Basic Game Physics

IMGD 4000

With material from: *Introduction to Game Development*, Second Edition, Steve Rabin (ed), Cengage Learning, 2009. (Chapter 4.3) and *Artificial Intelligence for Games*, Ian Millington, Morgan Kaufmann, 2006 (Chapter 3)

Introduction (1 of 2)

- *What is game physics?*
  - Computing *motion* of objects in virtual scene
    - Including player avatars, NPC's, inanimate objects
  - Computing *mechanical interactions* of objects
    - Interaction usually involves contact (collision)
  - Simulation must be real-time (versus high-precision simulation for CAD/CAM, etc.)
  - Simulation may be very realistic, approximate, or intentionally distorted (for effect)

2

Introduction (2 of 2)

- *And why is it important?*
  - Can improve immersion
  - Can support new gameplay elements
  - Becoming increasingly prominent (expected) part of high-end games
  - Like AI and graphics, facilitated by hardware developments (multi-core, GPU)
  - Maturation of physics engine market

3

Physics Engines

- Similar *buy vs. build* analysis as game engines
  - Buy:
    - Complete solution from day one
    - Proven, robust code base (hopefully)
    - Feature sets are pre-defined
    - Costs range from *free* to *expensive*
  - Build:
    - Choose exactly features you want
    - Opportunity for more game-specification optimizations
    - Greater opportunity to innovate
    - Cost guaranteed to be expensive (unless features extremely minimal)

4

Physics Engines

- Open source
  - Box2D, Bullet, Chipmunk, JigLib, ODE, OPAL, OpenTissue, PAL, Tokamak, Farseer, Physics2d, Glaze
- Closed source (limited free distribution)
  - Newton Game Dynamics, Simple Physics Engine, True Axis, PhysX
- Commercial
  - Havok, nV Physics, Vortex
- Relation to Game Engines
  - Native, e.g,. C4
  - Integrated, e.g., UE4 + PhysX
  - Pluggable, e.g., C4 + PhysX, jME + ODE (via jME Physics)

Basic Game Physics Concepts

- *Why are we studying this?*
  - To use an engine effectively, you need to understand something about what it's doing
  - You may need to implement small features or extensions yourself
  - Cf., owning a car without understanding anything about how it works
- Examples
  - Kinematics and dynamics
  - Projectile motion
  - Collision detection and response

6

Outline

- Introduction (done)
- Kinematics (next)
- The Firing Solution
- Collision Detection
- Ragdoll Physics
- PhysX

7

Kinematics (1 of 3)

- Study of motion of objects *without* taking into account mass or force
- Basic quantities: position, time
- ... and their derivatives: velocity, acceleration
- Basic equations:

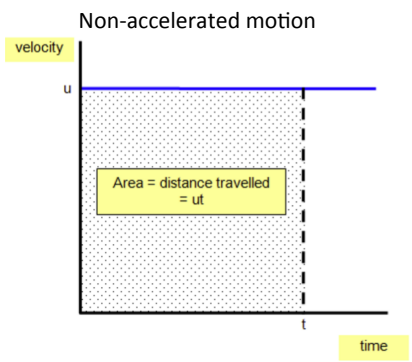
1) $d = vt$
2) $v = u + at$
3) $d = ut + \frac{1}{2}at^2$
4) $v^2 = u^2 + 2ad$

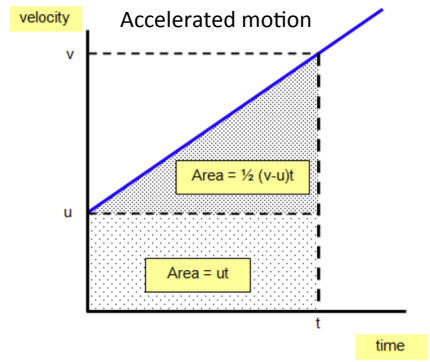
Where:
t - (elapsed) time
d - distance (change in position)
v - (final) velocity (change in distance per unit time)
a - acceleration (change in velocity per unit time)
u - (initial) velocity

Note, equation #3 is the integral of equation #2 with respect to time (see next slide). Equation #4 can be useful.

