Physically-based modelling

Ruth Aylett
Overview

- What is it and why do it?
- Obstacle detection and avoidance
- Moving and falling
  - A bit about vectors
- Particle systems
- Solid objects
What it is

- Building in physically accurate behaviour
- For example:
  - Solid objects cannot be walked through
  - Unsupported objects fall
    - In a physically accurate way under gravity
  - Interacting objects bounce or rebound
  - Soft or bendy objects flex and deform
  - Fragile objects break
  - Liquids flow
- Physically accurate models of light and sound
Why use PBMs?

**Autonomy**

- objects should act and react in a virtual world as they would in the real world
  - if you drop a rubber ball it should fall and bounce
  - if you drop a glass it should fall and smash
  - if you drop a jelly it should fall and wobble

- this automation is useful in animation, but it is *vital* in virtual environments
Why use PBMs?

- **Interaction**
  - for haptic devices physical forces are *required* for input and output
  - Virtual assembly would be odd without it
  - a good PBM can make interaction feel more real
    - may lead to more realistic interaction between the user and the environment
    - e.g. vehicle physics in simulators and games
Why use PBMs?

■ Presence
  – better interaction and autonomy are more likely to achieve a suspension of disbelief
  – increased visual realism
    • erosion and weathering
    • physically based rendering of paints & pigments
Why use PBMs?

- Get answers through experimentation
  - interactive
  - safe
  - accurate (potentially) enough to take measurements
- examples
  - virtual wind tunnel
  - civil engineering structural models
Problems with PBM’s

- VEs must run in real time
- Some physics models are extremely computationally intensive, e.g. fluid dynamics
  - SO: get a faster computer
  - OR: reduce the accuracy of the simulation (not always acceptable)
- Like photo-realism, it has a downside
  - ‘Nearly right’ may be worse if it raises expectations
  - ‘Real’ physics may complicate interaction
Collisions

- A requirement for use of PBMs
- Three stages:
  - Detecting collision
    - Not physical modelling but graphics algorithms
  - Determining exact location of collision
  - Generating response to collision
    - Where using a physical model helps a great deal
- Detection required for haptic interaction
  - Can be thought of as haptic clipping
- Or agents that grasp objects or each other
Exact collision detection

- Which polygons intersect?
  - Exactly where do they intersect?
- Make a pairwise piercing test between all polygons in the scene?
  - Computationally infeasible
- Would like to exclude polygons that cannot intersect
Approximate collision detection

- Uses a bounding box
  - A simple geometrical object roughly representing the volume of a real object’s geometry.
  - As small as possible while still enclosing all vertices of the object.
Bounding volumes - 1

- Trade-off between speed of application and accuracy of boundary

- Various choices:
  - Spheres
    - Very quick to set up
    - Especially simple intersection test depending only on radius $r$ and centre $v$ (a vector $x,y,z$):
      - Non-overlapping if:
        - $(v_1 - v_2) \cdot (v_1 - v_2) > (r_1 + r_2)^2$
    - Not especially accurate
Bounding volumes - 2

– Axis-aligning bounding boxes (AABB)
  • Aligned to axes of world: quite easy to set up
  • As objects rotate may become pretty inaccurate

– Object-oriented bounding boxes (OBB)
  • Aligned to main axes of object and move with it
  • Herder to set up since must translate into object coordinate space
  • But more accurate as a result
Detecting collision

- Project bounding box along axes of world coordinate system
  - If all three projection intervals overlap there is a collision

- Scene coherence
  - Objects do not move much between frames
  - All fixed size bounding boxes projected onto world coordinate system at start
  - Sort interval list and create overlap pairs
  - Traverse sorted lists at each frame
  - Collision detection is $O(n)$ for $n$ objects
Bounding volume hierarchy

- Combine approximate and exact tests
- Extend the bounding volume idea:
  - Associate each node of a tree with subset of primitives
  - With a bounding volume for this subset
  - Use BVs as simplified surface representation for collision test
Line-based obstacle detection

