



Computer Animation

CSE169: Computer Animation

Instructor: Steve Rotenberg

UCSD, Winter 2019

CSE169

- Computer Animation Programming
 - Instructor: Steve Rotenberg (srotenberg@eng.ucsd.edu)
 - TA: Abhilash Srivastava (a8srivas@eng.ucsd.edu)
 - Lecture: Sequoyah Hall 147 (TTh 5:00-6:20pm)
 - Final: Thursday, 3/21, 7:00pm-10:00pm
 - Office: CSE 2210 (TTh 3:50-4:50pm)
 - Lab: EBU3 basement
 - Discussion: Sequoyah Hall 147 (W 8:00-8:50pm)
 - Web page:
 - <https://cseweb.ucsd.edu/classes/wi19/cse169-a/>
-

Prerequisites

- CSE167 or equivalent introduction to computer graphics
 - Familiarity with:
 - Vectors (dot products, cross products...)
 - Matrices (4x4 homogeneous transformations)
 - Polygon rendering
 - Basic lighting (normals, shaders, lighting models...)
 - OpenGL, Vulkan, Direct3D, Java3D, or equivalent
 - C++ or Java
 - GLM (OpenGL matrix/vector library)
 - Object oriented programming
 - Basic physics (force, momentum, Newton's laws...)
-

Programming Projects

- Project 1: Due 1/17 (Week 2)
 - Skeleton Hierarchy: Load a .skel file and display a 3D pose-able skeleton
 - Project 2: Due 1/31 (Week 4)
 - Skin: Load .skin file and attach to the skeleton
 - Project 3: Due 2/14 (Week 6)
 - Animation: Load .anim file and play back a key-framed animation on the skeleton
 - Project 4: Due 2/28 (Week 8)
 - Cloth: Implement a simple cloth simulation with elasticity, damping, gravity and aerodynamics
 - Project 5: Due 3/14 (Week 10)
 - Final Project: Implement one of several sample projects (particle system, SPH, rigid body, inverse kinematics) or come up with your own idea
-

Programming Projects

- You can use any programming language & operating system that you choose
 - You can use any graphics API that you choose (OpenGL, Vulkan, Direct3D, etc.) but I may put some restrictions on the features you use
 - Most students use C++ with OpenGL
-

Grading

- 15% Project 1
 - 15% Project 2
 - 15% Project 3
 - 15% Project 4
 - 15% Project 5
 - 10% Midterm
 - 15% Final
-

Programming Assignment Turn-In

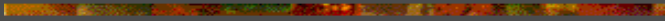
- The project must be shown to the instructor or TA before 4:50 on the due date (when class starts)
- They will both be in the lab from 3:00-4:50 on due days, but you can turn them in early as well
- If necessary, projects can be turned in immediately after class on due days if you speak to the instructor before hand (for example if there are too many projects to grade in time)
- If you finish on time but for some strange reason can't turn it in personally, you can email the code and images to the instructor *and* TA and demo it personally some time in the following week for full credit
- If you don't finish on time, you can turn what you have in for partial credit. Either way, you can turn it in late during the following week for -4 points. So for example on a 15 point assignment, you can turn it in for partial credit and get 6, but then finish it and turn it in late for up to 11 points
- Anything after 1 week can still be turned in but for -8 points
- Note that most projects build upon each other so you have to do them eventually...

Course Outline

1. 1/8: Introduction
 2. 1/10: Skeletons
 3. 1/15: Quaternions
 4. 1/17: Skinning
 5. 1/22: Facial Animation
 6. 1/24: Channels & Keyframes
 7. 1/29: Animation Blending
 8. 1/31: Inverse Kinematics 1
 9. 2/5: Inverse Kinematics 2
 10. 2/7: Midterm
 11. 2/12: Particle Systems
 12. 2/14: Cloth Simulation
 13. 2/19: Collision Detection
 14. 2/21: Locomotion
 15. 2/26: Fluid Dynamics
 16. 2/28: Particle Based Fluids
 17. 3/5: Rigid Body Physics
 18. 3/7: [TBD]
 19. 3/12: Advanced Animation Topics
 20. 3/14: Final Review
-

Who am I?