8

Kinematics (2 of 3)





$d = ut$

Example:

$u = 20$ m/s, $t = 300$ s

$d = 20 \times 3000 = 6000$ m

$d = ut + \frac{1}{2}at^2$

Example:

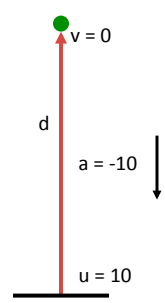
$u=0$ m/s, $a=4$m/s^2, $t=3$s

$d = 0\times3 + 0.5\times4\times9 = 18$m

Kinematics (3 of 3)

Prediction Example: If you throw a ball straight up into the air with an initial velocity of 10 m/sec, how high will it go?

$$v^2 = u^2 + 2ad$$



u = 10 m/sec (initial speed upward)
a = -10 m/sec^2 (approx gravity)
v = 0 m/sec (at top of flight)

$$0 = 10^2 + 2(-10)d$$

d = 5 meters

(Note, answer independent of mass of ball!)

10

Doing It In 3D

- Mathematically, Consider all quantities involving position to be vectors:

$$d = vt$$
$$v = u + at$$
$$d = ut + at^2/2$$

- Computationally, using appropriate 3-element vector datatype

11

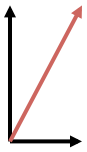
Dynamics

- Notice that preceding kinematic descriptions say nothing about *why* an object accelerates (or why its acceleration might change)
- To get a full "modern" physical simulation you need to add two more basic concepts:
  - force
  - mass
- Discovered by Sir Isaac Newton
- Around 1700 ☺

12

Newton's Laws

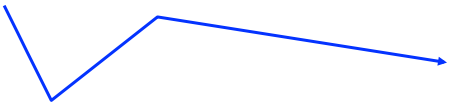
1. A body will remain at rest or continue to move in a straight line at a constant speed unless acted upon by a *force*.
2. The acceleration of a body is *proportional* to the resultant force acting on the body and is in the same direction as the resultant force.
3. For every action, there is an equal and opposite reaction.



13

Motion Without Newton's Laws

- Pac-Man or early Mario style
  - Follow path with *instantaneous changes* in speed and direction (velocity)



  - Not physically possible
- Note - fine for some casual games (especially with appropriate animations)

14

Newton's Second Law

F = m**a**

At each moment in time:

F = force vector, in Newton's

m = mass (intrinsic scalar property of matter), in kg

a = acceleration vector, in m/sec^2

Player cares about state of world (position of objects). Equation is fundamental driver of all physics simulations.

- Force causes acceleration (**a** = **F**/m)
- Acceleration causes change in velocity
- Velocity causes change in position

} Kinematics equations

15

How Are Forces Applied?

- May involve contact
  - Collision (rebound)
  - Friction (rolling, sliding)
- Without contact
  - Rockets/Muscles/Propellers
  - Gravity
  - Wind (if not modeling air particles)
  - Magic
- Dynamic (force) modeling also used for autonomous steering behaviors

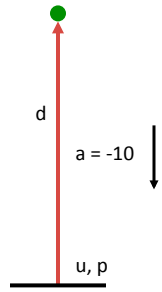
16

Computing Kinematics in Real Time

```
start = getTime()        // start time
p = 0                    // initial position
u = 10                   // initial velocity
a = -10

function update () {     // in game loop
   now = getTime()
   t = now - start
   simulate(t)
}

function simulate (t) {
   d = (u + (0.5 * a * t)) * t
   move object to p + d   // move to loc. computed since start
}
```



$d = ut + at^2/2$

Note! Number of calls and time values to `simulate()` depend on (changing) ***game loop time*** (frame rate)

Is this a problem? It can be! For rigid body simulation with colliding forces and friction (e.g., many interesting cases) … leading to "jerky" behavior

