Lecture 24 of 41

Collision Handling Part 1 of 2: Separating Axes, Oriented Bounding Boxes

William H. Hsu
Department of Computing and Information Sciences, KSU

KSOL course pages: http://bit.ly/hGvXlH / http://bit.ly/eVizrE
Public mirror web site: http://www.kddresearch.org/Courses/CIS636
Instructor home page: http://www.cis.ksu.edu/~bhsu

Readings:

Today: §2.4.3, 8.1, Eberly *2e* – see **http://bit.ly/ieUq45**; **GL handout**
Next class: Chapter 6, esp. §6.1, Eberly *2e*
Wikipedia, *Collision Detection*: **http://bit.ly/14rFzG**

Lecture Outline

- Reading for Last Class: Chapter 10, 13, §17.3 – 17.5, Eberly *2e*
- Reading for Today: §2.4.3, 8.1, Eberly *2e*, GL handout
- Reading for Next Class: Chapter 6, Esp. §6.1, Eberly *2e*
- Last Time: Quaternions Concluded
  - How quaternions work – properties, matrix equivalence, arithmetic
  - Composing rotations by quaternion multiplication
  - Incremental rotation and error issues
- Videos 4: Modeling & Simulation, Visualization; VR/VE/VA/AR
  - Virtual reality, environments, artifacts (VR/VE/VA); augmented reality
  - Relationship among visualization, simulation, & animation
- Today: Collision Detection Part 1 of 2
  - Test-intersection queries *vs.* find-intersection queries
  - Static: stationary objects (both not moving)
  - Dynamic: moving objects (one or both)
  - Distance *vs.* intersection methods

Where We Are

21	**Lab 4a: Animation Basics**	**Flash animation handout**
22	Animation 2: Rotations; Dynamics, Kinematics	Chapter 17, esp. §17.1 – 17.2
23	Demos 4: Modeling & Simulation; Rotations	Chapter 10[1], 13[2], §17.3 – 17.5
24	**Collisions 1: axes, OBBs, Lab 4b**	**§2.4.3, 8.1, GL handout**
25	Spatial Sorting: Binary Space Partitioning	Chapter 6, esp. §6.1
26	**Demos 5: More CGA; Picking; HW/Exam**	**Chapter 7[2]; § 8.4**
27	**Lab 5a: Interaction Handling**	**§ 8.3 – 8.4; 4.2, 5.0, 5.6, 9.1**
28	Collisions 2: Dynamic, Particle Systems	§ 9.1, particle system handout
	Exam 2 review; Hour Exam 2 (evening)	**Chapters 5 – 6, 7[2] – 8, 12, 17**
29	**Lab 5b: Particle Systems**	**Particle system handout**
30	**Animation 3: Control & IK**	**§ 5.3, CGA handout**
31	Ray Tracing 1: intersections, ray trees	Chapter 14
32	Lab 6a: Ray Tracing Basics with POV-Ray	RT handout
33	Ray Tracing 2: advanced topic survey	Chapter 15, **RT handout**
34	Visualization 1: Data (Quantities & Evidence)	Tufte handout (1)
35	**Lab 6b: More Ray Tracing**	**RT handout**
36	Visualization 2: Objects	Tufte handout (2 & 4)
37	Color Basics; Term Project Prep	**Color handout**
38	Lab 7: Fractals & Terrain Generation	**Fractals/Terrain handout**
39	**Visualization 3: Processes; Final Review 1**	Tufte handout (3)
40	**Project presentations 1; Final Review 2**	**–**
41	Project presentations 2	–
	Final Exam	**Ch. 1 – 8, 10 – 15, 17, 20**

Lightly-shaded entries denote the due date of a written problem set; heavily-shaded entries, that of a machine problem (programming assignment); blue-shaded entries, that of a paper review; and the green-shaded entry, that of the term project.

Green, **blue** and **red** letters denote **exam review**, **exam**, and **exam solution review** dates.

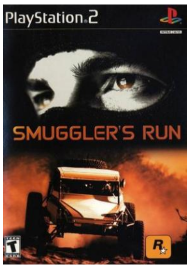
Acknowledgements: Quaternions, Collision Handling

Rick Parent

Professor
Department of Computer Science and Engineering
Ohio State University
http://www.cse.ohio-state.edu/~parent/

David C. Brogan

Visiting Assistant Professor, Computer Science Department, University of Virginia
http://www.cs.virginia.edu/~dbrogan/
Susquehanna International Group (SIG)
http://www.sig.com



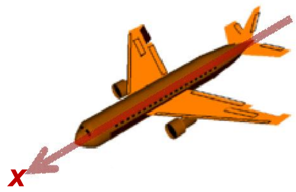
Steve Rotenberg

Visiting Lecturer
Graphics Lab
University of California – San Diego
CEO/Chief Scientist, PixelActive
http://graphics.ucsd.edu