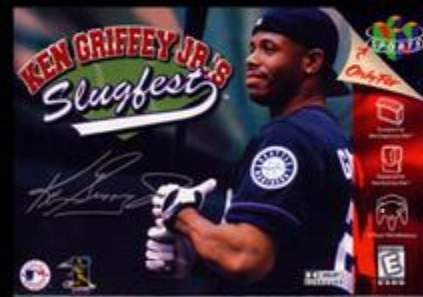
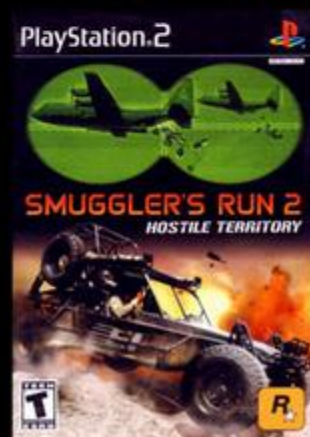
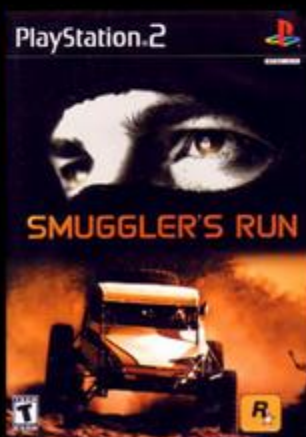
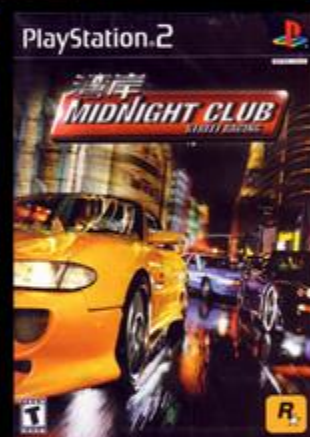
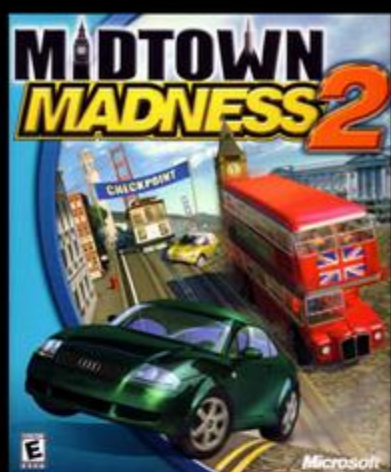
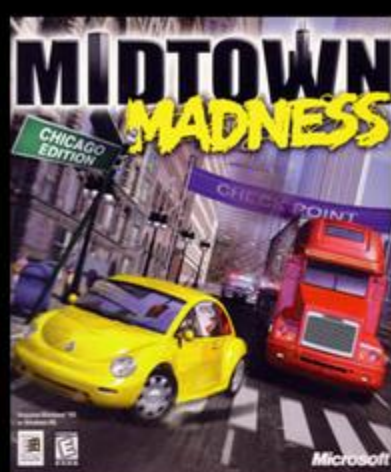


- Steve Rotenberg, Guest Lecturer at UCSD since 2003
 - Teaching
 - Taught CSE169 from 2004-2009, 2014-2018
 - Taught CSE168 in 2014 & 2017
 - Taught CSE167 a couple times
 - Will be teaching CSE291: Physics Simulation next quarter
 - Work History:
 - Angel Studios 1992-2002
 - PixelActive 2003-2010
 - NATVEQ 2010-2011
 - Nokia 2011-2013
 - Consultant/Contractor 2014-2016
 - VectorZero 2017-present
- 

Angel Studios



- I was Director of Software for 10 years
- 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

- I was founder and CEO of PixelActive Inc.
 - 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
 - Rebranded as HERE Maps in 2012
 - Sold to Daimler-Audi-BMW in 2015
-

PixelActive



VectorZero

- Started a new company in 2017 called VectorZero
 - We're creating 3D road modeling and traffic simulation software
 - Current focus is on tools to aid in testing & developing autonomous vehicles in a simulated environment
 - www.vectorzero.io
-

