# CG – T11 – Collision Detection

L:CC, MI:ERSI

***Miguel Tavares Coimbra***

***(course and slides designed by Verónica Costa Orvalho)***

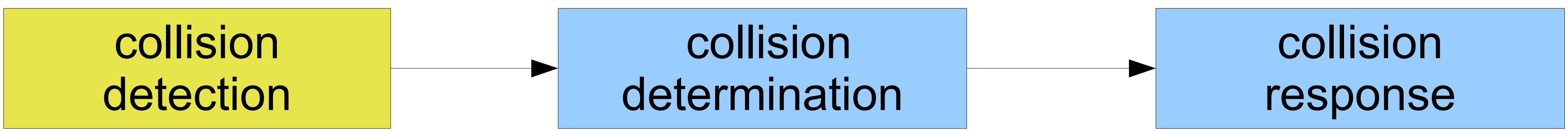
U. PORTO FC

# agenda

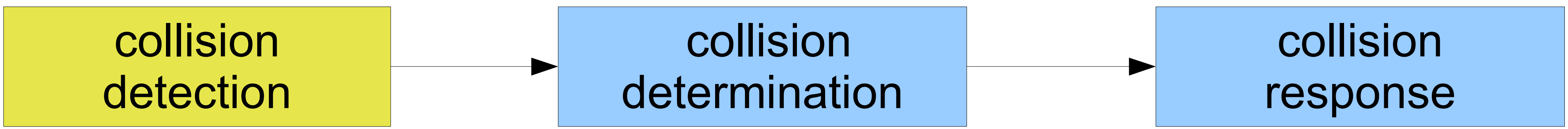
- introduction
- collision detection pipeline
- algorithms
- demos

# introduction

## collision handling:



# collision handling




result is a boolean

**Object Has Collision?**
(yes,no)

# collision handling


result is a boolean

**<u>Object Has Collision?</u>**
(yes,no)

finds the **intersection** between objects

# collision handling

collision detection → collision determination → collision response

**collision detection**

result is a boolean

**<u>Object Has Collision?</u>**
(yes,no)

**collision determination**

finds the **intersection** between objects

**collision response**

determines what actions should be taken in **response** to the collision of 2 or more objects.

# what you need to know

- Basic geometry
  - vectors, points, homogenous coordinates, affine transformations, dot product, cross product, vector projections, normals, planes
- math helps…
  - Linear algebra, calculus, differential equations

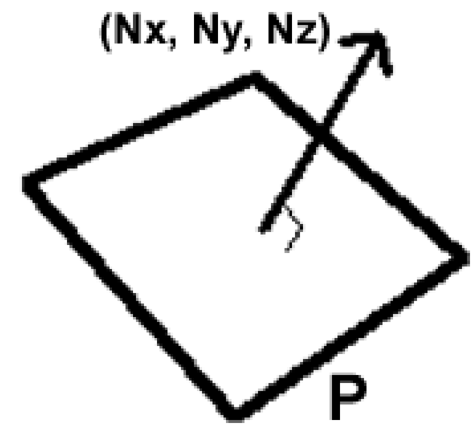
# Calculating Plane Equations

- A 3D Plane is defined by a normal and a distance along that normal
- Plane Equation: $(Nx, Ny, Nz) \bullet (x, y, z) + d = 0$
- Find d: $(Nx, Ny, Nz) \bullet (Px, Py, Pz) = -d$
- For test point (x,y,z), if plane equation

  > 0: point on 'front' side (in direction of normal),

  < 0: on 'back' side

  = 0: directly on plane
- 2D Line 'Normal': negate rise and run, find d using the same method

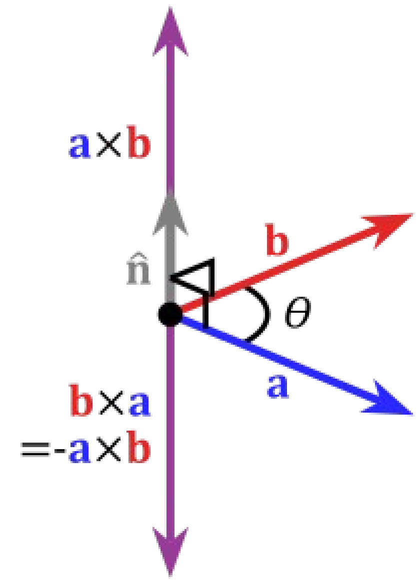


# Cross Product

```
point point::operator^(point p)

   point res;
   res.x = y*p.z  -  z*p.y;
   res.y = z*p.x  -  x*p.z;
   res.z = x*p.y  -  y*p.x;

return res;
```