Review [1]: Fixed Angles & Euler Angles

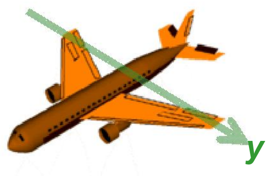
Rotation about *x* axis (Roll)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Rotation about *y* axis (Pitch)

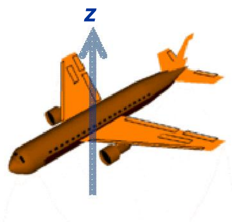
$$\begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



)06 Wikipedia,
ıht Dynamics
'/bit.ly/gVaQCX

Rotation about *z* axis (Yaw)

$$\begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Adapted from slides ♥ 2007 – 2011 R. Parent, Ohio State University
CSE 682 (Computer Animation), http://bit.ly/feUESy
CSE 683/684A (Computer Animation Algorithms & Techniques), http://bit.ly/f8Myky

Review [2]: Axis-Angle to Quaternion Conversion

A quaternion is a 4-D unit vector q = [x y z w]

- It lies on the unit hypersphere $x^2 + y^2 + z^2 + w^2 = 1$

For rotation about (unit) axis v by angle θ

- vector part = (sin θ/2) *v* = *[x y z]*
- scalar part = (cos θ/2) = *w*
- $(\sin(\theta/2)\, n_x, \sin(\theta/2)\, n_y, \sin(\theta/2)\, n_z, \cos(\theta/2))$

Only a unit quaternion encodes a rotation - normalize

Adapted from slides ♥ 2000 – 2004 D. Brogan, University of Virginia
CS 445/645, Introduction to Computer Graphics, http://bit.ly/h9AHRg

Computer Science at the University of Virginia

Review [3]: Quaternion to RM Conversion

Rotation matrix corresponding to a quaternion:

- [x y z w] = $\begin{bmatrix} 1-2y^2-2z^2 & 2xy+2wz & 2xz-2wy \\ 2xy-2wz & 1-2x^2-2z^2 & 2yz+2wx \\ 2xz+2wy & 2yz-2wx & 1-2x^2-2y^2 \end{bmatrix}$

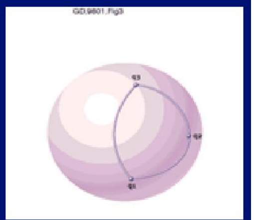
Quaternion Multiplication

- $q_1 * q_2 = [v_1, w_1] * [v_2, w_2] = [(w_1 v_2 + w_2 v_1 + (v_1 \times v_2)), w_1 w_2 - v_1 . v_2]$
- quaternion * quaternion = quaternion
- this satisfies requirements for mathematical *group*
- Rotating object twice according to two different quaternions is equivalent to one rotation according to product of two quaternions

Review [4]: Advantage – Interpolation

Biggest advantage of quaternions

- Interpolation
- Cannot linearly interpolate between two quaternions because it would speed up in middle
- Instead, Spherical Linear Interpolation, slerp()
- Used by modern video games for third-person perspective
- Why?



Hint: see http://youtu.be/-jBKKV2V8eU

Adapted from slides ♥ 2000 – 2004 D. Brogan, University of Virginia
CS 445/645, Introduction to Computer Graphics, http://bit.ly/h9AHRg

Computer Science at the University of Virginia

Interpolating Quaternions [1]: Lerp

- If we want to do a linear interpolation between two points **a** and **b** in normal space

  Lerp(t,**a**,**b**) = (1-t)**a** + (t)**b**

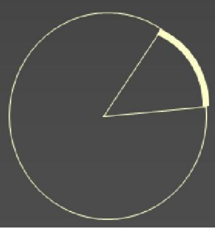
  where t ranges from 0 to 1
- Note that the Lerp operation can be thought of as a weighted average (convex)
- We could also write it in its additive blend form:

  Lerp(t,**a**,**b**) = **a** + t(**b**-**a**)

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Interpolating Quaternions [2]: Slerp

- If we want to interpolate between two points on a sphere (or hypersphere), we don't just want to Lerp between them
- Instead, we will travel across the surface of the sphere by following a 'great arc'



Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Interpolating Quaternions [3]: Slerp Optimization

- Remember that there are two redundant vectors in quaternion space for every unique orientation in 3D space
- What is the difference between:

  Slerp(t,**a**,**b**) and Slerp(t,-**a**,**b**) ?