VectorZero





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
-

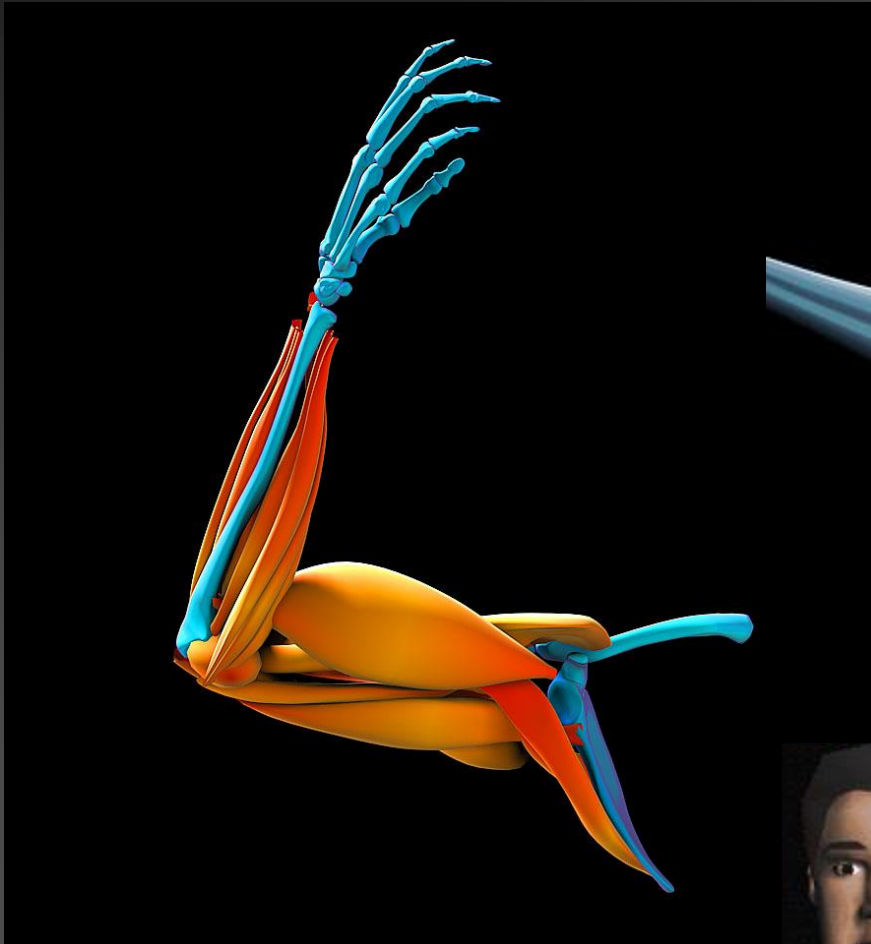
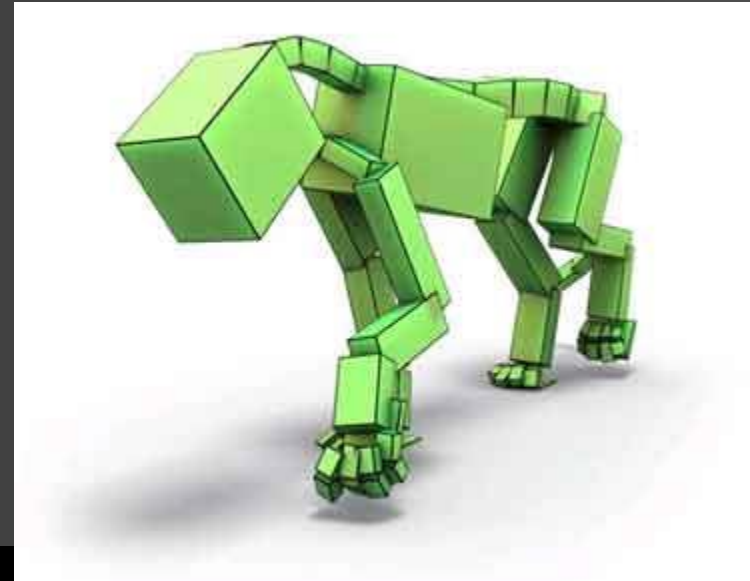
Character Rigging

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

Character Animation

- Keyframe Animation
 - Motion Capture
 - Inverse Kinematics
 - Locomotion
 - Procedural Animation
 - Artificial Intelligence
-

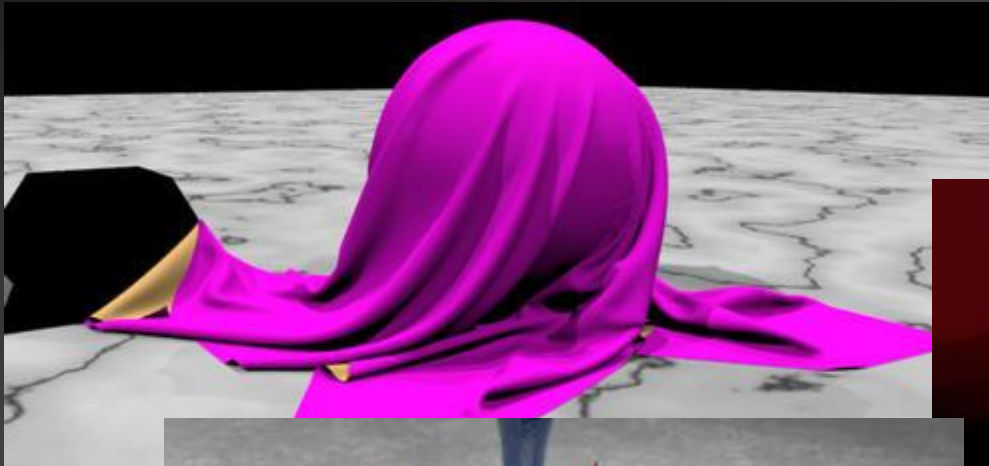
Character Animation



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
- UHD TV, 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*