- Useful for autonomous agents
  - Carry out their own obstacle avoidance
- Project line (s) and test for intersection with polygons
  - Similar to infra-red sensors on robots
- Uses higher-level knowledge about objects in scene
Vectors

- Many quantities are described by one value (a *scalar*)
- Vectors combine both magnitude *and* direction
  - in 3D, vectors have x, y & z components
  - can also have *n*-dimensional vectors with *n* components
Vector & scalar quantities

- Understanding the use of vectors and scalars is vital for creating PBM

- Vector quantities
  - velocity
  - acceleration
  - force
  - position (a vector which starts at the origin)

- Scalar quantities
  - mass
  - length
  - area
  - time
Vector notation

To differentiate between scalars and vectors we normally underline vector variables:

area \( a \) (scalar)
velocity \( v \) (vector)
Direction vectors

- To calculate the vector that goes from \( p_1 \) to \( p_2 \) (the direction vector)
  - simply subtract \( p_1 \) from \( p_2 \)
  - the result is a vector, \( \mathbf{v} \)

\[
\mathbf{v} = p_2 - p_1
\]

\[
\begin{align*}
\mathbf{v}_x &= p_{2x} - p_{1x} \\
\mathbf{v}_y &= p_{2y} - p_{1y} \\
\mathbf{v}_z &= p_{2z} - p_{1z}
\end{align*}
\]
Direction vectors

Example:

\[ p_1 = \{1, 5, 4\} \]
\[ p_2 = \{7, 6, 3\} \]

\[ v = \{7-1, 6-5, 3-4\} \]
\[ = \{6, 1, -1\} \]
Direction vectors

- If we wish to represent any point along the vector between $p_1$ and $p_2$
  - start at the start point ($p_1$)
  - add some fraction ($\lambda$) of $v$

\[
\begin{align*}
x &= p_{1x} + \lambda v_x \\
y &= p_{1y} + \lambda v_y \\
z &= p_{1z} + \lambda v_z
\end{align*}
\]
Adding vectors

By adding 2 vectors together, we can get a direction vector from the start of $v_1$ to the end of $v_2$

\[ v_{3x} = v_{1x} + v_{2x} \]
\[ v_{3y} = v_{1y} + v_{2y} \]
\[ v_{3z} = v_{1z} + v_{2z} \]
Scaling vectors

- Multiplying a vector, \( \mathbf{v} \), by a scalar, \( s \)
- The result is a vector with the same direction, but the length is scaled

\[
\begin{align*}
\mathbf{v}_{2x} &= s \mathbf{v}_{1x} \\
\mathbf{v}_{2y} &= s \mathbf{v}_{1y} \\
\mathbf{v}_{2z} &= s \mathbf{v}_{1z}
\end{align*}
\]
Modulus (length) of a vector

The modulus, or length, of a vector can be calculated using Pythagoras’ theorem:

\[ \text{length} = \sqrt{x^2 + y^2 + z^2} \]

Example:

\[ \|v\| = \sqrt{2^2 + 5^2 + 9^2} \]
\[ = \sqrt{4 + 25 + 81} \]
\[ = \sqrt{110} = 10.488 \]
Normalised (unit) vectors

- Sometimes we wish to have only the direction of a vector for use in calculations.
- To normalise (calculate a unit vector) we simply divide a vector by its length, giving the same direction, but a length of 1.0.

