**Laboratory 5: Modelling motion**

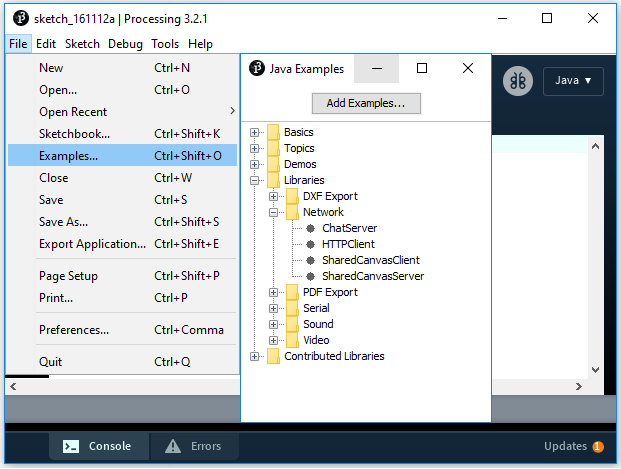
**Introduction:** The aim of this section of the course is to look to the importance of calculus to the software engineer. However, the general area has provided an opportunity to look at wider concepts that will be of use to you in the future. For example, we have already looked at reading data out of a file and plotting graphs.

We will return to differential equations shortly, however Processing provides an effective library for Network communications. Now is a good time to look at these libraries and connect your code to the rest of the world by the Network.

**Part 1:** Network communication

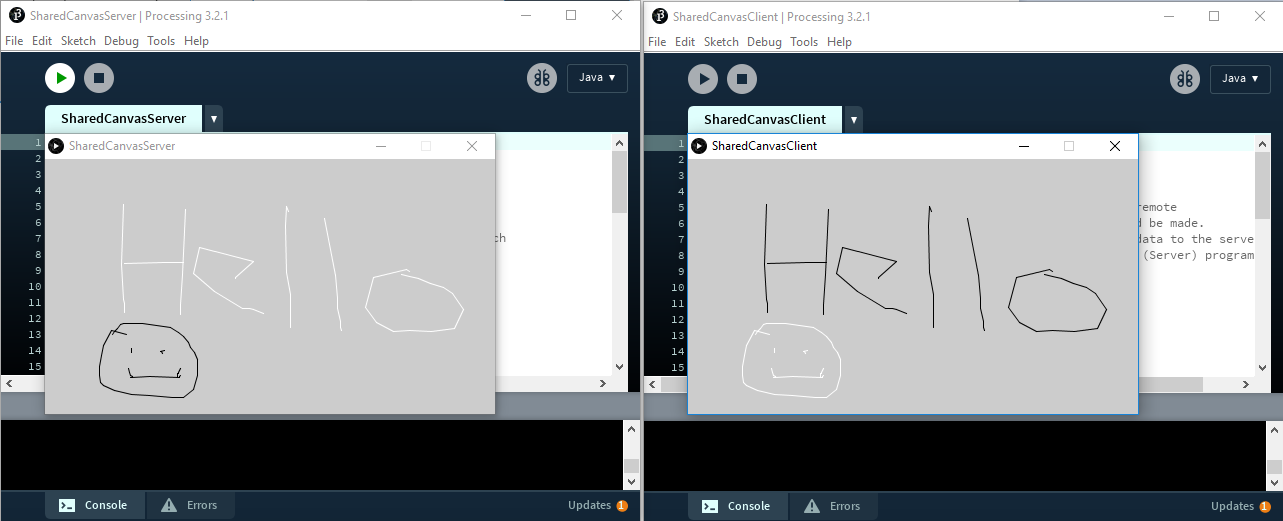
An IP (Internet Protocol) address is a 32 bit number assigned to a device connected to a Network. The number is a unique identifier for the device on the network. The 32 bit number is normally written as 4 bytes with values on then interval [0,255] written in the following format,

Open the examples in Processing and open the two files *SharedCanvasClient* and *Shared CanvasServer*. Run both of these programs on the same machine.



**Figure 1:** Loading an example from a library that has been added to Processing.

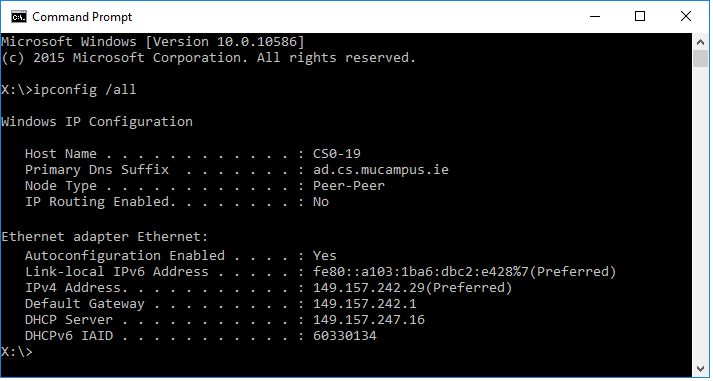
**To Do:** Run the Server program first then the Client program. The client will connect to the server via a local IP address 127,0,0,0.



