Game Physics

John E. Laird

with a few modifications by Matt Evett

Based on The Physics of the Game, Chapter 13 of Teach Yourself Game Programming in 21 Days,
pp. 681-715
Why Physics?

• Some games don’t need any physics.

• Games based on the real world should look realistic, meaning realistic action and reaction.
  • More complex games need more physics:
    • sliding through a turn in a racecar.
    • Running and jumping off the edge of a cliff.

• If you try to do it from scratch, you might get it wrong.

• It is easy to get it right *approximately*
  • For Newtonian physics where \( f = ma \)
  • Rigid bodies

• Not easy:
  • Clothes, pony tails, a whip, chain, volcanoes, boomerang, organics
Computational Physics

• We don’t want just the equations
• We want efficient ways to compute new values
  • Assume fixed discrete simulation – constant time step
  • Add $t_n$ * for variable simulation
• Approach in talk:
  • 2D physics, usually easy to generalize to 3D (add z)
  • Rigid bodies (no deformation)
  • Will just worry about center of mass
    • Not accurate for all physical effects
• Give basic equations at beginning
• Give calculations need in discrete, constant step simulation.
Position and Velocity

• Modeling the movement of objects with velocity
  • Where is an object at any time $t$?
  • Assume our metric is pixels

• Equations:
  • $player_x(t) = t \times x\_velocity + x\_initial$
  • $player_y(t) = t \times y\_velocity + y\_initial$

• Computation:
  • $player_x = player_x + x\_velocity$
  • $player_y = player_y + y\_velocity$
Acceleration

- Acceleration is change in velocity per unit time

Approximate
Acceleration

• Computation:
  • $x_{\text{velocity}} = x_{\text{velocity}} + x_{\text{acceleration}}$
  • $y_{\text{velocity}} = y_{\text{velocity}} + y_{\text{acceleration}}$

• Changing acceleration:
  • use a table based on other factors:
  • $\text{acceleration} = \text{acceleration\_value}(\text{gear}, \text{speed}, \text{pedal\_pressure})$
    • Cheat a bit $\text{acceleration} = \text{acceleration\_value}(\text{gear}, \text{speed}) \times \text{pedal\_pressure}$
  • $x_{\text{acceleration}} = \cos (v) \times \text{acceleration}$
  • $y_{\text{acceleration}} = \sin (v) \times \text{acceleration}$

• Piece-wise linear approximation to continuous functions
Gravity

- Gravity is a force between two objects:
  - Force $F = G \times \frac{(M1 \times M2)}{D^2}$
    - $G =$ Gravitational constant
    - $D =$ Distance between the two objects
  - So both objects have same force applied to them
    - $F=MA \rightarrow A=F/M$

- On earth, assume mass of earth is so large it doesn’t move, and $D$ is relatively constant
  - Assume uniform acceleration
Gravity

• Equation:
  • $V(t) = \frac{1}{2}g*t^2$
  • $g = 9.8 \text{ m/s}^2$ or $32 \text{ft/s}^2$

• Computation
  • $x\_velocity = x\_velocity + 0$
  • $y\_velocity = y\_velocity + gravity$
    • gravity must be normalized to time slice of game (or so that it looks “good”)
Space Game Physics

• Gravity
  • Influences both bodies
  • Can have two bodies orbit each other
  • Only significant for large mass objects

• What happens after you apply a force to an object?

• What happens when you shoot a missile from a moving object?

• What types of controls do you expect to have on a space ship?

• What about a flying game?
Friction

- Conversion of kinetic energy into heat
- Frictional Force = \( C \times G \times M \)
  - \( C \) = frictional coefficient = amount of force to maintain a constant speed
  - \( G \) = gravity
  - \( M \) = mass
- velocity = velocity - friction
  - For velocity > friction!
- Usually two frictional forces
  - Static friction when at rest. If velocity = 0. No movement unless overcome.
  - Kinetic friction, when moving (< static friction)
Race Game Physics

• Non-linear acceleration
• Resting friction > rolling friction
• Rolling friction < sliding friction
• Centripetal force?

• What controls do you expect to have for a racing game?
  • Turning requires forward motion!