- One of these will travel less than 90 degrees while the other will travel more than 90 degrees across the sphere
- This corresponds to rotating the 'short way' or the 'long way'
- Usually, we want to take the short way, so we negate one of them if their dot product is < 0

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

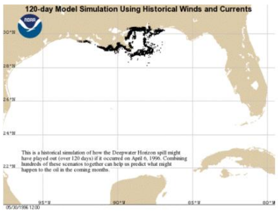
Review [5]: Dynamics & Kinematics

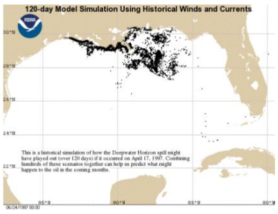
- Dynamics: Study of Motion & Changes in Motion
  - Forward: model forces over time to find state, *e.g.*,
    - Given: initial position p_0, velocity v_0, gravitational constants
    - Calculate: position p_t at time t
  - Inverse: given state and constraints, calculate forces, *e.g.*,
    - Given: *desired* position p_t at time t, gravitational constants
    - Calculate: position p_0, velocity v_0 needed
  - Wikipedia: http://bit.ly/hH43dX (see also: "Analytical dynamics")
  - For non-particle objects: rigid-body dynamics (http://bit.ly/dLvejg)
- Kinematics: Study of Motion without Regard to Causative Forces
  - Modeling systems – *e.g.*, articulated figure
  - Forward: from angles to position (http://bit.ly/eh2d1c)
  - Inverse: finding angles given desired position (http://bit.ly/hsyTb0)
  - Wikipedia: http://bit.ly/hr8r2u

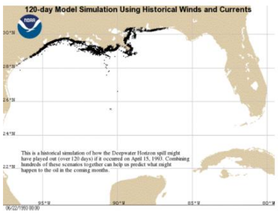
Forward Kinematics
© 2009 Wikipedia

Review [6]: Visualization & Simulation

Deepwater Horizon Oil Spill (20 Apr 2010)
http://bit.ly/9QHax4
120-day images © 2010 NOAA, http://1.usa.gov/c02xuQ







120-day simulation using
15 Apr 1993 weather conditions

132-day simulation using
2010 conditions
© 2010 National Center for Supercomputing Applications (NCSA)
http://youtu.be/pE-1G_476nA

Wilhelmson *et al.* (2004)
http://youtu.be/EgumU0Ns1YI
http://avl.ncsa.illinois.edu
http://bit.ly/eA8PXN

Review [7]: Virtual Reality (VR)

- Virtual Reality: Computer-Simulated Environments
- Physical Presence: Real & Imaginary
- Hardware: User Interface
  - Head-mounted display (HMD), gloves – see PopOptics goggles (left)
  - VR glasses, wand, *etc.* – see NCSA CAVE (right)

Virtual Reality, Wikipedia:
http://bit.ly/fAvNeP
Image © 2007 National Air & Space Museum

CAVE (Cave Automatic Virtual Environment)
Image © 2009 D. Pape
HowStuffWorks article: http://bit.ly/feQxNK
© 2009 J. Strickland
Wikipedia: http://bit.ly/dKNEnU

Review [8]: Virtual Environments (VE)

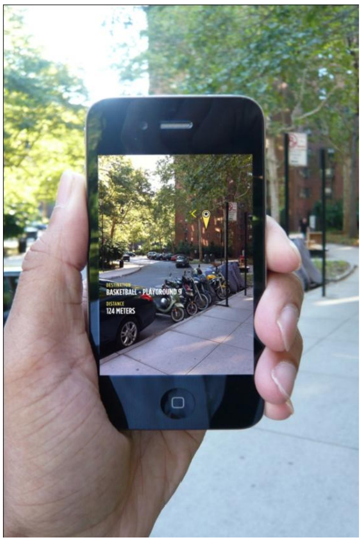
- Virtual Environment: Part of Virtual Reality Experience
- Other Parts
  - Virtual artifacts (VA): simulated objects – http://bit.ly/hskSyX
  - Intelligent agents, artificial & real – http://bit.ly/y2gQk

Experientia © 2006 M. Vanderbeeken *et al.*, http://bit.ly/hzfAQx
Second Life © 2003 – 2011 Linden Labs, Inc., http://bit.ly/wbvoL
Image © 2006 Philips Design

We Are Arcade © 2011 D. Grossett *et al.*, http://bit.ly/ftALjU
World of Warcraft: Cataclysm review © 2011
J. Greer, http://bit.ly/eENHXt
World of Warcraft © 2001 – 2011
Blizzard Entertainment, Inc.,
http://bit.ly/2qvPYF

Review [9]: Augmented Reality (AR)

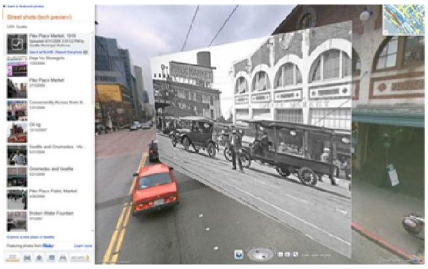
- Augmented Reality: Computer-Generated (CG) Sensory Overlay
- Added to Physical, Real-World Environment



"40 Best Augmented Reality iPhone Applications",
© 2010 iPhoneNess.com, http://bit.ly/2qT35y
MyNav © 2010 Winfield & Co. http://bit.ly/dLTir7

