Responses to 3D collision (without kinetics-style physics)

Similar considerations as with 2D collisions

- elastic bounce (but in 3D)
- deformation or fragmentation (but in 3D)
- penetrable objects (clouds, smoke)
- gameplay effects (wake monster, play music, change level, gain power-up, . . .)

Handling bounce (without real physics simulation: with *kinematics*) often involves

- determine instant and velocity (speed & direction) of impact between meshes
  - repeated subdivision of interval between frames gives approximate instant
- or, perhaps with Gilbert-Johnson-Keerthi algorithm (GJK) (for convex objects)
  - (see eg codezealot.org/archives/88 tutorial and link to 52' video on GJK)
  - determine fact of, then depth of, objects' interpenetration in next frame
  - move objects apart by that depth
- adjust velocities and recalculate next-frame positions

3 Stages of collision handling

Prologue stage

- callback function determines if motion-alteration response is needed
- prologue may also trigger eg sound effects, by notifying colliding (sw) objects

Collision stage (only needed if prologue has determined a response is needed)

- place objects at point of impact
- assign new velocities (with physics or otherwise)

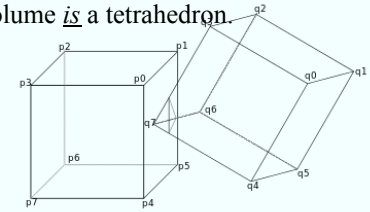
Epilogue stage

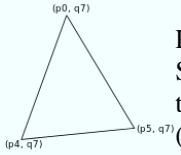
- handle post-collision effects, eg sound effects, destruction, fragmentation
- again by notifying affected (sw) objects

It is somewhat arbitrary what effects are caused by prologue or by epilogue, it is no accident that 'sound effects' is given as an example in both cases.

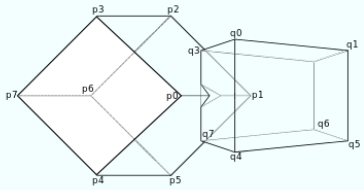
Object interpenetration (GJK)

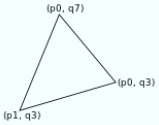
Where 3D objects do interpenetrate, some tetrahedron can be found which includes some of the volume shared by the two objects. *Sometimes* the whole of the shared volume *is* a tetrahedron.





Point-to-face collision
Successively approximate a tetrahedron of shared volume (from Wikipedia)





Edge-to-edge collision
Successively approximate a tetrahedron of shared volume

Minkowski Sum , Difference

The Minkowski Sum of two shapes (area, volumes) is the shape that results from adding one shape to another – as seen previously in circle-rectangle overlap

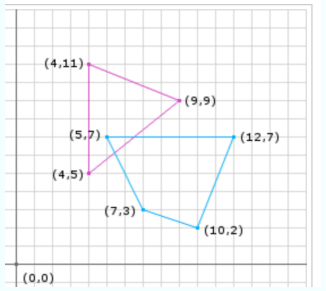
One of the shapes may be *subtracted* rather than *added* – "Minkowski Difference" is not really an approved term but it suggests the appropriate concept – yielding a "Configuration Space Object" (CSO) = "Minkowski Configuration Object"

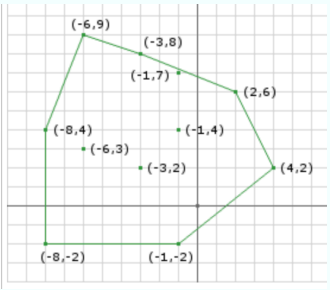
Two **convex** shapes overlap if this Minkowski Difference encompasses the origin

The workings for the example in http://www.dyn4j.org/2010/04/gjk-gilbert-johnson.

Observe that point -3,2 occurs twice in the sum

4,11	-	5,7	=	-1,4
	-	12,7	=	-8,4
	-	10,2	=	-6,9
	-	7,3	=	-3,8
9,9	-	5,7	=	4,2
	-	12,7	=	-3,2
	-	10,2	=	-1,7
	-	7,3	=	2,6
4,5	-	5,7	=	-1,-2
	-	12,7	=	-8,-2
	-	10,2	=	-6,3
	-	7,3	=	-3,2





GJK "Support Function"

A "Support Function" takes a direction and a shape, returns extreme point of shape

GJK iteratively finds "simplexes" progressively more likely to encompass origin

GJK algorithm converges on direction, and hence depth, of maximum penetration
which may be useful in deciding on direction & magnitude of
(kinematics) displacement to achieve separation
(kinetics) direction and strength of correctional "penalty force"

However this direction may not be the right one for deciding appropriate response!