Rendering

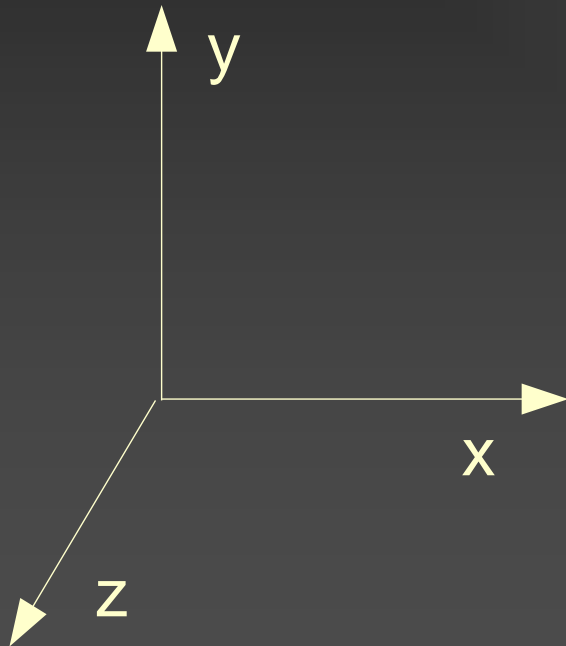
- There are many ways to design a 3D renderer
 - The two most common approaches are:
 - Traditional graphics pipeline
 - Ray-based rendering
 - With the traditional approach, primitives (usually triangles) are rendered into the image one at a time, and complex visual effects often involve a variety of 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:

$$\frac{\mathbf{v}}{|\mathbf{v}|}$$

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

$$\mathbf{a} \cdot \mathbf{b} = \sum a_i b_i$$

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

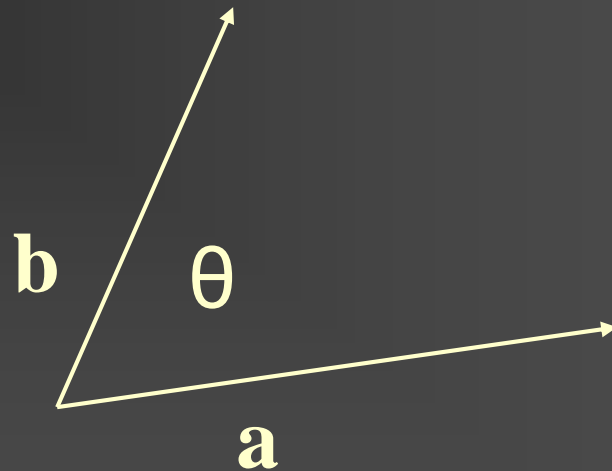
$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta$$

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b}$$

$$\mathbf{a} \cdot \mathbf{b} = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$$

Example: Angle Between Vectors

- How do you find the angle θ between vectors **a** and **b**?

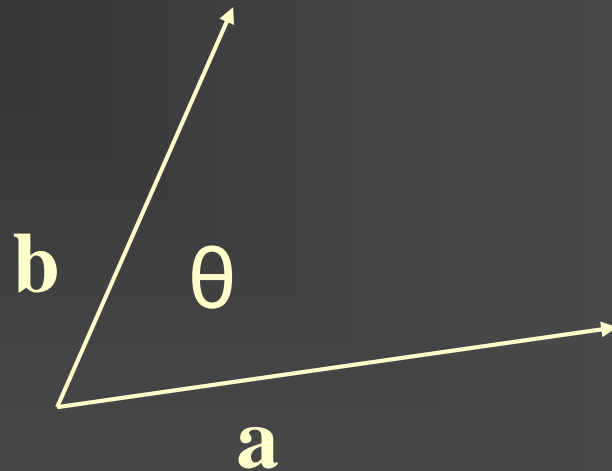


Example: Angle Between Vectors

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta$$

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

$$\theta = \cos^{-1} \left(\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|} \right)$$