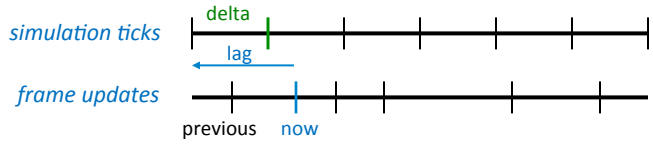
17

Frame Rate Independence

- Complex numerical simulations used in physics engines are sensitive to time steps (due to truncation error and other numerical effects)
- But results need to be repeatable regardless of CPU/GPU performance
  - for debugging
  - for game play
- So, if frame rate drops (game loop can't keep up), then physics will change
- Solution: Control physics simulation interval independently of frame rate

18

Frame Rate Independence




```
start = ...
delta = 0.02     // physics simulation interval (sec)
lag = 0          // time since last simulated
previous = 0     // time of previous call to update

function update() { // in game loop
     now = getTime()
     t = (previous - start) – lag   // previous simulate()
     lag = lag + (now - previous)   // current lag
     while ( lag > delta )          // repeat until caught up
         t = t + delta
         simulate(t)
         lag = lag - delta
     previous = now                 // simulation caught up to current time
}
```

19

Outline

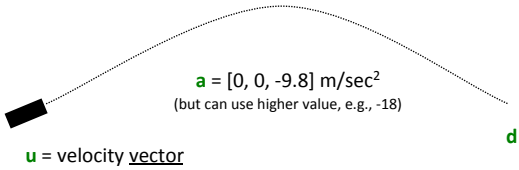
- Introduction (done)
- Kinematics (done)
- The Firing Solution (next)
- Collision Detection
- Ragdoll Physics
- PhysX

20

The Firing Solution (1 of 3)

- How to hit target
  - Beam weapon or high-velocity bullet over short ranges can be viewed as traveling in straight line
  - But projectile travels in parabolic arc
    - Grenade, spear, catapult, etc.

$$d = ut + at^2/2$$



Most typical game situations, *magnitude* of u fixed. We only need to know relative components (orientation) → **Challenge:**

- Given a, d, solve for u

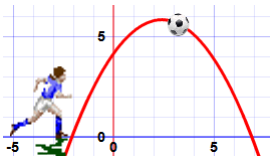
21

Remember Quadratic Equations?

- Make nice curves
  - Like firing at target!
- Solutions are where equals 0. E.g., when firing with gun on ground:
  - At gun muzzle
  - At target
- But unlike in algebra class, not just solving quadratic but finding angle with y = gun, y = target
- Angle changes speed in x-direction, but also time spent in air
- After hairy math [Millington 3.5.3], three relevant cases:
  - Target is out of range (no solution)
  - Target is at exact maximum range (single solution)
  - Target is closer than maximum range (two possible solutions)

this makes it Quadratic

$$5x^2 - 3x + 3 = 0$$

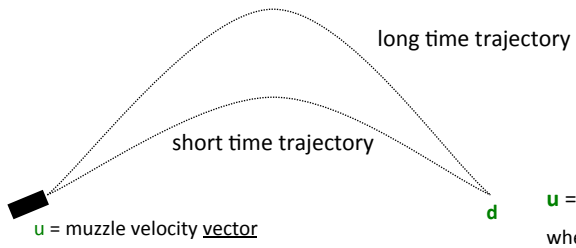


- Solve with quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

22

The Firing Solution (2 of 3)



[Millington 3.5.3]

u = (2**Δ** - **a**t²) / (2 (muzzle_v) t)

where muzzle_v = max muzzle speed

Δ is difference vector from d to u

a is gravity

- Usually choose short time trajectory
  - Gives target less time to escape
  - Unless shooting over wall, etc.

23

The Firing Solution (3 of 3)

```
function firingSolution (start, target, muzzle_v, gravity) {

   // Calculate vector back from target to start
   delta = target - start

   // Real-valued coefficents of quadratic equation
   a = gravity * gravity
   b = -4 * (gravity * delta + muzzle_v * muzzle_v)
   c = 4 * delta * delta

   // Check for no real solutions
   if ( 4 * a * c > b * b ) return null

   // Find short and long times to target
   disc = sqrt (b * b - 4 * a *c)
   t1 = sqrt ( ( -b + disc ) / (2 * a ))
   t2 = sqrt ( ( -b - disc ) / (2 * a ))

   // Pick shortest valid time to target (ttt)
   if ( t1 < 0 ) && ( t2 < 0) return null   // No valid times
   if ( t1 < 0 ) ttt = t2 else
   if ( t2 < 0 ) ttt = t1 else
   ttt = min ( t1, t2 )

   // Return firing vector
   return ( 2 * delta - gravity * ttt * ttt ) / ( 2 * muzzle_v * ttt )
}
```

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Note, a, b and c are scalars so a normal square root.