Kinetics and Physics Engines

Besides having a graphically represented boundary, rigid objects are modelled as **particles** with **mass**, **centre of mass**, **moments of inertia**
(which relate torque to angular acceleration; one per axis)

- Numerical integration is required: stability of numerical integration may be an issue
  - Force *f* yields acceleration *a*; (angular accelerations too)
  - integrate *a* to get velocity *v*; (rotational velocities too)
  - integrate *v* to get position *p*; (rotational positions too)
- Newton's Laws of Motion
  - bodies move steadily unless a force is applied
  - bodies accelerate in the direction of a force applied, at a rate proportional to it
  - to every action there is an equal and opposite reaction
- Conservation of Momentum (and angular momentum)

Non rigid body dynamics

Not all game objects will be rigid bodies:

Static "kinematic" objects, that affect game physics but are not affected by it, may perhaps be modelled as having infinite mass, but most likely not because of problems:

- how to conserve momentum? The immobile infinite mass won't have any.
- what if kinematic objects are not static, but are to move? Infinite forces

Objects that have mass and mobility but do not keep their shape:

- Soft body objects, that deform;
- Cloth, that folds and stretches and tears;
- Fluids, that slosh and splash;
- Brittle objects that shatter.
  - **Links between particles** may be employed, or special purpose materials

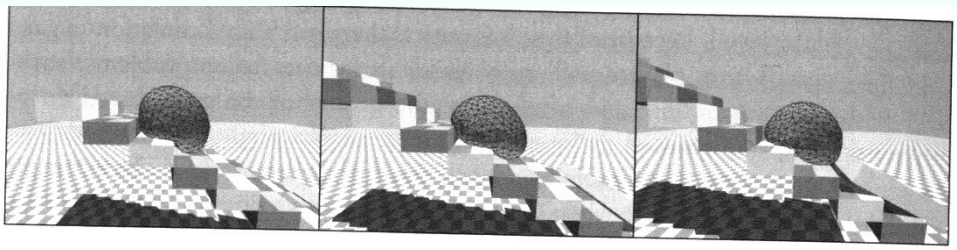


FIGURE 4.3.8 *A partially inflated ball, a soft body, rolls down the stairs.*

This picture scanned from "Introduction to Game Development" 2nd ed by Steve Rabin

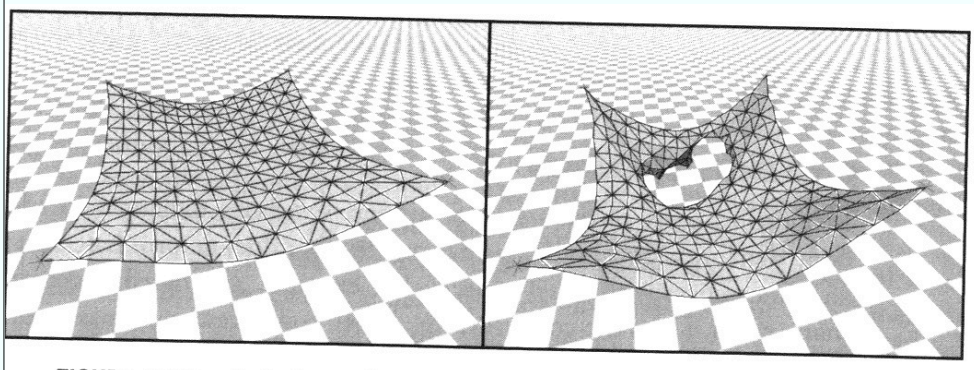


FIGURE 4.3.9 *A cloth panel, pinned at each corner, sits above a game world. In the right frame, the cloth was allowed to tear due to excessive forces.*

This picture scanned from "Introduction to Game Development" 2nd ed by Steve Rabin