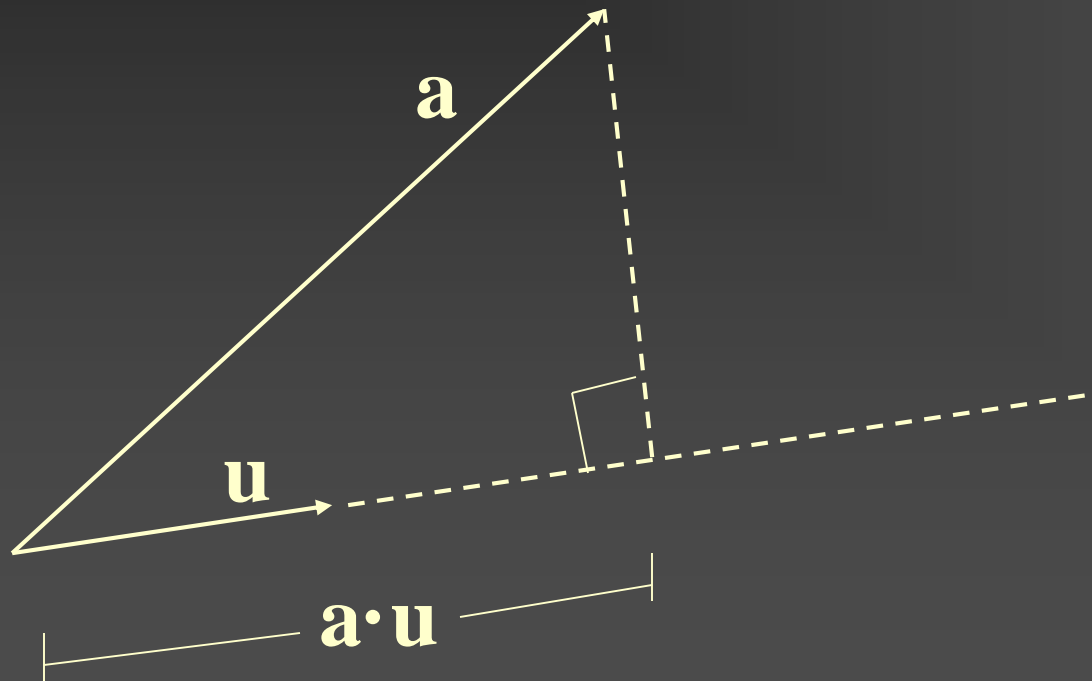


Dot Products with General Vectors

- 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))

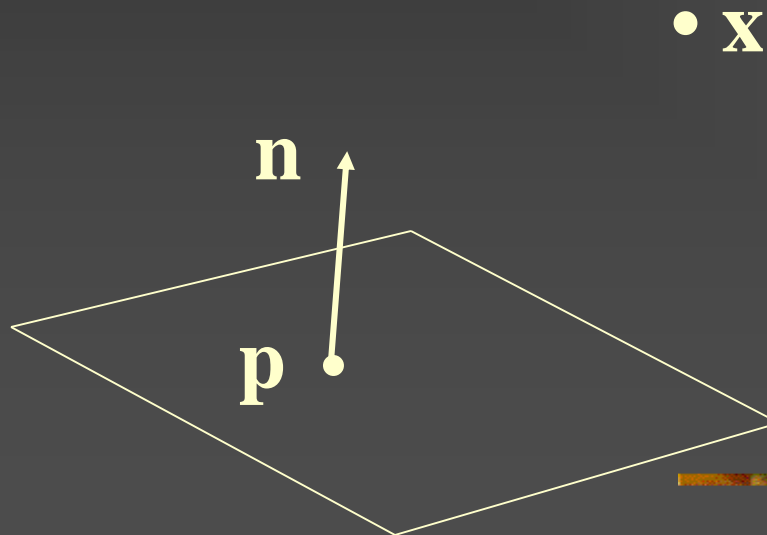
Dot Products with One Unit Vector

- 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

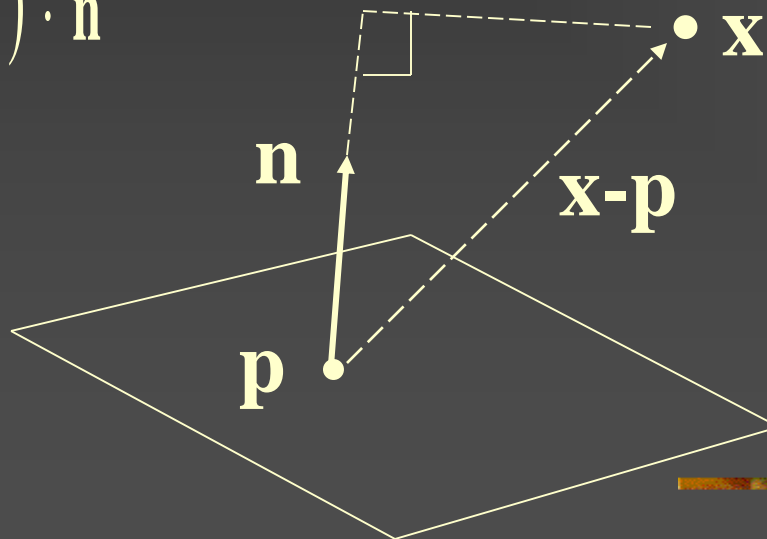
- A plane is described by a point \mathbf{p} on the plane and a unit normal \mathbf{n} . Find the distance from point \mathbf{x} to the plane