**<<depends on the choice of orientation>>**

# Dot Product

```
double point::operator*(point p)
{
   return (p.x*x  +  p.y*y  +  p.z*z);
}
```

# So where do you start….?

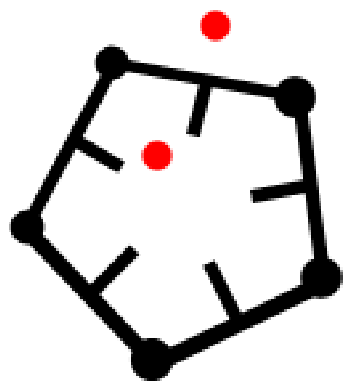
- First you have to detect collisions
  - With discrete timesteps, every frame you check to see if objects are intersecting (overlapping)
- Testing if your model's actual volume overlaps another's is too slow
- Use bounding volumes (BV's) to approximate each object's real volume

slide adapted Ryan Schmidt, U. Calgary

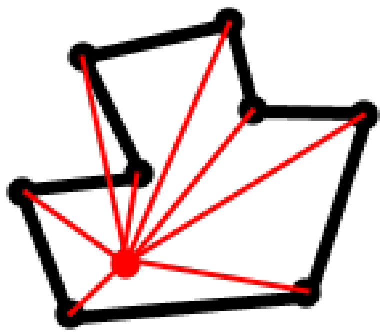
# Bounding Volumes?

- Convex-ness is important*
- **spheres, cylinders, boxes**, polyhedra, etc.
- Spheres are mostly used for fast culling
- For boxes and polyhedra, most intersection tests start with point inside-outside tests
  - That's why convexity matters. There is no general inside-outside test for a 3D concave polyhedron.

# 2D Point Inside-Outside Tests



- Convex Polygon Test
  - Test point has to be on same side of all edges
- Concave Polygon Tests
  - 360 degree angle summation
  - Compute angles between test point and each vertex, inside if they sum to 360
  - Slow, dot product and acos for each angle!



- Other methods:
  Quadrant Method (see Gamasutra Article)
  Edge Cross Test (see Graphics Gems IV)

slide adapted Ryan Schmidt, U. Calgary

# Closest point on a line

- Handy for all sorts of things…

$$A = P_2 - P_1$$

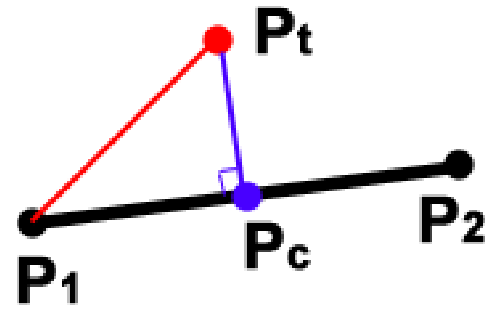
$$B = P_1 - P_t$$

$$C = P_2 - P_t$$

$$if\,(A \bullet B \leq 0) \quad P_c = P_1$$

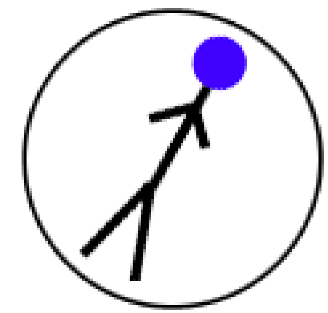
$$else\, if\,(A \bullet C \leq 0) \quad P_c = P_2$$

$$else \quad P_c = P_1 + \frac{(P_1 - P_1) * (B \bullet A)}{(B \bullet A) + (C \bullet A)}$$