*Note scalar product of two vectors using *:*

*[a,b,c] * [d,e,f] = a*d + b*e + c*f*

[Millington 3.5.3]

24

Outline

- Introduction (done)
- Kinematics (done)
- The Firing Solution (done)
- Collision Detection (next)
- Ragdoll Physics
- PhysX

25

Collision Detection

- Determining when objects collide is not as easy as it seems
  - Geometry can be complex
  - Objects can be moving quickly
  - There can be *many* objects
    - naive algorithms are $O(n^2)$
- Two basic approaches:
  - Overlap testing
    - Detects whether collision has already occurred
  - Intersection testing
    - Predicts whether collision will occur in future

26

Overlap Testing

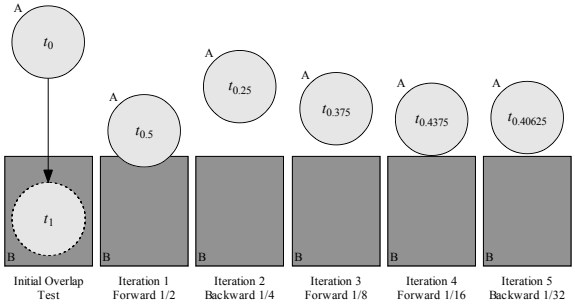
Basics – discussed and implemented in IMGD 3000!

- Most common technique used in games
- Exhibits more error than intersection testing
- Basic idea:
  - at every simulation step, test *every pair* of objects to see if overlap
- Easy for simple volumes (e.g., spheres), harder for polygonal models
- Results of test:
  - collision normal vector (useful for reaction)
  - time that collision took place

27

Overlap Testing: Finding Collision Time

- Calculated by doing "binary search" in time, moving object back and forth by 1/2 steps (bisections)

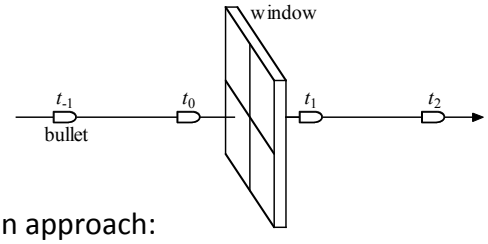


- In practice, five iterations usually enough

28

Limitations of Overlap Testing

- Fails with objects that move too fast (no overlap during simulation time slice)

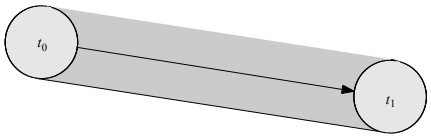


- Solution approach:
  - *constrain game design* so that fastest object moves smaller distance in one physics "tick" (delta) than thinnest object
  - may require reducing simulation step size (adds computation overhead)

29

Intersection Testing

- Predict future collisions
- Extrude geometry in direction of movement
  - e.g., "swept" sphere turns into capsule shape



- Then, see if extruded shape overlaps objects
- When collision found (predicted)
  - Move simulation to time of collision (have collision point)
  - Resolve collision
  - Works for bullet/window example (bullet becomes line segment)

30

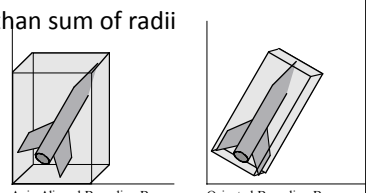
Speeding Up Collision Detection

- Bounding Volumes
  - Oriented
  - Hierarchical
- Partitioning
- Plane Sweep

31

Bounding Volumes

- Commonly used volumes
  - sphere - distance between centers less than sum of radii
  - boxes
    - axis aligned (loose fit, easier math)
    - oriented (tighter fit, more expensive)



- If bounding volumes don't overlap, then no more testing is required
  - If overlap, more refined testing required
  - Bounding volume alone may be good enough for some games

