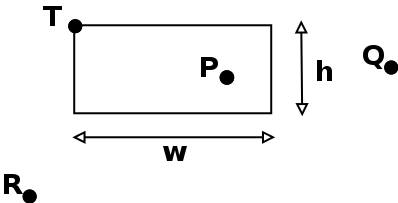
## Math for COMP258

**Collision Detection**

*Does a point lie inside a rectangle with top left corner point* T*, width* w *and height* h*?*



boolean ptInRect(PVector point,PVector T,float w,float h) {

boolean result = false;

if (point.x >= T.x && point.x <= T.x + w)

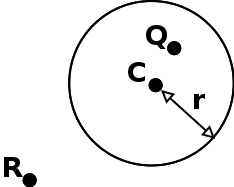
if (point.y >= T.y && point.y <= T.y + h)

result=true;

return result;

}

*Does a point lie inside a circle with center* C*, and radius* r*?*



boolean ptInCircle(PVector point,PVector C,float r) {

boolean result = false;

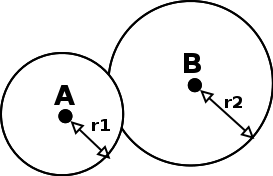
if (point.dist(C)<=r)

result=true;

return result;

}

*Do two circles with centers* A *and* B*, and radii* r1 *and* r2*, intersect?*



boolean circleCollision(PVector A, PVector B,float r1, float r2) {

boolean result = false;

if (A.dist(B)<=r1+r2)

result=true;

return result;

}

**Points around the Perimeter of a Circle**

See <http://www.mathopenref.com/coordparamcircle.html> for a nice, interactive demonstration

PVector perimeterPt(PVector center, float radius, float angle) {

PVector result = new PVector();

result.x = center.x + radius \* cos(angle);

result.y = center.y + radius \* sin(angle);

return result;

}

Note that in Processing the default measurement for angles is **radians** which range from 0 to TWO\_PI. **Degrees**, on the other hand, range from 0 to 360. Processing provides a useful function called **radians()** to convert degrees to radians.

For example, to calculate the point at 60 degrees around a circle centered on (25,30) with radius 10:

PVector center = new PVector(25,30);

PVector pt = perimeterPt(center, 10, radians(60));

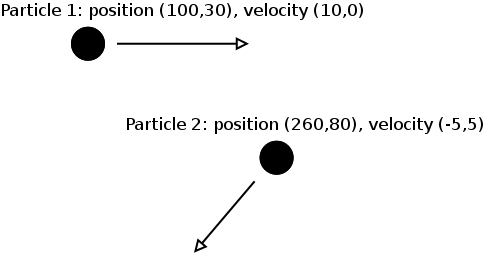
**Basic Physics**

The position of an object in a 2D world can be represented by a **position vector (x,y)**. In most simple physics simulations, objects can also move, and this is represented by a **velocity vector (vx, vy)**.

Velocity is defined as the object’s *rate of change of position with time*. It is also called speed.

An object that is not moving would have velocity (0,0). An object that is moving would have non-zero values for its velocity vector.

For example:



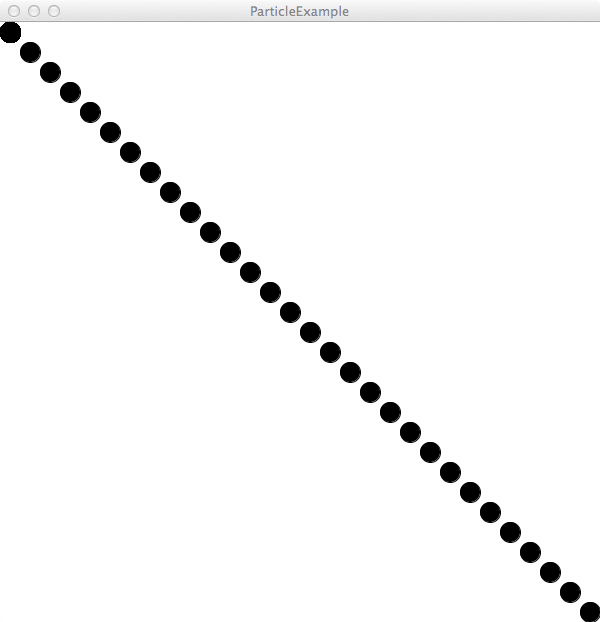
In Processing, the simplest way to apply physics is to assume that time is measured in frames. Every frame, update an object’s position by adding its velocity (measured in pixels per frame) to its current position, i.e.:

public void move() {

position.add( velocity ); }

}

Over many frames, an object with a constant velocity will travel in a straight line:



Acceleration can also be represented as a 2D **acceleration vector (ax,ay)**. Acceleration is defined as the *rate of change of velocity with time*.