slide adapted Ryan Schmidt, U. Calgary

# Spheres as Bounding Volumes

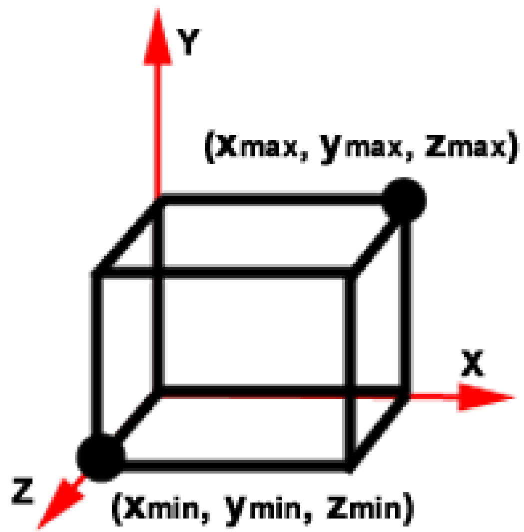


- Simplest 3D Bounding Volume
  - Center point and radius
- Point in/out test:
  - Calculate distance between test point and center point
  - If distance <= radius, point is inside
  - You can save a square root by calculating the squared distance and comparing with the squared radius !!!
  - (this makes things a lot faster)
- It is **ALWAYS** worth it to do a sphere test before any more complicated test. **ALWAYS**. I said **ALWAYS**.

# Axis-Aligned Bounding Boxes

- Specified as two points:

$$(x_{\min}, y_{\min}, z_{\min}), (x_{\max}, y_{\max}, z_{\max})$$




- Normals are easy to calculate
- Simple point-inside test:

$$x_{\min} \leq x \leq x_{\max}$$

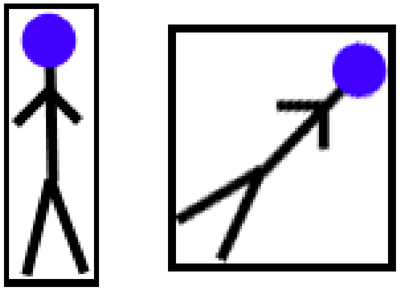
$$y_{\min} \leq y \leq y_{\max}$$

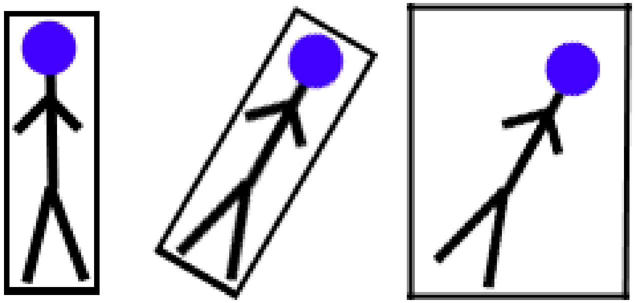
$$z_{\min} \leq z \leq z_{\max}$$

slide Ryan Schmidt, U. Calgary

# Problems With AABB's

- Not very efficient
- Rotation can be complicated
  - Must rotate all 8 points of box
  - Other option is to rotate model and rebuild AABB, but this is not efficient

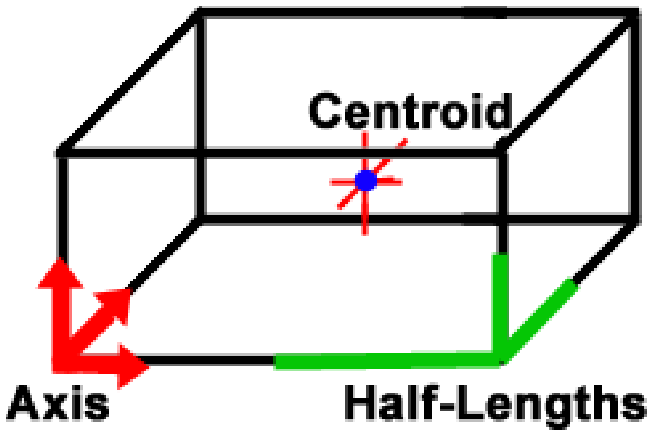


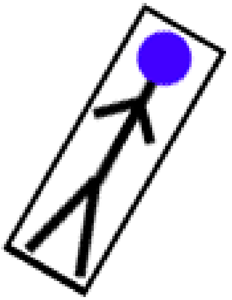


slide Ryan Schmidt, U. Calgary

# Oriented Bounding Boxes

- Center point, 3 normalized axis, 3 edge half-lengths
- Can store as 8 points, sometimes more efficient
  - Can become not-a-box after transformations
- Axis are the 3 face normals
- Better at bounding than spheres and AABB's