32

Complex Bounding Volumes

- Multiple volumes per object
  - e.g., separate volumes for head, torso and limbs of avatar object
- Hierarchical volumes
  - e.g., boxes inside of boxes

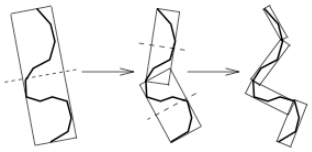


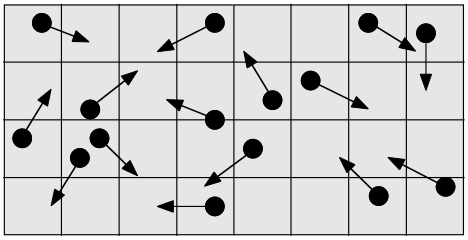
Figure 1: Building the OBBTree: recursively partition the bounded polygons and bound the resulting groups.

[Gottschalk, Lin, Minocha, SIGGRAPH '96]

33

Partitioning for Collision Testing

- To address the n^2 problem...
- Partition space so only test objects in same cell

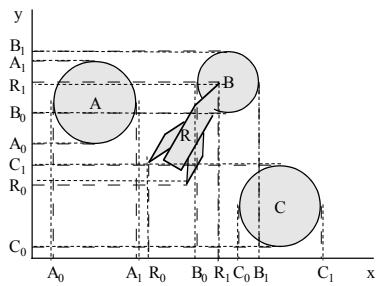


- In best case (uniform distribution) reduces n^2 to linear
  - Can happen for uniform size, density objects (e.g., cloth/fluids)
- In worst case (all objects in same cell) no improvement

34

Plane Sweep for Collision Testing

- *Observation: many* moveable objects stay in one place most of the time
- *Sort* bounds along axes (expensive to do, so do just once!)
- Only adjacent sorted objects which overlap on all axes need to be checked further
- Since many objects don't move, can keep sort up to date very cheaply with bubblesort (nearly linear)



35

Outline

- Introduction (done)
- Kinematics (done)
- The Firing Solution (done)
- Collision Detection (done)
- Ragdoll Physics (next)
- PhysX

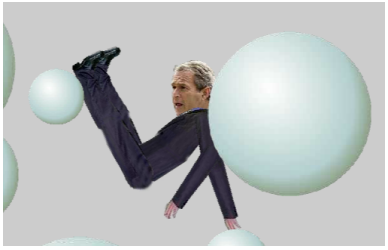
36

What is Ragdoll Physics?

- Procedural animation often used as replacement for traditional (static) death animation
  - Generated by code, not hand
  - Using physics constraints on body limbs & joints in real-time

Still from early animation using ragdoll physics

https://en.wikipedia.org/wiki/Ragdoll_physics



http://www.freeonlinegames.com/game/ragdoll-physics-2

Diablo 3 Ragdolls

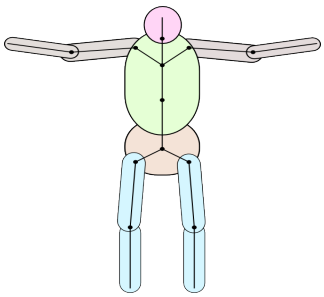
"How to Smack a Demon"