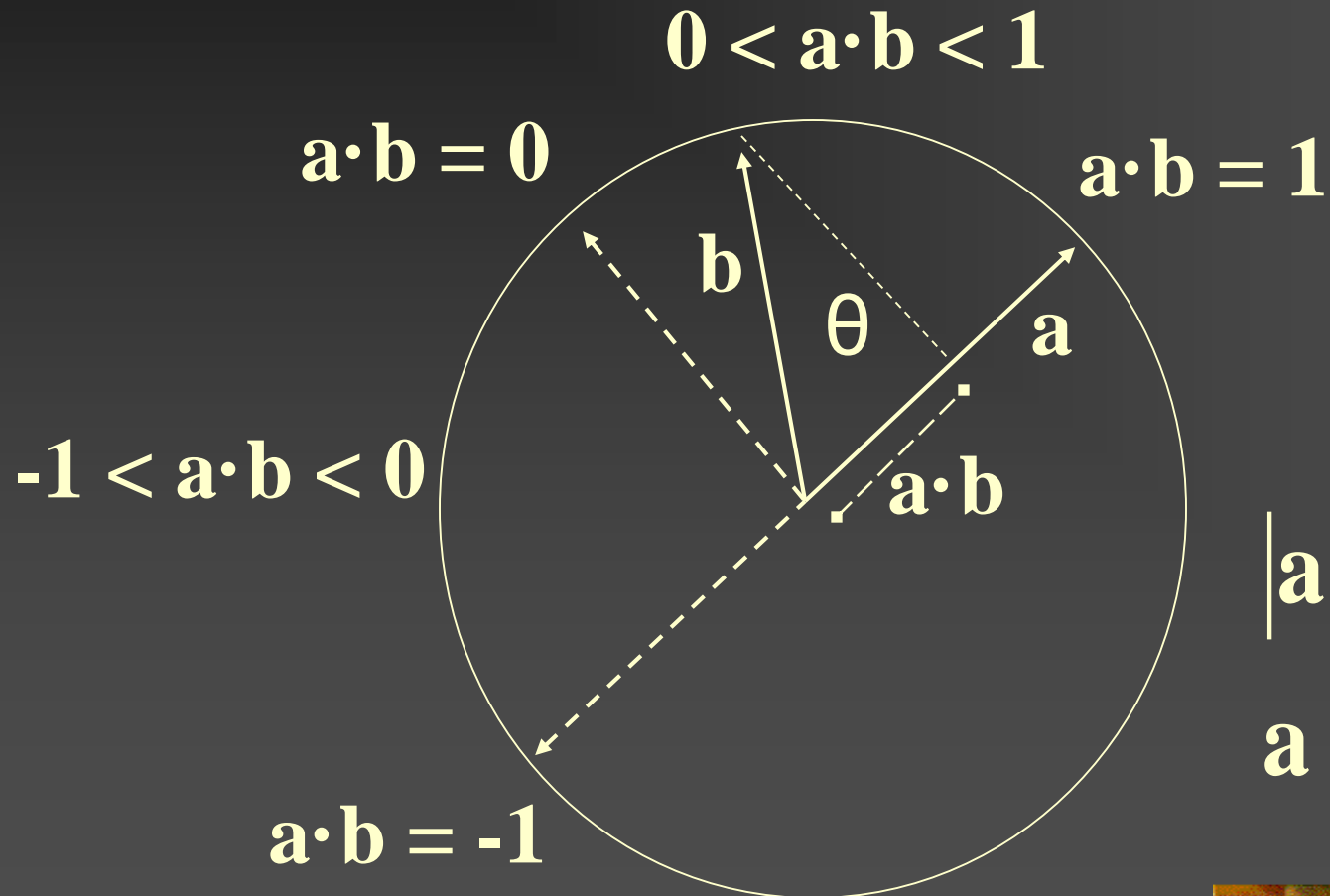
Example: Distance to Plane

- The distance is the length of the projection of $\mathbf{x} - \mathbf{p}$ onto \mathbf{n} :

$$dist = (\mathbf{x} - \mathbf{p}) \cdot \mathbf{n}$$



Dot Products with Unit Vectors



$$|\mathbf{a}| = |\mathbf{b}| = 1.0$$
$$\mathbf{a} \cdot \mathbf{b} = \cos(\theta)$$

