rigid Body Physics 1
18. 3/5: Rigid Body Physics 2
19. 3/10: TBD
20. 3/12: Final Review
Who am I?

- Steve Rotenberg, Guest Lecturer at UCSD

- Teaching
  - Previously taught at UCSD from 2003-2009
  - Taught CSE169 from 2004-2009
  - Taught CSE167 a couple times

- Work History:
  - Angel Studios 1992-2002
  - PixelActive 2003-2010
  - NATVEQ 2010-2011
  - Nokia 2011-2013
Angel Studios

- Videos:
  - Peter Gabriel’s “Kiss That Frog”
  - Enertopia (stereoscopic IMAX)

- Games:
  - Midnight Club 1 & 2 (PS2, XBox)
  - Transworld Surf (PS2, XBox, GameCube)
  - Smuggler’s Run 1 & 2 (PS2, XBox, GameCube)
  - Midtown Madness 1 & 2 (PC)
  - Savage Quest (Arcade)
  - Test Drive Offroad: Wide Open (PS2)
  - N64 version of Resident Evil 2 (N64)
  - Ken Griffey Jr.’s Slugfest (N64)
  - Major League Baseball Featuring Ken Griffey Jr. (N64)

- Sold to Take Two Interactive (Rockstar) in November, 2002
PixelActive

Technology
- Main tech was ‘CityScape’, an interactive 3D city modeling tool
- Originally targeted to video game development
- Evolved for government, military, mapping, and urban planning

History
- Tech development began in early 2003
- Company incorporated in April 2006
- Sold to NAVTEQ in November 2010
- Merged into Nokia 2011
- Currently ‘Nokia Carlsbad’
Computer Animation Overview
Applications

- Special Effects (Movies, TV)
- Video Games
- Virtual Reality
- Simulation, Training, Military
- Medical
- Robotics, Animatronics
- Visualization
- Communication
Computer Animation

- Kinematics
- Physics (a.k.a. dynamics, simulation, mechanics)
- Character animation
- Artificial intelligence
- Motion capture / data driven animation
Animation Process

while (not finished) {
    MoveEverything();
    DrawEverything();
}

- Simulation vs. Animation
- Interactive vs. Non-Interactive
- Real Time vs. Non-Real Time
Character Rigging

- Skeleton
- Skin
- Facial Expressions
- Muscles
- Secondary motion: fat, hair, clothing...
Character Animation

- Keyframe Animation
- Motion Capture
- Inverse Kinematics
- Locomotion
- Procedural Animation
- Artificial Intelligence
Character Animation
Physics Simulation

- Particles
- Rigid bodies
  - Collisions, contact, stacking, rolling, sliding
- Articulated bodies
  - Hinges, constraints
- Deformable bodies (solid mechanics)
  - Elasticity, plasticity, viscosity
  - Fracture
  - Cloth
- Fluid dynamics
  - Fluid flow (liquids & gasses)
  - Combustion (fire, smoke, explosions…)
  - Phase changes (melting, freezing, boiling…)
- Vehicle dynamics
  - Cars, boats, airplanes, helicopters, motorcycles…
- Character dynamics
  - Body motion, skin & muscle, hair, clothing
Physics Simulation
Animation Software Tools

- Maya
- 3D Studio
- Lightwave
- Filmbox
- Blender

- Many more…
Animation Production Process

- Conceptual Design
- Production Design
- Modeling
- Materials & Shaders
- Rigging
- Blocking
- Animation
- Lighting
- Effects
- Rendering
- Post-Production
Resolution & Frame Rates

- **Video:**
  - NTSC: 720 x 480 @ 30 Hz (interlaced)
  - PAL: 720 x 576 @ 25 Hz (interlaced)

- **HDTV:**
  - 720p: 1280 x 720 @ 60 Hz
  - 1080i: 1920 x 1080 @ 30 Hz (interlaced)
  - 1080p: 1920 x 1080 @ 60 Hz

- **Film:**
  - 35mm: ~2000 x ~1500 @ 24 Hz
  - 70mm: ~4000 x ~2000 @ 24 Hz
  - IMAX: ~5000 x ~4000 @ 24-48 Hz