Example:

\[ \mathbf{v} = \{2, 5, 9\} \]
\[ \mathbf{v}_{n} = \{2/10.488, 5/10.488, 9/10.488\} \]
\[ = \{0.190694, 0.476735, 0.858124\} \]
We can also represent a unit vector as cosines of the angles made between the vector and each positive axis.

\[ \mathbf{v}_n = \{0.190694, 0.476735, 0.858124\} = (\cos \alpha, \cos \beta, \cos \gamma) \]

\[ \alpha = \cos^{-1} 0.190694 = 79.00^\circ \]
\[ \beta = \cos^{-1} 0.476735 = 61.53^\circ \]
\[ \gamma = \cos^{-1} 0.858124 = 30.89^\circ \]
Dot (scalar) products

- The dot product of 2 vectors can be calculated, producing a scalar

\[ \mathbf{u} \cdot \mathbf{v} = u_x v_x + u_y v_y + u_z v_z \]

- If \( \mathbf{u} \) and \( \mathbf{v} \) are perpendicular, \( \mathbf{u} \cdot \mathbf{v} = 0 \)
- If \( \mathbf{u} \) and \( \mathbf{v} \) are parallel, \( \mathbf{u} \cdot \mathbf{v} = 1 \)
Angle between 2 vectors

- To calculate the angle, $\theta$, between 2 vectors, we use the Cosine Rule

- Using the cosine in the plane of the triangle

$$|c|^2 = |a|^2 + |b|^2 - 2 |a| |b| \cos \theta$$
Cross (vector) products

- Given 2 vectors \( \mathbf{a} \) and \( \mathbf{b} \), we can calculate a vector, \( \mathbf{c} \), that is perpendicular to \( \mathbf{a} \) and \( \mathbf{b} \)

\[
\mathbf{c} = \mathbf{a} \times \mathbf{b} = \left( a_y b_z - a_z b_y, a_z b_x - a_x b_z, a_x b_y - a_y b_x \right)
\]
Solid Objects

- Rigid bodies
  - particles
  - extended masses
- Collision response
- Deformable objects
Newton’s Laws of Motion

- **Newton’s first law**
  - if the forces on an object are balanced then its acceleration will be zero
  - if the forces are unbalanced then the object will accelerate, changing its velocity

- **Newton’s second law**
  - if the forces are unbalanced, the resulting acceleration depends on:
    - the object’s mass
    - the forces applied to the object
Newton’s Laws of Motion

– The second law is expressed by the equation

\[ f = ma \]

■ Newton’s third law
– For every action there is an equal and opposite reaction
Mechanics

- If acceleration, $a$, is constant over a time, $t$, we can derive some equations that govern motion.
- If the time, $t$, is short enough we can assume the acceleration to be constant.
Mechanics

Acceleration

\[ a = \frac{v - u}{t} \]

or

\[ v = u + at \]

where:

- \( u \) is the initial velocity
- \( v \) is the final velocity
Mechanics

Displacement, $s$

$$s = t \times \frac{u+v}{2}$$

giving:

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

and:

$$v^2 = u^2 + 2as$$
Mechanics

- If the only force that is acting is gravity (an acceleration of 9.81 ms\(^{-2}\))

\[ s = \frac{1}{2} gt^2 \]
Moving

- Implemented graphically as interpolation between frames
  - However linear interpolation (e.g. VRML) clearly **not** physically correct in most cases
  - Acceleration produces non-linear motion
  - Use Newtonian mechanics to calculate displacement under acceleration
The VRML Bouncing Ball

DEF Ball Transform {
  children [
    Shape {
      appearance Appearance {
        material Material {
          ambientIntensity 0.5
          diffuseColor 1.0 1.0 1.0
          specularColor 0.7 0.7 0.7
          shininess 0.4
        }
        texture ImageTexture { url "beach.jpg" }
        textureTransform TextureTransform { scale 2.0 1.0 }
      }
      geometry Sphere {
      } ]
}
DEF Clock TimeSensor {
    cycleInterval 2.0
    startTime 1.0
    stopTime 0.0
    loop TRUE}

ROUTE Clock.fraction_changed TO Bouncer.set_fraction
ROUTE Bouncer.value_changed TO Ball.set_translation
Using a script

```plaintext
DEF Bouncer Script {
    field   SFFloat bounceHeight 3.0
    eventIn SFFloat set_fraction
    eventOut SFVec3f value_changed

    url "javascript:
    function set_fraction( frac, tm ) {
        y = 4.0 * bounceHeight * frac * (1.0 - frac);
        value_changed[0] = 0.0;
        value_changed[1] = y;
        value_changed[2] = 0.0;
    }
}"
```
Calculating the equation

Use Newton’s equation:

\[ y(t) = ut + \frac{1}{2} gt^2 \]

\[ y(1) = 0 = u*1 + \frac{1}{2} g*1*1 \] assuming \( y=0 \) again when \( t=1 \) and ball is again on the ground.