Google goggles labs

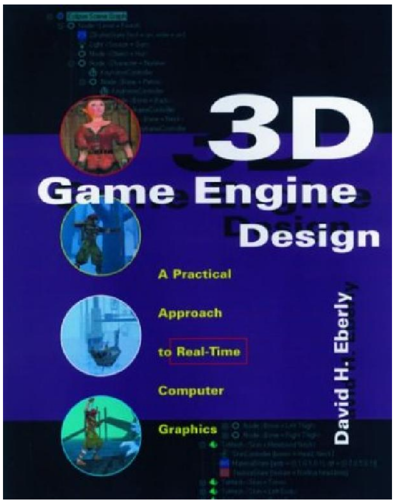
Wikipedia, *Google Goggles*:
http://bit.ly/gRRMLS



Bing Maps © 2010 – 2011
Microsoft Corporation
http://bit.ly/a9UviT
© 2010 TED Talks

Acknowledgements: Intersections, Containment – Eberly 1e

David H. Eberly
Chief Technology Officer
Geometric Tools, LLC
http://www.geometrictools.com
http://bit.ly/enKbfs



3D Game Engine Design © 2000 D. H. Eberly
See http://bit.ly/ieUq45 for second edition table of contents (TOC)

Today's material:

- **View Frustum clipping**
  - §2.4.3, p. 70 – 77, 2e
  - §3.4.3, p. 93 – 99, & §3.7.2, p. 133 – 136, 1e
- **Collision detection: separating axes**
  - §8.1, p. 393 – 443, 2e
  - §6.4. p. 203 – 214, 1e

Later:

- **Distance methods**
  - Chapter 14, p. 639 – 679, 2e
  - §2.6, p. 38 – 77, 1e
- **Intersection methods**
  - Chapter 15, p. 681 – 717, 2e
  - §6.2 – 6.5, p. 188 – 243, 1e

View Frustum Clipping: Triangle Splitting

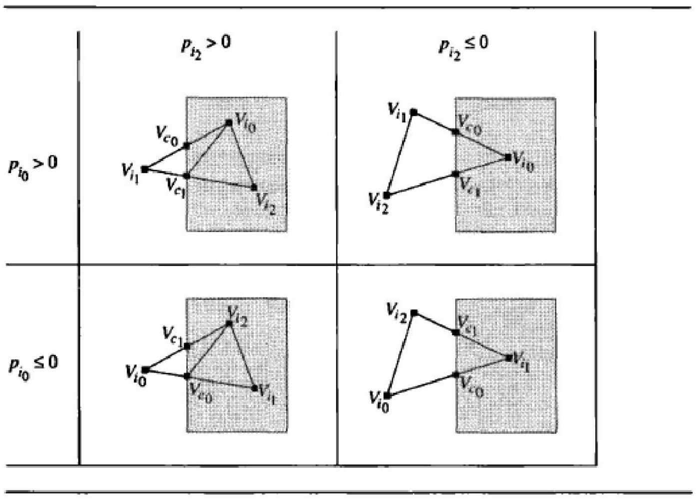


Figure 3.4 Four configurations for triangle splitting. Only the triangles in the shaded region are important, so the quadrilaterals outside are not split.

3D Game Engine Design © 2000 D. H. Eberly
See http://bit.ly/ieUq45 for second edition table of contents (TOC)

Collision Handling: Detection vs. Response

- Collision Detection
  - Collision detection is a geometric problem
  - Given two moving objects defined in an initial and final configuration, determine if they intersected at some point between the two states
- Collision Response
  - The response to collisions is the actual physics problem of determining the unknown forces (or impulses) of the collision

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [1]: Technical Problem Defined

- 'Collision detection' is really a geometric intersection detection problem
- Main subjects
  - Intersection testing (triangles, spheres, lines…)
  - Optimization structures (octree, BSP…)
  - Pair reduction (reducing N^2 object pair testing)

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [2]: Intersections – Testing vs. Finding

- General goals: given two objects with current and previous orientations specified, determine if, where, and when the two objects intersect
- Alternative: given two objects with only current orientations, determine if they intersect
- Sometimes, we need to find all intersections. Other times, we just want the first one. Sometimes, we just need to know if the two objects intersect and don't need the actual intersection data.

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [3]: Queries – Test- vs. Find-Intersection

- **Test-Intersection: Determine If Objects Intersect**
  - Static: test whether they do at given instant
  - Dynamic: test whether they intersect at any point along trajectories
- **Find-Intersection: Determine Intersection (or Contact) Set of Objects**
  - Static: intersection set (compare: A ∩ B)
  - Dynamic: contact time (interval of overlap), sets (depends on time)

Adapted from *3D Game Engine Design* © 2000 D. H. Eberly
See http://bit.ly/ieUq45 for second edition table of contents (TOC)

Collision Detection [3]: Queries – Distance vs. Intersection

- **Distance-Based**
  - Parametric representation of object boundaries/interiors
  - Want: closest points on two objects (to see whether they intersect)
  - Use: constrained minimization to solve for closest points
- **Intersection-Based**
  - Also uses parametric representation
  - Want: overlapping subset of interior of two objects
  - General approach: equate objects, solve for parameters
  - Use one of two kinds of solution methods
    - Analytical (when feasible to solve exactly – *e.g.*, OBBs)
    - Numerical (approximate region of overlap)
  - Solving for parameters in equation
  - Harder to compute than distance-based; use only when needed

