Computer Animation

CSE169: Computer Animation
Instructor: Steve Rotenberg
UCSD, Winter 2009
CSE169

- Computer Animation Programming
- Instructor: Steve Rotenberg (steve@graphics.ucsd.edu)
- TA: Robert Thomas (r1thomas@ucsd.edu)
- Lecture: EBU3 2154 (MW 6:30-7:50pm)
- Office: ??? (MW 5:15-6:15pm)
- Lab: EBU3 basement
- Discussion: ???
- Web page:
  - http://graphics.ucsd.edu/courses/cse169_w09/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 Beginning of Week 3
  - Skeleton Hierarchy: Load a .skel file and display a 3D pose-able skeleton

- Project 2: Due Beginning of Week 5
  - Skin: Load .skin file and attach to the skeleton

- Project 3: Due Beginning of Week 7
  - Animation: Load .anim file and play back a key-framed animation on the skeleton

- Project 4: Due Beginning of Week 10 (Choose one of the following)
  - Cloth: Implement a simple cloth simulation
  - Locomotion & Inverse Kinematics: Implement an IK algorithm and use it to achieve a walking character
  - 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

<table>
<thead>
<tr>
<th></th>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/5</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>1/7</td>
<td>Skeletons</td>
</tr>
<tr>
<td>3</td>
<td>1/12</td>
<td>Quaternions</td>
</tr>
<tr>
<td>4</td>
<td>1/14</td>
<td>Skinning</td>
</tr>
<tr>
<td>5</td>
<td>1/19</td>
<td>(Holiday)</td>
</tr>
<tr>
<td>6</td>
<td>1/21</td>
<td>Facial Animation</td>
</tr>
<tr>
<td>7</td>
<td>1/26</td>
<td>Advanced Skinning</td>
</tr>
<tr>
<td>8</td>
<td>1/28</td>
<td>Channels &amp; Keyframes</td>
</tr>
<tr>
<td>9</td>
<td>2/2</td>
<td>Review</td>
</tr>
<tr>
<td>10</td>
<td>2/4</td>
<td>Midterm</td>
</tr>
<tr>
<td>11</td>
<td>2/9</td>
<td>Animation Blending</td>
</tr>
<tr>
<td>12</td>
<td>2/11</td>
<td>Inverse Kinematics 1</td>
</tr>
<tr>
<td>13</td>
<td>2/16</td>
<td>(Holiday)</td>
</tr>
<tr>
<td>14</td>
<td>2/18</td>
<td>Inverse Kinematics 2</td>
</tr>
<tr>
<td>15</td>
<td>2/23</td>
<td>Locomotion</td>
</tr>
<tr>
<td>16</td>
<td>2/25</td>
<td>Particle Systems</td>
</tr>
<tr>
<td>17</td>
<td>3/2</td>
<td>Cloth Simulation</td>
</tr>
<tr>
<td>18</td>
<td>3/4</td>
<td>Collision Detection</td>
</tr>
<tr>
<td>19</td>
<td>3/9</td>
<td>Rigid Body Physics</td>
</tr>
<tr>
<td>20</td>
<td>3/11</td>
<td>Final Review</td>
</tr>
</tbody>
</table>
Motion Capture

- The new CAL(IT)2 building is equipped with a ‘motion capture’ studio
- The system is capable of capturing 3D motion of a complex articulated figure (such as a human actor)
- It uses a Vicon motion capture system with 22 4-megapixel cameras capable of capturing at 160 frames per second
- We have an opportunity to work with some of this equipment this quarter
- Please come talk to me if you are interested in learning more about this
Angel Studios

- **Movies:**
  - The Lawnmower Man
  - Enertopia (stereoscopic IMAX)
- **Videos:** Peter Gabriel’s “Kiss That Frog”
- **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**
Currently, I run a company in Carlsbad called PixelActive

We make video game technology including a 3D world editor capable of making complex road networks and dense urban areas (cities)

We also develop a game engine for the PC, XBox360, Wii, and PS3
PixelActive
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();
}

- 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 Tools

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

- Many more…
Animation Production

- 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

- **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.
Rendering

- 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
Vector Review
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 *unit vector*
- We can also *normalize* a vector to make it a unit vector:
  \[ \frac{|\mathbf{v}|}{|\mathbf{v}|} = \frac{\mathbf{v}}{|\mathbf{v}|} \]
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

\[ a \cdot b = \sum a_i b_i \]

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

\[ a \cdot b = |a| |b| \cos \theta \]

\[ a \cdot b = a^T b \]

\[ a \cdot b = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \]
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 $a \cdot b > 0$ then $\angle < 90^\circ$
  - If $a \cdot b < 0$ then $\angle > 90^\circ$
  - If $a \cdot b = 0$ then $\angle = 90^\circ$ (or one or more of the vectors is degenerate $(0,0,0)$)
If $|u|=1.0$ then $a \cdot u$ is the length of the projection of $a$ onto $u$. 
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

\[ \mathbf{a} \cdot \mathbf{b} = 0 \]

\[ 0 < \mathbf{a} \cdot \mathbf{b} < 1 \]

\[ \mathbf{a} \cdot \mathbf{b} = -1 \]

\[ -1 < \mathbf{a} \cdot \mathbf{b} < 0 \]

\[ |\mathbf{a}| = |\mathbf{b}| = 1.0 \]

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

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

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

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

- Find the unit length normal of the triangle defined by 3D points \(a\), \(b\), and \(c\)
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

\[ \text{area} = \frac{1}{2} \left| (b - a) \times (c - a) \right| \]
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

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

\[ \theta = \cos^{-1}\left(\frac{h \cdot (t - p)}{|(t - p)|}\right) \]
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; y-=a.y; z-=a.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;
};