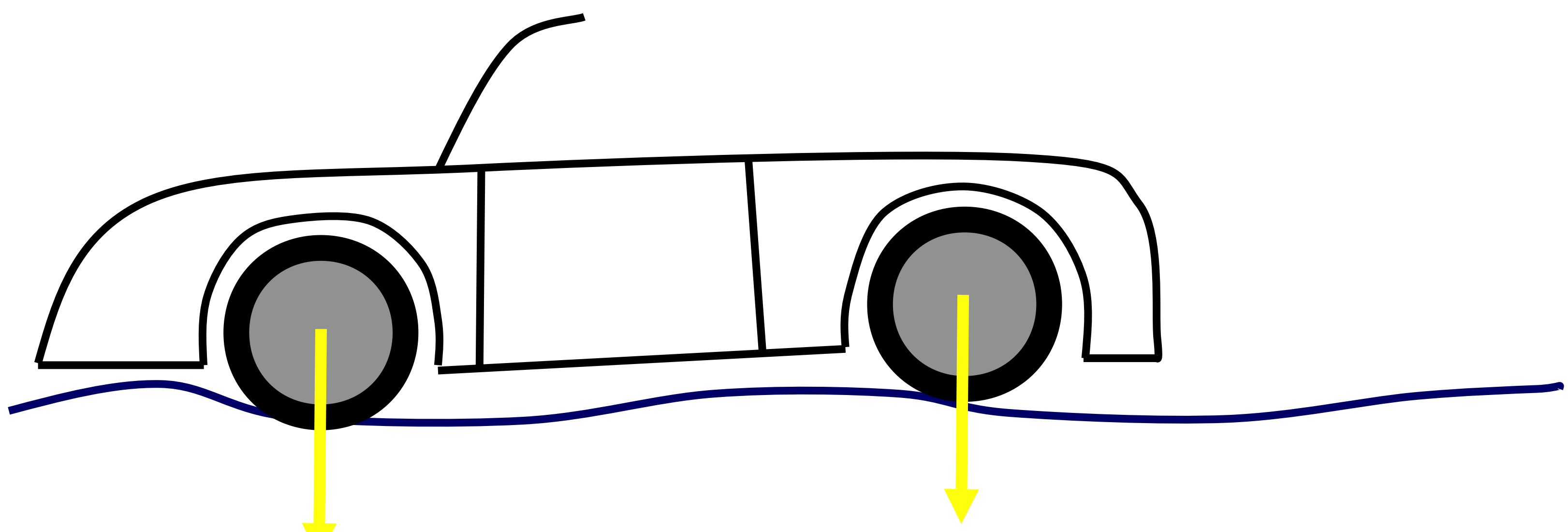


slide Ryan Schmidt, U. Calgary

# simple collision detection

- only shoot rays to find collisions, i.e., approximate an object with a set of rays
- cheaper, but less accurate
- Test for **point in plane or point in sphere**



# simple collision detection

- only shoot rays to find collisions, i.e., approximate an object with a set of rays
- cheaper, but less accurate
- **Test: point inside sphere**

$(X-Xc)2 + (Y-Yc)2 + (Z-Zc)2 = R2$


point P    P=(px,py,pz)

P is inside the sphere if and only if

Inside= (sqrt ( (px-xc)2 + (py-yc)2 + (pz-zc)2 ) < radius

# Collision Detection Packages

* Collision Detection Packages from UNC Chapel Hill (this is an extensive, ever-growing collection).

* Bullet Physics Library - library for performing rigid-body collision detection and response. Open source and free for commercial use, and is integrated with Blender and COLLADA. video

* SOLID - Software Library for Interference Detection. Now a commercial product, and GPL'ed with source available.

* V-clip - a low level object collision library.

* OPCODE - more memory-friendly and often faster than SOLID and RAPID, free for reuse in any application.

* ODE - a free rigid body dynamics package which includes collision detection.

* ColDet - a free collision detection library for generic polyhedra.

* Havok - the most popular commercial library for games is free for non-commercial use.

# conclusion

- cannot test every pair of triangles:  $O(n^2)$
- use BVs because these are cheap to test
- better:  use a hierarchical scene graph

# Cool Demos

- Watch 3000 barrels fall down in Crysis
  http://kotaku.com/gaming/clips/watch-3000-barrels-fall-down-in-crysis-333902.php
- The most epic GMode
  http://www.wegame.com/watch/The_most_epic_GMod_Rube_Goldberg_video_ever/

# references

- Real Time Rendering, chapter 17 (the book)
  - http://www.realtimerendering.com
- Journal of Graphics Tools
  - http://www.acm.org/jgt/
- Bulletphysics Library
  - http://bulletphysics.org/wordpress/