Adapted from ***3D Game Engine Design*** © 2000 D. H. Eberly
See http://bit.ly/ieUq45 for second edition table of contents (TOC)

Collision Detection [4]: Primitives

- We often deal with various different 'primitives' that we describe our geometry with. Objects are constructed from these primitives
- Examples
  - Triangles
  - Spheres
  - Cylinders
  - AABB = axis aligned bounding box
  - OBB = oriented bounding box
- At the heart of the intersection testing are various primitive-primitive tests

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [5]: Particle Collisions

- For today, we will mainly be concerned with the problem of testing if particles collide with solid objects
- A particle can be treated as a line segment from its previous position to its current position
- If we are colliding against static objects, then we just need to test if the line segment intersects the object
- Colliding against moving objects requires some additional modifications that we will also look at

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [6]: Code – Basic Components

```
class Segment {
   Vector3 A,B;
};

class Intersection {
   Vector3 Position;
   Vector3 Normal;
   Material *Mtl;   (Mtl can contain info about
        elasticity, friction, etc)
};
```

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [7]: Code – Primitives

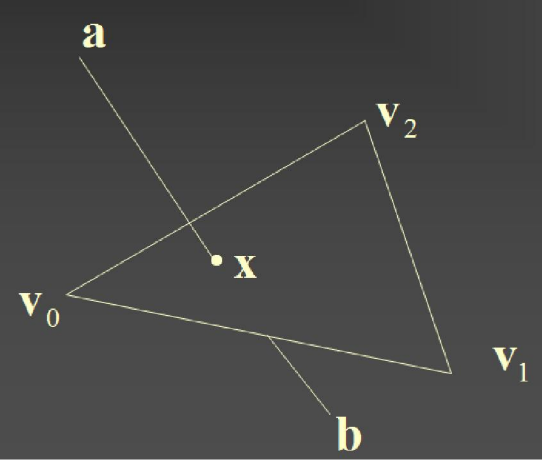
```
class Primitive {
   virtual bool TestSegment(const Segment &s,
         Intersection &i);
};

class Sphere:public Primitive…
class Triangle:public Primitive…
class Cylinder:public Primitive…
```

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [8]: Segment vs. Triangle – Query

- Does segment **ab** intersect triangle $\mathbf{v}_0\mathbf{v}_1\mathbf{v}_2$?

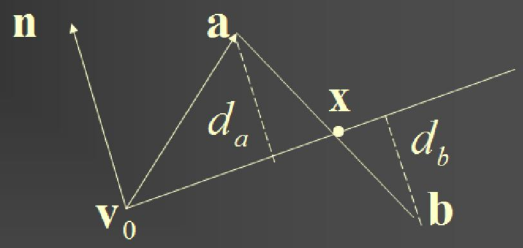


Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [9]: Segment vs. Triangle – Solution

- First, compute signed distances of a and b to plane

$$d_a = (\mathbf{a} - \mathbf{v}_0) \cdot \mathbf{n}$$
$$d_b = (\mathbf{b} - \mathbf{v}_0) \cdot \mathbf{n}$$



- Reject if both are above or both are below triangle
- Otherwise, find intersection point **x**

$$\mathbf{x} = \frac{d_a \mathbf{b} - d_b \mathbf{a}}{d_a - d_b}$$

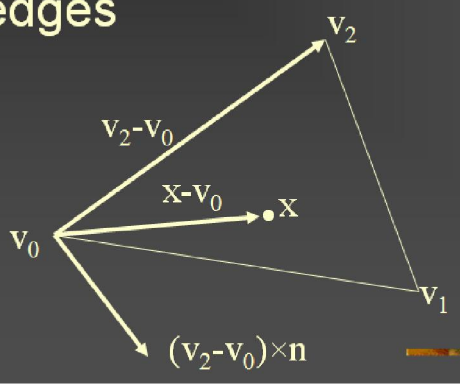
Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [10]: Segment vs. Triangle – Point Test

- Is point x inside the triangle?

$$(x-v_0)\cdot((v_2-v_0)\times n) > 0$$

- Test all 3 edges



Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [11]: Faster Triangle – Point Containment