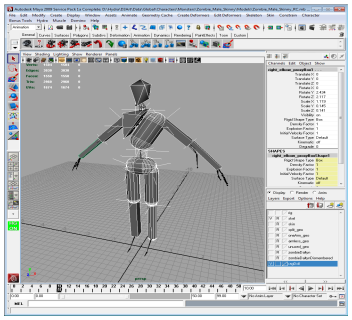
Erin Catto

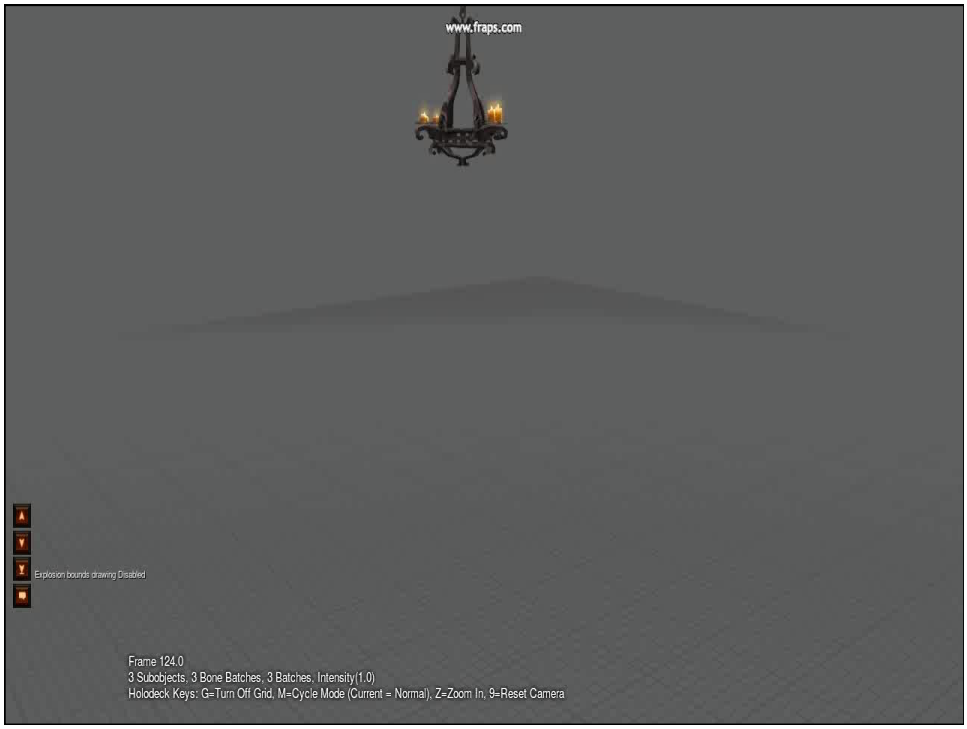
(*Game Developer's Conference*, San Francisco, California, USA, 2013)

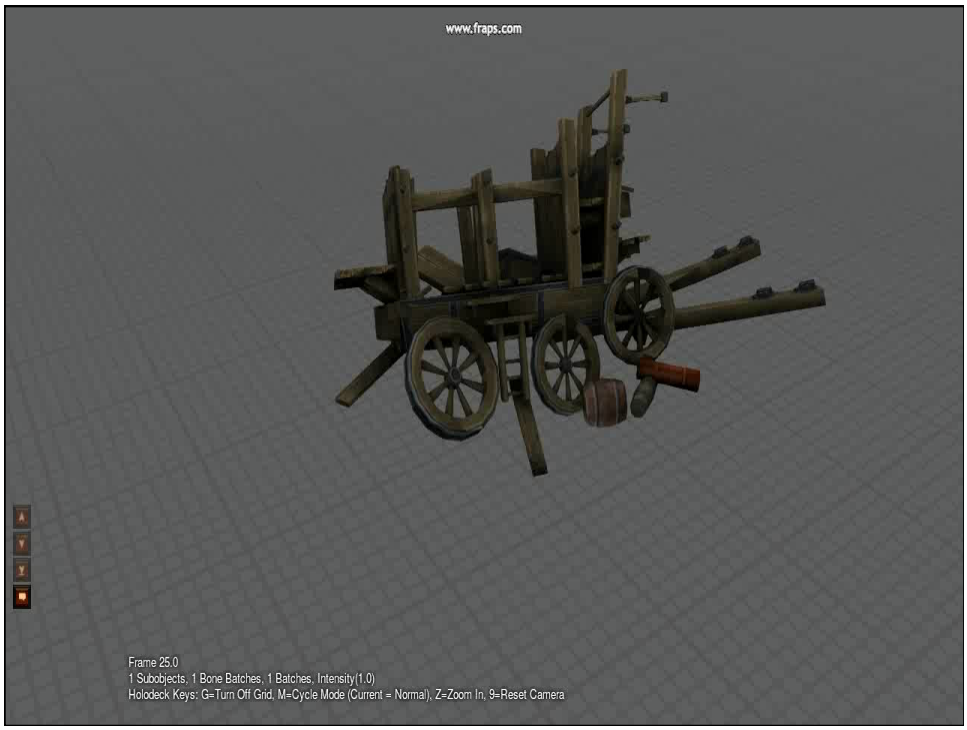
Physics Programmer for Diablo 3 (Blizzard, 2012)