So \( u = -\frac{1}{2} g \)

Halfway through, at top of the bounce, \( y=h \) (some chosen bounce height)

\[ y(0.5) = u*0.5 + \frac{1}{2} g*0.5*0.5 = h \]

So \( -\frac{1}{4} g + \frac{1}{8} g = h \) and \( g = -8h \)

Putting into 1st eq, \( y(t) = 4ht(1-t) \)

\( h = \text{bounceHeight} \), \( t = \frac{\text{frac}}{\text{frac}} \)
Modelling a particle

- With Newton’s equations of motion can simulate motion of a particle
- Particles are an ‘ideal’ concept
  - They do not account for a distribution of mass
    - it is all focused at one point
  - The simulation will lack some accuracy
  - These equations do not account for aspects such as friction and spin
- So was the bouncing ball a particle?
Modelling a particle

- To model the motion of a particle with these equations, we use the following process for each frame of the simulation:

  - Calculate forces acting on the particle
  - Calculate the acceleration of the particle
  - Calculate the new velocity of the particle (include drag here)
  - Calculate the new position of the particle
Extended bodies

- When a force is applied to an object with extent (i.e. not a particle) the object’s movement will consist of
  - translation
  - rotation
Extended bodies

- We consider the induced motion as two separate components
  - A force acting on the centre of mass of the object, inducing motion as if it were a particle
  - A torque (rotational force) acting about a particular axis
Extended bodies

■ In order to calculate the rotational components, we need to understand the distribution of the mass of the object with respect to its centre of mass.

■ We measure this distribution with moments of inertia:
  – For symmetrical bodies we require 3 moments of inertia, one about each axis.
  – For asymmetrical bodies we require an inertial tensor consisting of 9 components.
Extended bodies

Say we consider a rigid body to consist of an infinite number of particles

– Then a moment of inertia is just the sum of the mass of each particle weighted by the square of the distance to each axis:

\[ I_x = \int (y^2 + z^2) \, dm \]
\[ I_y = \int (x^2 + z^2) \, dm \]
\[ I_z = \int (x^2 + y^2) \, dm \]
Extended bodies

- **Symmetrical bodies**
  - A sphere, with radius $r$ and mass $m$
    \[ I_x = I_y = I_z = \frac{2}{5} m r^2 \]
  - A cylinder, with radius $r$ and height $h$, where the $z$ axis is along the long axis of the cylinder
    \[ I_x = I_y = \frac{1}{4} m \left( r^2 + \frac{1}{3} h^2 \right) \]
    \[ I_z = \frac{1}{2} m r^2 \]
Extended bodies

– A box, with sides a, b and c along the x, y and z axes respectively

\[
I_x = \frac{1}{12} \text{m} (a^2 + b^2)
\]

\[
I_y = \frac{1}{12} \text{m} (a^2 + c^2)
\]

\[
I_z = \frac{1}{12} \text{m} (b^2 + c^2)
\]

- The box formulation is very useful as you can often approximate an object to its bounding box, providing a simplified (but less accurate) solution
Extended bodies

- For asymmetrical objects we must also calculate the products of inertia
  - Then assemble the inertial tensor (a 3 x 3 matrix) that describes the asymmetric distribution of mass
Extended bodies

- The translational components of motion are defined with the same equations for particle motion.
- The rotational component relates:
  - angular acceleration, mass, torque.