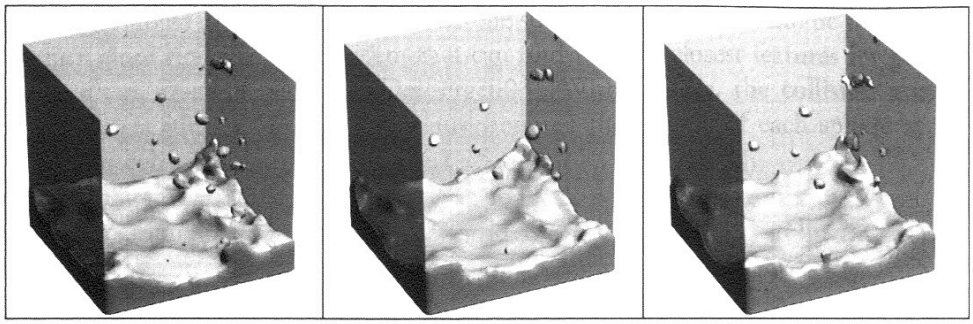


FIGURE 4.3.10 *Fluid simulation based on the smoothed-particle hydrodynamics (SPH) technique.*

This picture scanned from "Introduction to Game Development" 2nd ed by Steve Rabin

Issues to avoid in physics simulation

Game crash

- which can be caused by numerical integration over discrete time steps with bizarre results - infinite forces, objects squashed into negative volume, …

Unrealistic collision behaviour

- energy and direction of object movements after collision should be believable

Inaccurate friction behaviour

- friction should not cause objects to start moving in reverse

Framerate drop

- physics simulation time step can be independent of graphics rendering time step

Unrealistic oscillations

- dropped objects should not lie forever chattering on the floor

Physics simulation advantages

- Reduces artist workload
  - A game production team typically has members with variety of skills and associated tasks. Artists are important.
  - Artwork can be very costly (time AND money)
- Artwork must anticipate what is possible in a game, and provide a necessarily limited number of images; physics simulation however can react dynamically to player's choices of actions and create appropriate images as needed
- Physics simulation makes possible the rendering of emergent behaviour, where multiple objects interact in practically unforeseeable ways.
- see http://physxinfo.com/articles/?page_id=389 for one surviving video clip

"Is Blender actually hard to learn?" https://www.youtube.com/watch?v=StTF1c2Jngs

- **Character Cloth Simulation in Blender 2.8 – YouTube**
- *https://www.youtube.com/watch?v=ctXn0YdqkqI*
- **Inflatable Duck simulation (Blender) – YouTube**
- *https://www.youtube.com/watch?v=L2YyxFgMrGY*
- **[Blender] Water Splash Simulation – YouTube**
- *https://www.youtube.com/watch?v=g02Ct_7WwAA*

Physics simulation disadvantages

- Real-time simulation can be very expensive computationally
  - memory may be short, especially for console games
  - see *offline* water simulation at http://code.google.com/p/flip3d/
  - "less than 30 seconds per frame" for 128^3 particles
  - imagine simulating a lake or river that way … don't try!
- Duration of simulation step may be unpredictable, variable
- Integration errors may cause unrealistic behaviours
  - collisions, oscillations, frictional overcompensation

Physics engines (libraries) can be obtained which address these sorts of issue, eg
Havok (commercial, Irish origins!)
Bullet (open source)

Parellelism, GPUs

- Many physics simulation issues are amenable to parallelisation
  - particles of water or other fluids
  - strands of hair
  - patches of cloth
- Using *shader programs* running on GPUs (in video cards) is increasingly used to achieve physics effects, not just graphics effects
  - Vertex Shaders for geometric transformation, vs Pixel Shaders for lighting
  - GPU instruction sets are gradually becoming more general-purpose
  - GPUs have many cores and run many threads
  - Bottleneck often lies in transferring data into and back out of GPU
- Microsoft's DirectX HLSL, Nvidia's CUDA, allow programming of GPUs

Meshes and links

- A 3D polyhedron mesh for physics can be distinct from the graphics mesh
- Links between vertices in physics mesh can be associated with natural length, tensile strength, compressibility, stiffness
- Forces generated by springiness of links can be calculated and used in simulating soft-body deformation, fluid flow, clothing movement
- When tensile strength is exceeded, cloth may tear, droplets may form, Ming Vases may shatter. (Random choice among many candidate links works well)

- Realistic effects obtainable with calculation of stored elastic energy, together with material-appropriate splinter shapes along precalculated fracture surfaces
- Fracture cascade can give a vibrant energetic look, not the deadness of falling apart

http://graphics.berkeley.edu/papers/Parker-RTD-2009-08/Parker-RTD-2009-08.pdf
describes in more depth several issues and how they have been addressed