- **UHDTV, 4K, streaming standards…**

Note: Hz (Hertz) = frames per second (fps)

Note: Video standards with an i (such as 1080i) are *interlaced*, while standards with a p (1080p) are *progressive scan*
Interlacing

- Older video formats (NTSC, PAL) and some HD formats (1080i) use a technique called *interlacing*.
- With this technique, the image is actually displayed twice, once showing the odd *scanlines*, and once showing the even scanlines (slightly offset).
- This is a trick for achieving higher vertical resolution at the expense of frame rate (cuts effective frame rate in half).
- The two different displayed images are called *fields*.
- NTSC video, for example, is 720 x 480 at 30 *frames* per second, but is really 720 x 240 at 60 *fields* per second.
- Interlacing is an important issue to consider when working with video, especially in animation as in TV effects and video games.
- Computer monitors are generally not interlaced.
There are many ways to design a 3D renderer. The two most common approaches are:

- Traditional graphics pipeline
- Ray-based rendering

With the traditional approach, primitives (usually triangles) are rendered into the image one at a time, and complex visual effects often involve a variety of different tricks.

With ray-based approaches, the entire scene is stored and then rendered one pixel at a time. Ray based approaches can simulate light more accurately and offer the possibility of significant quality improvements, but with a large cost.

In this class, we will not be very concerned with rendering, as we will focus mainly on how objects move rather than how they look.
Coordinate Systems

- Right handed coordinate system
Vector Arithmetic

\[ \mathbf{a} = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix} \]

\[ \mathbf{b} = \begin{bmatrix} b_x & b_y & b_z \end{bmatrix} \]

\[ \mathbf{a} + \mathbf{b} = \begin{bmatrix} a_x + b_x & a_y + b_y & a_z + b_z \end{bmatrix} \]

\[ \mathbf{a} - \mathbf{b} = \begin{bmatrix} a_x - b_x & a_y - b_y & a_z - b_z \end{bmatrix} \]

\[ -\mathbf{a} = \begin{bmatrix} -a_x & -a_y & -a_z \end{bmatrix} \]

\[ s\mathbf{a} = \begin{bmatrix} sa_x & sa_y & sa_z \end{bmatrix} \]
Vector Magnitude

- The magnitude (length) of a vector is:

\[ |\mathbf{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2} \]

- A vector with length=1.0 is called a \textit{unit vector}

- We can also \textit{normalize} a vector to make it a unit vector:
Dot Product

\[ \mathbf{a} \cdot \mathbf{b} = \sum a_i b_i \]
\[ \mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y + a_z b_z \]
\[ \mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta \]
Dot Product

\[ \mathbf{a} \cdot \mathbf{b} = \sum a_i b_i \]

\[ \mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y + a_z b_z \]

\[ \mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta \]

\[ \mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b} \]

\[ \mathbf{a} \cdot \mathbf{b} = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \]
Example: Angle Between Vectors

How do you find the angle $\theta$ between vectors $\mathbf{a}$ and $\mathbf{b}$?
Example: Angle Between Vectors

\[ \mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta \]

\[ \cos \theta = \left( \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|} \right) \]

\[ \theta = \cos^{-1} \left( \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|} \right) \]
Dot Products with General Vectors

- The dot product is a scalar value that tells us something about the relationship between two vectors.
  - If $\mathbf{a} \cdot \mathbf{b} > 0$ then $\theta < 90^\circ$
  - If $\mathbf{a} \cdot \mathbf{b} < 0$ then $\theta > 90^\circ$
  - If $\mathbf{a} \cdot \mathbf{b} = 0$ then $\theta = 90^\circ$ (or one or more of the vectors is degenerate (0,0,0))
Dot Products with One Unit Vector

- If \(|\mathbf{u}| = 1.0\) then \(\mathbf{a} \cdot \mathbf{u}\) is the length of the projection of \(\mathbf{a}\) onto \(\mathbf{u}\)
Example: Distance to Plane