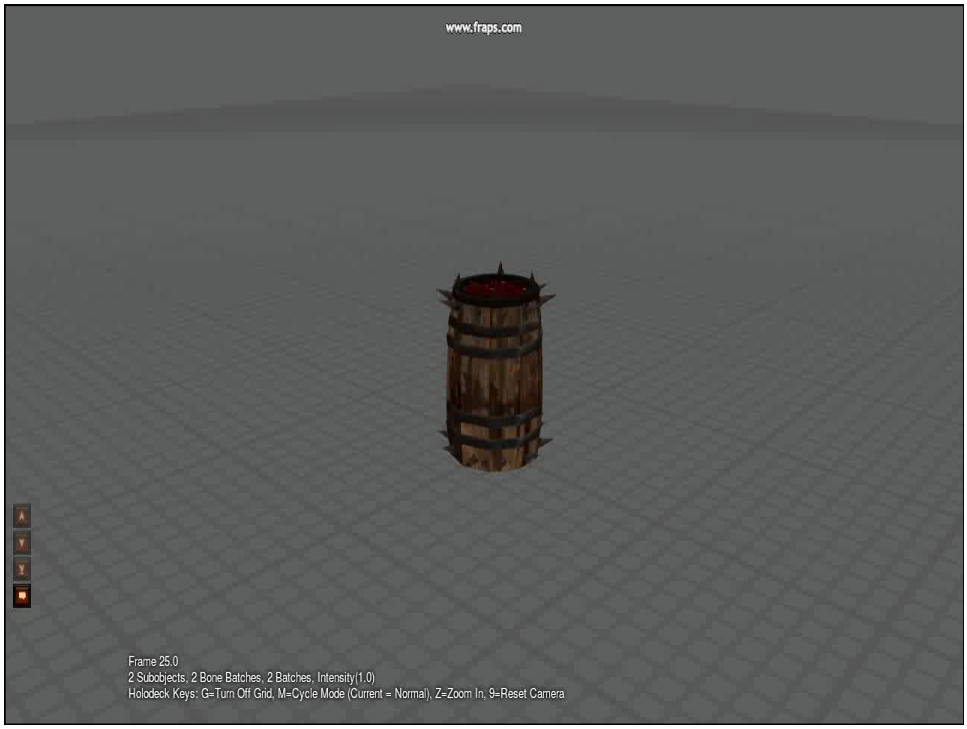
A ragdoll is a collection of collision shapes connected to bones





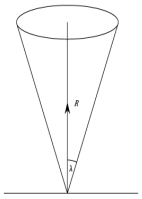

www.fraps.com
Explosion bounds drawing Disabled
Frame 124.0
3 Subobjects, 3 Bone Batches, 3 Batches, Intensity(1.0)
Holodeck Keys: G=Turn Off Grid, M=Cycle Mode (Current = Normal), Z=Zoom In, 9=Reset Camera

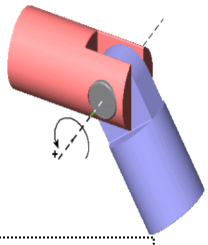

www.fraps.com
Frame 25.0
1 Subobjects, 1 Bone Batches, 1 Batches, Intensity(1.0)
Holodeck Keys: G=Turn Off Grid, M=Cycle Mode (Current = Normal), Z=Zoom In, 9=Reset Camera



Physics joints connect two bones

Cone Joint
(like shoulder)

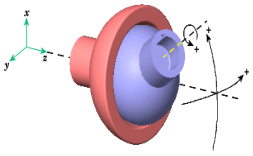




Revolute Joint
(like elbow)

Tech artist connects bones with Physics joints

Spherical Joint
(for chandeliers)



Weld Joint
(locks two bodies, for advanced)