• What about other types of racing games
  • Boat?
  • Hovercraft?
Projectile Motion

- Forces

\[ V_i = \text{initial velocity} \]
\[ V_{xi} = \cos(A) \cdot V_i \]
\[ V_{yi} = \sin(A) \cdot V_i \]
\[ V_x = V_{x} - WR(V_{x}) \]
\[ V_y = V_{y} - WR(V_{y}) + G \]

\[ X = x + V_x + W \]
\[ Y = y + V_y \]

\[ M: \text{mass of projectile} \]
\[ A: \text{angle of inclination} \]

\[ G: \text{gravity} \]

\[ WR: \text{wind resistance} \]

\[ W: \text{wind} \]
Back to Collisions

- Steps of analysis for different types of collisions
- Different types of collisions
  - Circle/sphere against a fixed, flat object
  - Two circles/spheres
  - Rigid bodies
  - Deformable
- Model the simplest - don’t build a general engine
Collisions: Steps of Analysis

• Detect that a collision occurred
• Determine the time of the collision
  • So can back up to point of collision
• Determine where the objects are when they touch
• Determine the collision normal
• Determine the velocity vectors after collision
• Determine changes in rotation
Circles and Lines 1

- Simplest case
  - Good step for your games - pinball
  - Assume circle hitting an *immovable* barrier

- Detect that a collision occurred
  - If the distance from the circle to the line < circle radius
  - Reformulate as a point about to hit bigger walls
  - If vertical and horizontal walls, simple test of x, y.
Circles and Lines 2

- What if more complex background: pinball?
  - For complex surfaces, pre-compute and fill an array with collision points (and surface normal).
Circles and Lines 3

- Determine the time of the collision
  - \( t_c = \frac{(x_h-x_1)}{(x_2-x_1)} \times dt + t_i \)
  - \( dt = \) delta time = time increment
  - \( t_i = \) initial time
  - \( t_c = \) collision time

- Determine where the objects are when they touch
  - \( y_c = y_1 - (y_1-y_2) \times t_c \); \( x_c = x_h \)

- Determine the collision normal
  - Angle of line using \((x_1-x_h)\) and \((y_1-y_c)\)
Circles and Lines 4

• Determine the velocity vectors after collision
  • Orthogonal to collision normal
  • Vertical - change sign of x velocity
  • Horizontal - change sign of y velocity
  • Corner - change sign of both
  • Other - invert velocity at collision normal

• Compute new position
  • Use dt - tc to calculate new position from collision point

• Determine changes in rotation
  • None! Unless we really want to add complexity of rotational dynamics

• How to “invert velocity” at collision…
Angle of Reflection

- Angle of incidence = angle of reflection
- Normalize the vectors (divide vector by its own length) so $|l|=|n|=1$
- We want $|r|=1$
- Since $\theta_i = \theta_r$, $\cos \theta_i = \cos \theta_r$
- Dot product: $\cos \theta_i = l \cdot n = \cos \theta_r = n \cdot r$
  - $V1 \cdot V2 = (x1x2+y1y2)$
- Because the vectors are coplanar we can rewrite $r = \alpha l + \beta n$, dot product with $n=$
- $n \cdot r = \alpha l \cdot n + \beta = l \cdot n$
More Reflection…

- $n \cdot r = \alpha l \cdot n + \beta = l \cdot n$
- Because $r$ should be unit length
- $1 = r \cdot r = \alpha^2 + 2\alpha\beta l \cdot n + \beta^2$
- Two equations and unknowns; solving yields
- $r = 2(l \cdot n) n - l$
- $r$ is new unit velocity, “denormalize” by multiplying by length of original velocity, $l$
  - Elastic collision
Circles and Spheres 1

• Another important special case
  • Good step for your games.
  • Many techniques developed here can be used for other object types

• Assume elastic collisions
Detect that a collision occurred

- If the distance between two objects is less than the sum of their radii
  - Trick: avoid square root in computing distance!
  - $(r_1 + r_2)^2 > ((x_1-x_2)^2 + (y_1-y_2)^2)$

- Unfortunately, this is $N^2$ in number of objects
Detect Collision

• With non-circles, gets more complex and more expensive for each pair-wise comparison

• General approach:
  • Observations: collisions are rare.
    • Most of the time, objects are not colliding
  • Create series of filters so that only need to do expensive tests on very few pairs.
Detect that a collision occurred

- Avoid most of the calculations by using a grid:
  - Size of cell = diameter of biggest object
- Test objects in cells adjacent to object’s center
  - Can be computed using mod’s of objects coordinates:
    - bin sort (do you recall run-time of this alg.?)
- Linear in number of objects (usually—if no big clusters)
- For non-circles, use bounding circle/sphere
Detect that a collision occurred

- Alternatively, if many different sizes:
  - Size of cell is arbitrary.
  - Here I used twice size of average object

- Test objects in cells touched by object.
  - Must determine cells object is in.
  - Works for non-circles too.
Detect that a collision occurred

- For non-circles, next test could be to see if bounding circles/spheres overlap
  - Pretty cheap
  - Not great for thin objects
  - If circles overlap, can then test if the objects truly overlap
Circles and Spheres 2

• Determine the time of the collision
  • Interpolate based on old and new positions of objects.