A plane is described by a point $p$ on the plane and a unit normal $n$. Find the distance from point $x$ to the plane.
Example: Distance to Plane

The distance is the length of the projection of $x-p$ onto $n$:

$$dist = (x - p) \cdot n$$
Dot Products with Unit Vectors

- $\mathbf{a} \cdot \mathbf{b} = 0$
- $0 < \mathbf{a} \cdot \mathbf{b} < 1$
- $-1 < \mathbf{a} \cdot \mathbf{b} < 0$
- $\mathbf{a} \cdot \mathbf{b} = 1$
- $\mathbf{a} \cdot \mathbf{b} = -1$

$|\mathbf{a}| = |\mathbf{b}| = 1.0$

$\mathbf{a} \cdot \mathbf{b} = \cos(\theta)$
Cross Product

\[ \mathbf{a} \times \mathbf{b} = \begin{vmatrix} i & j & k \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix} \]

\[ \mathbf{a} \times \mathbf{b} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix} \]
Properties of the Cross Product

\( \mathbf{a} \times \mathbf{b} \) is a vector perpendicular to both \( \mathbf{a} \) and \( \mathbf{b} \), in the direction defined by the right hand rule.

\[
|\mathbf{a} \times \mathbf{b}| = |\mathbf{a}| |\mathbf{b}| \sin \theta
\]

\[
|\mathbf{a} \times \mathbf{b}| = \text{area of parallelogram } \mathbf{ab}
\]

\[
|\mathbf{a} \times \mathbf{b}| = 0 \text{ if } \mathbf{a} \text{ and } \mathbf{b} \text{ are parallel}
\]
Example: Normal of a Triangle

- Find the unit length normal of the triangle defined by 3D points $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$.
Example: Normal of a Triangle

\[ n^* = (b - a) \times (c - a) \]

\[ n = \frac{n^*}{|n^*|} \]
Example: Area of a Triangle

- Find the area of the triangle defined by 3D points \(a, b,\) and \(c\)
Example: Area of a Triangle

\[
\text{area} = \frac{1}{2} \left| (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a}) \right|
\]
Example: Alignment to Target

- An object is at position $p$ with a unit length heading of $h$. We want to rotate it so that the heading is facing some target $t$. Find a unit axis $a$ and an angle $\theta$ to rotate around.
Example: Alignment to Target

\[ a = \frac{h \times (t - p)}{|h \times (t - p)|} \]

\[ \theta = \cos^{-1}\left(\frac{h \cdot (t - p)}{|(t - p)|}\right) \]
Vector Class

class Vector3 {
public:
  Vector3() {x=0.0f; y=0.0f; z=0.0f;}
  Vector3(float x0, float y0, float z0) {x=x0; y=y0; z=z0;}
  void Set(float x0, float y0, float z0) {x=x0; y=y0; z=z0;}
  void Add(Vector3 &a) {x+=a.x; y+=a.y; z+=a.z;}
  void Add(Vector3 &a, Vector3 &b) {x=a.x+b.x; y=a.y+b.y; z=a.z+b.z;}
  void Subtract(Vector3 &a) {x=a.x-b.x; y=a.y-b.y; z=a.z-b.z;}
  void Subtract(Vector3 &a, Vector3 &b) {x=a.x-b.x; y=a.y-b.y; z=a.z-b.z;}
  void Negate() {x=-x; y=-y; z=-z;}
  void Negate(Vector3 &a) {x=-a.x; y=-a.y; z=-a.z;}
  void Scale(float s) {x*=s; y*=s; z*=s;}
  void Scale(float s, Vector3 &a) {x=s*a.x; y=s*a.y; z=s*a.z;}
  float Dot(Vector3 &a) {return x*a.x+y*a.y+z*a.z;}
  void Cross(Vector3 &a, Vector3 &b) {x=a.y*b.z-a.z*b.y; y=a.z*b.x-a.x*b.z; z=a.x*b.y-a.y*b.x;}
  float Magnitude() {return sqrtf(x*x+y*y+z*z);}
  void Normalize() {Scale(1.0f/Magnitude());}

  float x, y, z;
};