- Reduce to 2D: remove smallest dimension
- Compute barycentric coordinates

  $x' = x - v_0$

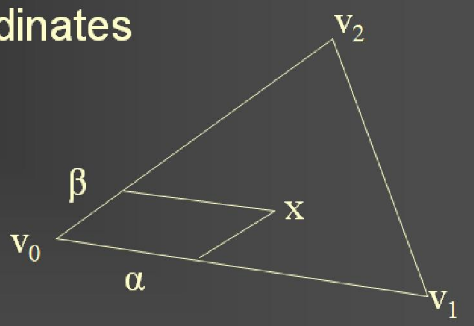
  $e_1 = v_1 - v_0$

  $e_2 = v_2 - v_0$

  $\alpha = (x' \times e_2)/(e_1 \times e_2)$

  $\beta = (x' \times e_1)/(e_1 \times e_2)$

- Reject if $\alpha < 0$, $\beta < 0$ or $\alpha + \beta > 1$



Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [12]: Segment vs. Mesh

- To test a line segment against a mesh of triangles, simply test the segment against each triangle
- Sometimes, we are interested in only the 'first' hit along the segment from **a** to **b**. Other times, we want all intersections. Still other times, we just need any intersection.
- Testing against lots of triangles in a large mesh can be time consuming. We will look at ways to optimize this later

Collision Detection [13]: Segment vs. Moving Mesh

- M_0 is the object's matrix at time t_0
- M_1 is the matrix at time t_1
- Compute delta matrix:

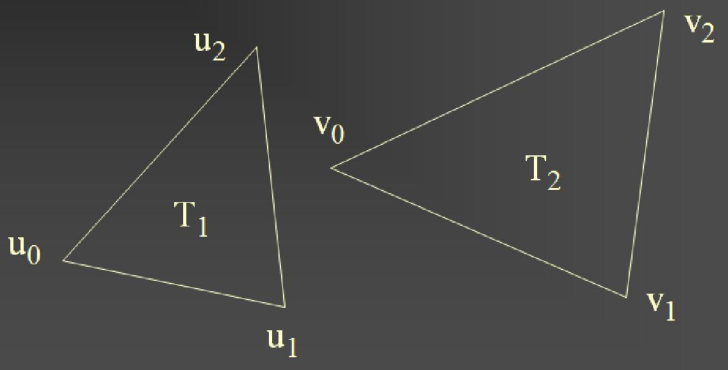
  $M_1 = M_0 \cdot M_\Delta$

  $M_\Delta = M_0^{-1} \cdot M_1$
- Transform a by M_Δ

  $a' = a \cdot M_\Delta$
- Test segment a'b against object with matrix M_1

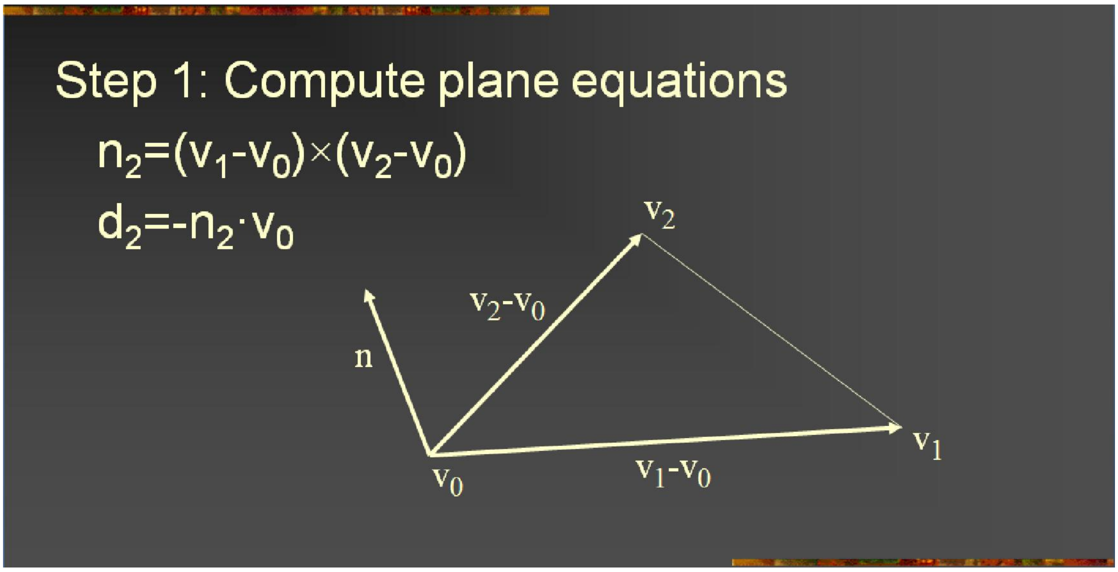
Collision Detection [14]: Trianglevs. Triangle – Query

- Given two triangles: T_1 ($u_0u_1u_2$) and T_2 ($v_0v_1v_2$)



Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [15]: Trianglevs. Triangle – Plane Equations

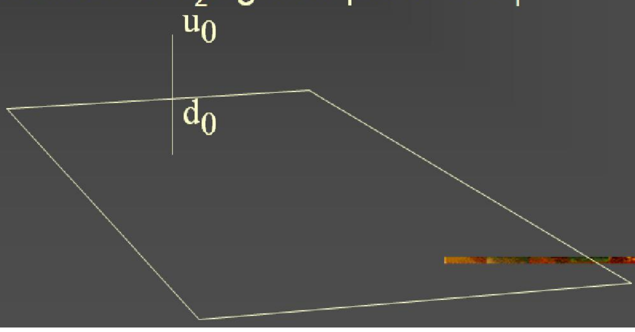


Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [16]: Triangle vs. Triangle – Distances