In order to update an object’s position with its acceleration, simply add it to the the acceleration:

public void move() {

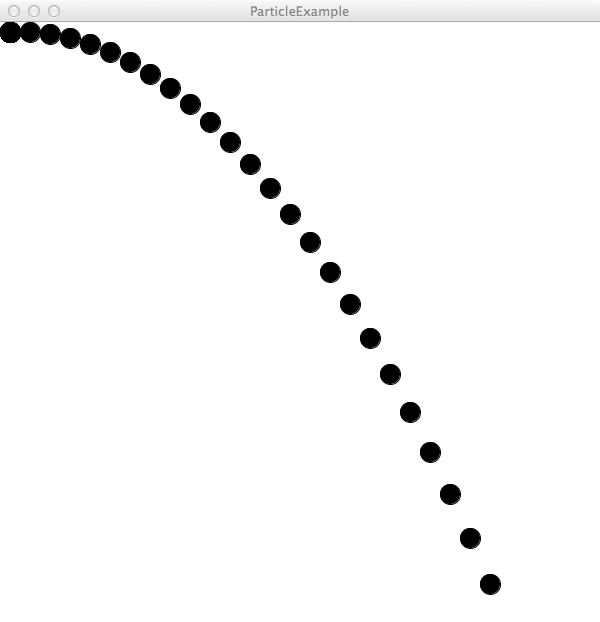
position.add( velocity );

velocity.add( acceleration );

}

If the acceleration vector is zero but the velocity vector is non-zero, then the object will move in a straight line.

If the acceleration vector is non-zero, then the object moves in a curve. For example, the following object has initial position (10,10), initial velocity (20,0), and acceleration (0,2):



Simulating a “bounce”:

A bounce of an object against a rigid fixed object such as a wall can be simulated by inverting the direction of the velocity.

That is, if the object is moving down (positive vy) and it hits the ground, its velocity must change to up (negative vy). If the object is moving left (negative vx) and it hits a wall, its velocity by change to moving right (positive vx).

Here’s some code for simulating a bound from the ground:

public void move() {

position.add( velocity );

velocity.add( acceleration );

if (position.y>=groundLevel)

if (velocity.y>0) {

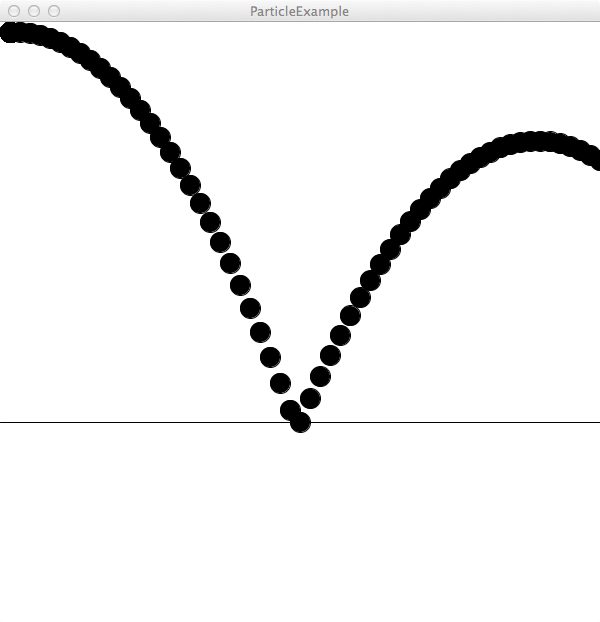
position.y = groundLevel;

velocity.y \*= -0.8 ;

}

}

Result:



Note that the acceleration doesn’t change here. Instead, when the object bounces, the sign of velocity.y is flipped. The velocity is also reduced in size to 80% of its previous value, to simulate loss of energy due to the collision.

Finally, the object’s position position.y is checked to make sure that the object doesn’t go through the ground. If the current position is below ground, the position is updated to ground level.

**Gaussian vs Uniform Random Numbers**