Cross Product

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

$$\mathbf{a} \times \mathbf{b} = \left[a_y b_z - a_z b_y \quad a_z b_x - a_x b_z \quad a_x b_y - a_y b_x \right]$$

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

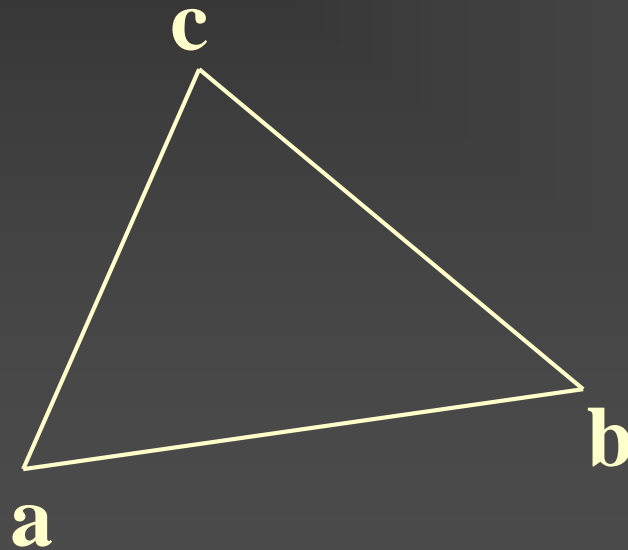
$$|\mathbf{a} \times \mathbf{b}| = |\mathbf{a}| |\mathbf{b}| \sin \theta$$

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

$$|\mathbf{a} \times \mathbf{b}| = 0 \text{ if } \mathbf{a} \text{ and } \mathbf{b} \text{ are parallel}$$

Example: Normal of a Triangle

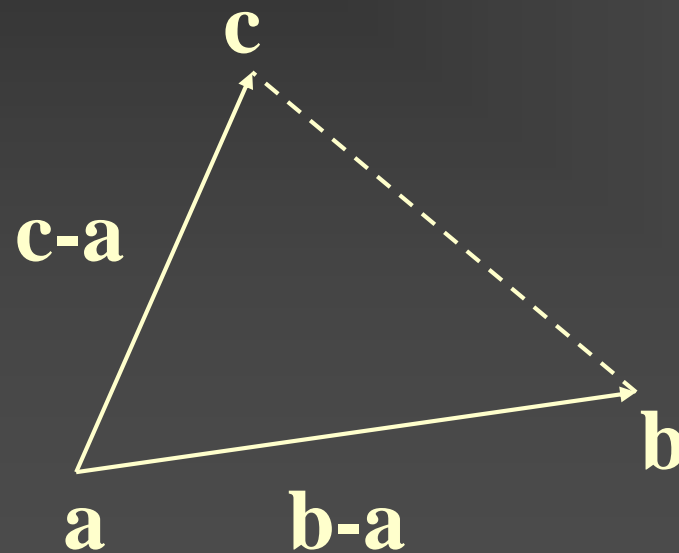
- Find the unit length normal of the triangle defined by 3D points **a**, **b**, and **c**



Example: Normal of a Triangle

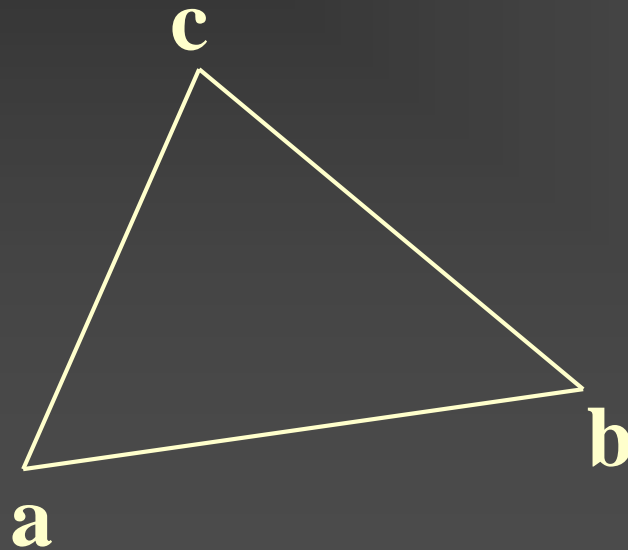
$$\mathbf{n}^* = (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})$$

