Basic Game Physics

John E. Laird and Sugih Jamin

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, sports games, flight simulation, etc.
    • Running and jumping off the edge of a cliff
• Two types of physics:
  • Elastic, rigid-body physics, $F = ma$, e.g., pong
  • Non-elastic, physics with deformation: clothes, pony tails, a whip, chain, hair, volcanoes, liquid, boomerang
• Elastic physics is easier to get right
Game Physics

• Approximate real-world physics
• We don’t want just the equations
• We want efficient ways to compute physical values
  • Assume fixed discrete simulation – constant time step
  • Must account for actual time passed for variable simulation

• Assumptions:
  • 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
  • Constant time step
Position and Velocity

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

• Position at time $t$ for an object moving at velocity $v$, from starting position $x_0$:
  • $x(t) = x_0 + v_x t$
  • $y(t) = y_0 + v_y t$

• Incremental computation per frame, assuming constant time step and no acceleration:
  • $v_x$ and $v_y$ constants, pre-compute
  • $x += v_x$, $y += v_y$
Acceleration

- Acceleration \((a)\): change in velocity per unit time

Approximate
Acceleration

• Constant acceleration: \( v_x += a_x, \ v_y += a_y \)

• Variable acceleration:
  • use table lookup based on other factors:
  • \( acceleration = acceleration\_value(\text{gear, speed, pedal\_pressure}) \)
    • Cheat a bit: \( acceleration = acceleration\_value(\text{gear, speed}) \times pedal\_pressure \)
  • \( a_x = cos (\nu) * acceleration \)
  • \( a_y = sin (\nu) * acceleration \)

• Piece-wise linear approximation to continuous functions
Gravity

- Gravity is a force between two objects:
  - Force $F = G \frac{m_1 m_2}{D^2}$
    - $G = 6.67 \times 10^{-11} \text{ Nm}^2\text{kg}^{-2}$
    - $m_i$: the mass of the two objects
    - $D$: 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 constant
  - Assume uniform acceleration
  - Position of falling object at time $t$:
    - $x(t) = x_0$
    - $y(t) = y_0 + 1/2 \times 9.8 \text{ m/s}^2 \times t^2$
    - Incrementally, $y +=$ gravity (normalized to frame rate)
Space Game Physics

- Gravity
  - Influences both bodies
  - Can have two bodies orbit each other
  - Only significant for large mass objects
  - Consider $N$-body problem

- 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?
Mass

- Objects represented by their *center of mass*, not accurate for all physical effects

- Center of mass \((x_c, y_c)\) for a polygon with \(n\) vertices:
  - Attach a mass to each vertex
  - \(x_c = \frac{\sum x_i m_i}{\sum m_i}, \quad i = 0 \ldots n\)
  - \(y_c = \frac{\sum y_i m_i}{\sum m_i}, \quad i = 0 \ldots n\)

- For sprites, put center of mass where pixels are densest

- For arcade games, model gravity in sprite frames:

![Sprite frames]
Friction

• Conversion of kinetic energy into heat

Frictional force $F_{\text{friction}} = m\ g\ \mu$
  • $m =$ mass, $g = 9.8 \text{ m/s}^2$,
  • $\mu =$ frictional coefficient = amount of force to maintain a constant speed

$F_{\text{actual}} = F_{\text{push}} - F_{\text{friction}}$
  • Careful that friction doesn’t cause your object to move backward!
  • Consider inclined plane

Usually two frictional forces
  • Static friction when at rest (velocity = 0). No movement unless overcome.
  • Kinetic friction when moving ($\mu_k < \mu_s$)
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**
  - $W$: wind
  - $g$: gravity
  - $W_r$: wind resistance
  - $v_i$: initial velocity
  - $m$: mass of projectile
  - $\theta$: angle of inclination

\[
v_{ix} = v_i \cos(\theta)
\]
\[
v_{iy} = v_i \sin(\theta)
\]

Reaches apex at $t = \frac{v_i \sin(\theta)}{g}$, hits ground at $x = \frac{v_{ix} \times v_{iy}}{g}$

With wind:
\[
x += v_{ix} + W
\]
\[
y += v_{iy}
\]

With wind resistance and gravity:
\[
v_{ix} += W_{rx}
\]
\[
v_{iy} += W_{ry} + g, \text{ g normalized}
\]
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 upon collision or after fixed time

- Can add vapor trail (different color, lifetime, wind)
Advanced Physics

- Modeling liquid (*Shrek, Finding Nemo*)
- Movement of clothing
- Movement of hair (*Monster Inc.*)
- Fire/Explosion effects
- Reverse Kinematics
Physics Engines

• Havok, AGEIA PhysX, Tokamak, etc.