Partial ragdolls add flavor to living characters

Not just for death and destruction

More Physics We Are Not Covering

- Collision response
- Conservation of momentum
- Elastic collisions
- Non-elastic collisions – coefficient of restitution
- Rigid body simulation (vs. point masses)
- Joints as constraints to motion
- Soft body simulation

[see excellent book by Millington, "Game Physics Engine Development", MK, 2007]

47

Outline

- Introduction (done)
- Kinematics (done)
- The Firing Solution (done)
- Collision Detection (done)
- Ragdoll Physics (done)
- PhysX (next)

48

PhysX Overview

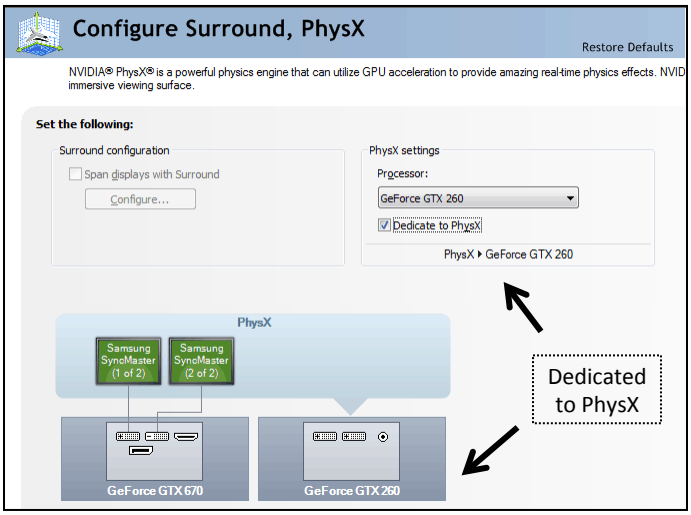
- Developed by NVIDIA for C++ applications
- Windows, Mac, Linux, Playstation, Xbox, Android, Apple iOS and Wii
- Simulate
  - Fluids
  - Soft bodies (cloth, hair)
  - Rigid bodies (boxes, bones)

Why Does NVIDIA Make Physics Software?

- NVIDIA is mainly known as a developer and manufacturer of graphics hardware (GPU's)
- So taking advantage of GPU for hardware acceleration of their physics engine
  - Algorithms can be tuned to their hardware
  - Giving a competitive advantage over other GPU manufacturers

50

Configure Video Card as Dedicated PhysX Processor



What Algorithms Does PhysX Use?

- Hard to know exactly, because algorithm details are NVIDIA's intellectual property (IP)
- However from various forums and clues, it is clear PhysX uses:
  - Both sweep and overlap collision detection
  - AABB and OBBT and (both axis-aligned and oriented bounding bounding box trees)
  - Constraints: hinges, springs, etc.
  - and lots of other hairy stuff, see https://devtalk.nvidia.com/default/board/66/physx-and-physics-modeling/

52

Rocket Sled

CES 2010, **Rocket Sled** demonstrates both graphics and physics computing capabilities of new **GF100** (Fermi) GPUs.

Raging Rapids Ride

Graphics ok, but with **intensive and complex real-time fluid** simulation

Havok Cloth

PhysX competitor bought by Microsoft

- Binary – Free!
- Source code (e.g., to modify), costs

How to Use PhsyX

- General documentation NVIDIA® PhysX® SDK Documentation
  http://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Index.html
- UE4 guide – PhysX, Integrating PhysX Code into Your Project (by Rama)
  https://wiki.unrealengine.com/PhysX,_Integrating_PhysX_Code_into_Your_Project

```
#include "YourGame.h"
#include "HappyDestructible.h"

//~~~~~~~~~~~~~~~~~~~~~~~~~~
//          PhysX
#include "PhysXIncludes.h"
//~~~~~~~~~~~~~~~~~~~~~~~~~~

AHappyDestructible::AHappyDestructible(const class FPostConstructInitializeProperties& PCIP) :
{

}

//Tick
void AHappyDestructible::Tick(float DeltaTime)
{
        Super::Tick(DeltaTime);
        //~~~~~~~~~~~~


        //Draw All Centers to the screen!
        #if WITH_PHYSX
        ScreenMsg("Got into PhysX!!!");
```