$$\mathbf{n} = \frac{\mathbf{n}^*}{|\mathbf{n}^*|}$$



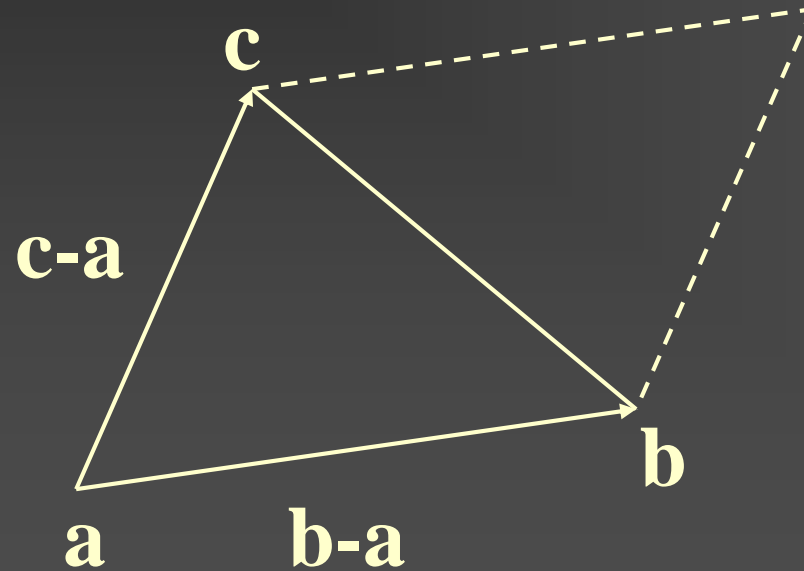
Example: Area of a Triangle

- Find the area of the triangle defined by 3D points **a**, **b**, and **c**



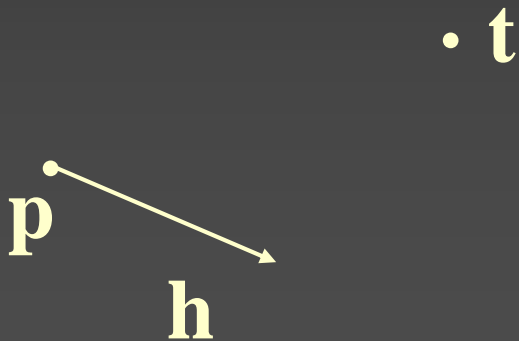
Example: Area of a Triangle

$$area = \frac{1}{2} |(\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})|$$



Example: Alignment to Target

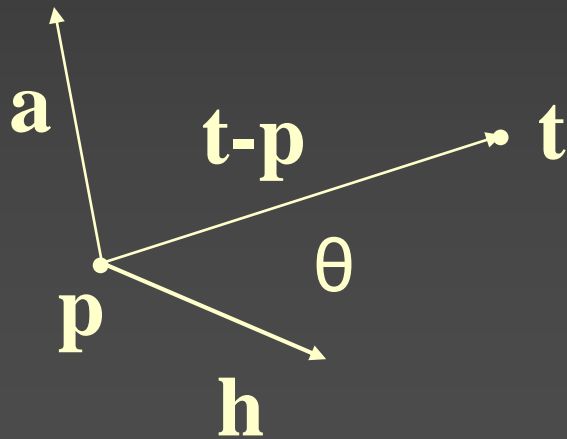
- An object is at position \mathbf{p} with a unit length heading of \mathbf{h} . We want to rotate it so that the heading is facing some target \mathbf{t} . Find a unit axis \mathbf{a} and an angle θ to rotate around.



Example: Alignment to Target

$$\mathbf{a} = \frac{\mathbf{h} \times (\mathbf{t} - \mathbf{p})}{|\mathbf{h} \times (\mathbf{t} - \mathbf{p})|}$$

$$\theta = \cos^{-1} \left(\frac{\mathbf{h} \cdot (\mathbf{t} - \mathbf{p})}{|\mathbf{t} - \mathbf{p}|} \right)$$



GLM

- For this class, I recommend using GLM for vector & matrix operations
 - GLM is a header-only library that is compatible with most C++ compilers and works well with GL
 - It provides a set of generic vector & matrix classes useful for 3D graphics
 - See <https://glm.g-truc.net>
-

Sample Code

- I will post some sample/starter code on the web page in the next day or so
 - It will handle some basic OpenGL stuff to get things started (basic shader set-up, camera control, vertex buffer rendering)
 - Feel free to use it or use something else
-