- Step 2: Compute signed distances of T_1 vertices to plane of T_2:

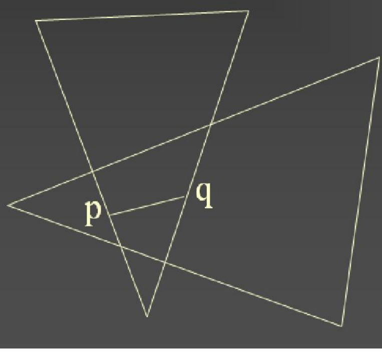
  $d_i = n_2 \cdot u_i + d_2$ (i=0,1,2)
- Reject if all $d_i<0$ or all $d_i>0$
- Repeat for vertices of T_2 against plane of T_1



Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [17]: Trianglevs. Triangle – Intersection

- Step 3: Find intersection points
- Step 4: Determine if segment pq is inside triangle or intersects triangle edge



Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [18]: Meshvs. Mesh – Kinds of Collisions

- Geometry: points, edges, faces
- Collisions: p2p, p2e, p2f, e2e, e2f, f2f
- Relevant ones: p2f, e2e (point to face & edge to edge)
- Multiple simultaneous collisions

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [19]: Moving Mesh vs. Moving Mesh

- Two options: 'point sample' and 'extrusion'
- Point sample:
  - If objects intersect at final positions, do a binary search backwards to find the time when they first hit and compute the intersection
  - This approach can tend to miss thin objects
- Extrusion:
  - Test '4-dimensional' extrusions of objects
  - In practice, this can be done using only 3D math

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Collision Detection [20]: Moving Meshes: Extrusion

- Use ‘delta matrix’ trick to simplify problem so that one mesh is moving and one is static
- Test moving vertices against static faces (and the opposite, using the other delta matrix)
- Test moving edges against static edges (moving edges form a quad (two triangles))

Collision Detection [21]: Intersection Issues

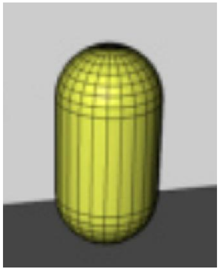
- Performance
- Memory
- Accuracy
- Floating point precision

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

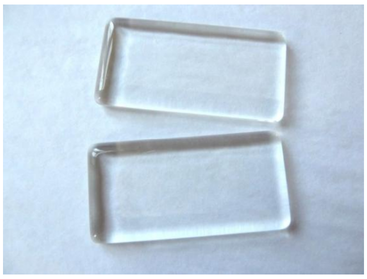
Static Intersection [1]: Sphere-Swept Volumes

- Sphere
  - Locus of points in 3-D equidistant from center point
  - Rotational sweep of circle (hollow sphere) or disc (solid ball)
  - "Null" sweep of sphere (invariant under rotation, translation by $\mathbf{0}$)
- Capsule: Translational Sweep of Sphere Along Line Segment
- Lozenge: Sweep of Sphere Across Rectangle

Wikipedia: *Sphere*
http://bit.ly/9OWjQi
Image © 2008 ClipArtOf.com
http://bit.ly/eKhE2f



Capsule
Image © 2007 Remotion Wiki
http://bit.ly/huEzNW



Lozenge
Image © 2011 Jasmin Studio Crafts
http://bit.ly/euEopw

Adapted from *3D Game Engine Design* © 2000 D. H. Eberly
See http://bit.ly/ieUq45 for second edition table of contents (TOC)

Static Intersection [2]: Distance Calculators

Table 6.1 Relationship between sphere-swept volumes and distance calculators (*pnt*, point; *seg*, line segment; *rct*, rectangle).

	Sphere	*Capsule*	*Lozenge*
Sphere	dist(pnt,pnt)	dist(pnt,seg)	dist(pnt,rct)
Capsule	dist(seg,pnt)	dist(seg,seg)	dist(seg,rct)
Lozenge	dist(rct,pnt)	dist(rct,seg)	dist(rct,rct)

3D Game Engine Design © 2000 D. H. Eberly
See http://bit.ly/ieUq45 for second edition table of contents (TOC)

Dynamic Intersection [1]: One Moving Object

Table 6.6 Relationship between sphere-swept volumes and distance calculators when the second object is moving (*pnt*, point; *seg*, line segment; *rct*, rectangle; *pgm*, parallelogram; *ppd*, parallelepiped; *hex*, hexagon).