• Strengths
  • Do all of the physics for you as a package

• Weaknesses
  • Can be slow when there are many objects (use PPU?)
  • May have trouble with small vs. big object interactions
  • Have trouble with boundary cases

Source: AGEIA
Back to Collisions

- Steps of analysis for 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 has occurred
• Determine the time of the collision
  • So can back up to point of collision
• Determine where the objects were at time of collision
• Determine the collision angle off the collision normal
• Determine the velocity vectors after collision
• Determine changes in rotation
Circles and Lines

• 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 a bigger wall
  • If vertical and horizontal walls, simple test of x, y
Circles and Angled Lines

- What if more complex background: pinball?
  - For complex surfaces, pre-compute and fill an array with collision points (and surface normals)
Circle on Wall Collision Response

- Determine the time of collision ($t_c$):
  - $t_c = t_i + (x_h-x_1)/(x_2-x_1) \cdot \Delta t$
  - $t_i =$ initial time
  - $\Delta t =$ time increment

- Determine where the objects are when they touch
  - $y_c = y_1 - (y_1-y_2) \cdot (t_c-t_i)/\Delta t$

- Determine the collision angle against collision normal
  - Collision normal is the surface normal of the wall in this case
  - Compute angle of line using $(x_1-x_h)$ and $(y_1-y_c)$
Circle on Wall Collision Response

- Determine the velocity vectors after collision
  - Angle of reflectant = angle of incidence; reflect object at an angle equal and opposite off the surface normal
  - If surface is co-linear with the $x$- or $y$-axes:
    - Vertical - change sign of $x$ velocity
    - Horizontal - change sign of $y$ velocity
    - Corner - change sign of both

- Compute new position
  - Use $\Delta t - t_c$ to calculate new position from collision point

- Determine changes in rotation
  - None!

- Is this worth it? Depends on speed of simulation, …
Circle-circle Collision

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

- Assume elastic collisions:
  - Conservation of momentum
  - Conservation of kinetic energy

- Non-elastic collision converts kinetic energy into heat and/or mechanical deformations
Detect that a collision occurred

- If the distance between two circles is less than the sum of their radii
  - Trick: avoid square root in computing distance!
  - Instead of checking \((r_1 + r_2) > D\), where \(D = \sqrt{(x_1-x_2)^2 + (y_1-y_2)^2}\)
  - Check \((r_1 + r_2)^2 > ((x_1-x_2)^2 + (y_1-y_2)^2)\)

- Unfortunately, this is still \(O(N^2)\) comparisons, \(N\) number of objects
Detect that a collision occurred

- With non-circles, gets more complex and more expensive for each pair-wise comparison
- Use bounding circles/spheres and check for overlap
  - Pretty cheap
  - Not great for thin objects
Avoiding Collision Detection

• General approach:
  • Observations: collisions are rare
    • Most of the time, objects are not colliding
  • Use various filters to remove as many objects as possible from the comparison set
Area of Interest

• 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
  • Linear in number of objects
Detect that a collision occurred

• Alternative if many different sizes
  • Cell size can be arbitrary
  • E.g., twice size of average object

• Test objects in cells touched by object
  • Must determine all the cells the object touches
  • Works for non-circles also
Circle-circle Collision Response

- 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 through the colliding intersection
Circle-circle Collision Response

• Determine the velocity: assume elastic, no friction, head on collision

• Conservation of Momentum (mass * velocity):
  • \( m_1v_1 + m_2v_2 = m_1v'_1 + m_2v'_2 \)

• Conservation of Energy (Kinetic Energy):
  • \( m_1v_1^2 + m_2v_2^2 = m_1v'_1^2 + m_2v'_2^2 \)

• Final Velocities
  • \( v'_1 = \frac{(2m_2v_2 + v_1(m_1-m_2))}{(m_1+m_2)} \)
  • \( v'_2 = \frac{(2m_1v_1 + v_2(m_1-m_2))}{(m_1+m_2)} \)
    • What if equal mass, \( m_1 = m_2 \)
    • What if \( m_2 \) is infinite mass?
Circle-circle Collision Response

For non-head on collision, but still no friction:

- **Velocity change:**
  - Maintain conservation of momentum
  - Change of velocity reflect against the collision normal

```
collision “surface”
```
Must be careful

- Round-off error in floating point arithmetic can throw off computation
  - Careful with divides
- Especially with objects of very different masses
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