• Determine where objects are when they touch
  • Backup positions to point of collision

• Determine the collision normal
  • Bisects the centers of the two circles at position where they collided
Circles and Spheres 3

- Determine the velocity
  - Assume elastic, and no friction.
  - Assume head on (can generalize to more dimensions)

- Conserve Momentum: Mass * Velocity
  - \( M_1 \cdot V_{i1} + M_2 \cdot V_{i2} = M_1 \cdot V_{f1} + M_2 \cdot V_{f2} \)

- Conservation of Energy (Kinetic Energy)
  - \( M_1 \cdot V_{i1}^2 + M_2 \cdot V_{i2}^2 = M_1 \cdot V_{f1}^2 + M_2 \cdot V_{f2}^2 \)

- Final Velocities
  - \( V_{f1} = \frac{2 \cdot M_2 \cdot V_{i2} + V_{i1} \cdot (M_1 - M_2)}{M_1 + M_2} \)
  - \( V_{f2} = \frac{2 \cdot M_1 \cdot V_{i1} + V_{i2} \cdot (M_1 - M_2)}{M_1 + M_2} \)
    - What if equal mass, \( M_1 = M_2 \)
    - What if \( M_2 \) is infinite mass?
Alternative Formulation

- Let $u_1$ and $u_2$ be the initial velocities, and $v_1$ and $v_2$ the final

\[
\begin{align*}
v_1 &= \left(\frac{m_1 - m_2}{m_1 + m_2}\right) u_1 + \left(\frac{2m_2}{m_1 + m_2}\right) u_2 \\
v_2 &= \left(\frac{m_2 - m_1}{m_1 + m_2}\right) u_2 + \left(\frac{2m_1}{m_1 + m_2}\right) u_1
\end{align*}
\]
Must be careful

• Problems with round-off error in floating-point arithmetic.
  • Careful with divides

• Especially when have objects of very different masses
  • (can you see why this might be problematic?)
Avoiding Physics in Collisions

- For simple collisions, don’t do the math
  - Two identical balls swap velocities
- For collisions between dissimilar objects
  - Create a collision matrix
Circles and Spheres 4

- Non-head on collision – but still no friction
- Velocity change:
  - Maintain conservation of momentum
  - Change of velocity orthogonal to the collision normal
  - Rewrite velocity via components along and normal to the plane of collision, each retains velocity along the plane. Use previous calc for normal component.
Internet Resources

• Nice discussion of the mathematics of elastic collisions at http://www.mcasco.com/p1lmc.html
Physics Engines

- Havok
- **Strengths**
  - Do all of the physics for you as a package
- **Weaknesses**
  - Can be slow when there are many objects
  - Have trouble with small vs. big object interactions
  - Have trouble with boundary cases
Particle System Explosions

- Start with lots of point objects (1-4 pixels)
- Initialize with random velocities based on velocity of object exploding
- Apply gravity
- Transform color intensity as a function of time
- Destroy objects when collide or after fixed time

- Can add vapor trail (different color, life, wind)
PseudoCode Particle Animation

- *Modified from devmaster.net*

Set up particle

While Animation In Progress
  If Particle Not Dead Then
    Particle Position += Particle Direction * Speed
    Particle Speed += Particle Acceleration
    Modify Particle’s Speed
    Modify Particle’s Energy
  End If

  If Particle’s Energy < Threshold Then
    Mark Particle As Dead
  End If

  If Particle Hits Object Then
    Modify Particle’s Position, Direction, Speed and Energy
  End If

  Display Particle
End If
End While
Example Particle System

- See http://www.jhlabs.com/java/particles.html
- Can be used for explosions, sparkler trails, exhaust
Advanced Physics

- Modeling liquid
- Movement of clothing
- Movement of hair
- Fire/Explosion effects
- Reverse Kinematics