	Dynamic		
	Sphere	*Capsule*	*Lozenge*
Static			
Sphere	dist(pnt,{pnt,seg})	dist(pnt,{seg,pgm})	dist(pnt,{rct,hex,ppd})
Capsule	dist(seg,{pnt,seg})	dist(seg,{seg,pgm})	dist(seg,{rct,hex,ppd})
Lozenge	dist(rct,{pnt,seg})	dist(rct,{seg,pgm})	dist(rct,{rct,hex,ppd})

See http://bit.ly/ieUq45 for second edition table of contents (TOC)

Dynamic Intersection [2]: Two Moving Objects – Separating Axes

Table 6.7 Values for R, R_0, and R_1 for the separating axis test $R > R_0 + R_1$ for two boxes in the direction of motion.

$\vec{L}$	R_0	R_1	R
$\vec{W} \times \vec{A}_0$	$a_1\|\alpha_2\| + a_2\|\alpha_1\|$	$\sum_{i=0}^{2} b_i\|c_{1i}\alpha_2 - c_{2i}\alpha_1\|$	$\|\vec{A}_0 \cdot \vec{W} \times \vec{D}\|$
$\vec{W} \times \vec{A}_1$	$a_0\|\alpha_2\| + a_2\|\alpha_0\|$	$\sum_{i=0}^{2} b_i\|c_{0i}\alpha_2 - c_{2i}\alpha_0\|$	$\|\vec{A}_1 \cdot \vec{W} \times \vec{D}\|$
$\vec{W} \times \vec{A}_2$	$a_0\|\alpha_1\| + a_1\|\alpha_0\|$	$\sum_{i=0}^{2} b_i\|c_{0i}\alpha_1 - c_{1i}\alpha_0\|$	$\|\vec{A}_2 \cdot \vec{W} \times \vec{D}\|$
$\vec{W} \times \vec{B}_0$	$\sum_{i=0}^{2} a_i\|c_{i1}\beta_2 - c_{i2}\beta_1\|$	$b_1\|\beta_2\| + b_2\|\beta_1\|$	$\|\vec{B}_0 \cdot \vec{W} \times \vec{D}\|$
$\vec{W} \times \vec{B}_1$	$\sum_{i=0}^{2} a_i\|c_{i0}\beta_2 - c_{i2}\beta_0\|$	$b_0\|\beta_2\| + b_2\|\beta_0\|$	$\|\vec{B}_1 \cdot \vec{W} \times \vec{D}\|$
$\vec{W} \times \vec{B}_2$	$\sum_{i=0}^{2} a_i\|c_{i0}\beta_1 - c_{i1}\beta_0\|$	$b_0\|\beta_1\| + b_1\|\beta_0\|$	$\|\vec{B}_2 \cdot \vec{W} \times \vec{D}\|$

3D Game Engine Design © 2000 D. H. Eberly
See http://bit.ly/ieUq45 for second edition table of contents (TOC)

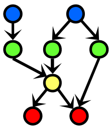
Preview: Collision Response & Optimization

- **What Happens After Collision Is Detected?**
  - Contact & application of force *vs.* impact & impulse
  - Compression: deformation of solid
  - Restitution: springing back of solid
  - Friction?
  - Secondary collisions due to changes in trajectories
  - Bouncing?
- **Optimization**
  - Spatial partitioning: bounding volume hierarchies (BVHs) revisited
    - Binary space partitioning (BSP) trees
    - *k*-d trees
    - Quadtrees & octrees
  - Volume graphics: uniform grids and data parallelism

Adapted from slides ♥ 2004 – 2005 S. Rotenberg, UCSD
CSE169: Computer Animation, Winter 2005, http://bit.ly/f0ViAN

Summary

- Reading for Last Class: Chapter 10, 13, §17.3 – 17.5, Eberly 2^e
- Reading for Today: §2.4.3, 8.1, Eberly 2^e, GL handout
- Reading for Next Class: Chapter 6, Esp. §6.1, Eberly 2^e
- Last Time: Quaternions Concluded
  - How quaternions work – properties, matrix equivalence, arithmetic
  - Composing rotations by quaternion multiplication
  - Incremental rotation and error issues
- Review: Virtual Reality & Virtual Environments; Augmented Reality
- Today: Collision Detection Part 1 of 2
  - Static: stationary objects (both not moving)
  - Dynamic: moving objects (one or both)
  - Test-intersection queries *vs.* find-intersection queries
  - Distance *vs.* intersection methods



Terminology

- **Visualization – Communicating with Images, Diagrams, Animations**
- **VR, VE, VA, AR**
  - **Virtual Reality: computer-simulated environments, objects**
  - **Virtual Environment: part of VR dealing with surroundings**
  - **Virtual Artifacts: part of VR dealing with simulated objects**
  - **Augmented Reality: CG sensory overlay on real-world images**
- **Collision Detection**
  - **Static: stationary objects (both not moving)**
  - **Dynamic: moving objects (one or both)**
  - **Queries**
    - **Test-intersection: determine whether objects do/will intersect**
    - **Find-intersection: calculate intersection set or contact set, time**
  - **Parametric methods: use parameters to describe objects**
    - **Distance-based: constrained minimization (closest points)**
    - **Intersection-based: solving for parameters in equation**