**Figure 1:** Shared canvas program running on localhost.

This “local host” allows us to test our network ideas on one machine without requiring the Network infrastructure. Many devices (such as eye trackers, IP webcams etc) use Network communication to send data to an application using this method. With recent growth in the “internet of things” it is an area of considerable commercial and academic interest.

To share a canvas with a friend you will need to find your IP address. To do this launch the command prompt (black rectangle on desktop or search for “cmd”) and enter the command *ipconfig /all* look down the list of details for the IPv4 address and note it.

****

**Figure 2:** Identifying the IP address using ipconfig /all

**To Do:**

**<this may be a challenge to set up remotely so skip the room message posting section>**

Edit the client program so that it refers to the server machine you wish to communicate with rather than the local host IP address 127,0,0,1.

// Connect to the server&apos;s IP address and port

c = new Client(this, "149.157.242.29", 12345); // Replace with your server&apos;s IP and port

Re-run the server program and client program and see if you can establish two-way communication with another person in the lab. Well done...

We can now look more closely at these programs to see how they work; I have done this and reduced each of the programs to a minimum form that should aid understanding. This should allow you to send text messages to each other (or to the front of the room). Do not be tempted to send anything you would not wish to see publicly displayed.

Call this Project *Room\_Server.pde*,

import processing.net.\*;

Server s;

Client c;

String input;

int data[];

int line=0;

void **setup**()

{

  size(600, 500);

  background(204);

  textSize(48);

  fill(0);

  s = new Server(this, 12345); // Start a simple server on a port 12345

}

void **draw**()

{

  // Receive data from client

  c = s.available();

  if (c != null)

  {

    input = c.readString();

    input = input.substring(0, input.indexOf("\n")); // Only up to the newline

    text(input,10,48+48\*(line));

    line++;

    if(line==10) {line=0; background(204);}

  }

}

and call the client program, *Room\_Client.pde*; Network communication with about two lines of code!

import javax.swing.JOptionPane;

import processing.net.\*;

Client c;

String input;

int data[];

void **setup**()

{

  size(50, 50);

  background(204);

  c = new Client(this, "149.157.242.29", 12345); // Replace with your IP and port

}

void **draw**()

{

  String r = JOptionPane.showInputDialog(null,"1223", "Text input", JOptionPane.QUESTION\_MESSAGE);

  c.write(r+"\n");

}

You could try creating a bi-directional communication interface, however a good example (if a little more complicated) is available in the Processing Examples.

**<skip to here>**

**To Do:** The following code snippet was found online. Integrate it into your code to provide a means of the user avoiding having to use ipconfig /all to find the IP address. You could have the IP address shown on the top right of the screen at run time.

import java.net.InetAddress;

InetAddress inet;

String myIP;

  try {

    inet = InetAddress.getLocalHost();

    myIP = inet.getHostAddress();

  }

  catch (Exception e) {

    e.printStackTrace();

    myIP = "couldnt get IP";

  }

  println(myIP);

Take a screen shot of your program running and use the HTML Copy and paste of the source code (in Processing) to record this work in a Word document under the heading “Network Communication”.

Finally, to show you what can be done with a little more time try running the following Pong game across the network.

// PongServer.pde

import processing.net.\*;

Server s;

Client c;

String input;

int data[];

int left\_paddle=150;

int right\_paddle=150;

int ballx=40;

int bally=300;

int dx=2;

int dy=2;

int score\_left=0;

int score\_right=0;

int right\_click=0;

int serve=1; // 0:in play 1: left serve 2: right serve

void **setup**()

{

  size(600, 300);

  background(204);

 // frameRate(15); // You may need to slow it down a little

 textSize(40);

  s = new Server(this, 12345); // Start a simple server on a port

}

void **draw**()

{

  background(255);

  // Receive position of right paddle from client

  c = s.available();

  if (c != null)

  {

    input = c.readString();

    int i=input.indexOf("\n");

    if(i>0)

    {

    input = input.substring(0, i); // Only up to the newline

    data = int(split(input, ",")); // Split values into an array

    // Client sends paddle and mouse click state

    if(data.length==2)

    {

    right\_paddle=data[0];

    right\_click=data[1];

    }

    }

  }

  left\_paddle=mouseY;

  fill(255,0,0);

  rect(10,left\_paddle-30,20,60);

  // Send position of left paddle and ball to client

  s.write(left\_paddle + ","+ballx+"," +bally+","+score\_left+"," +score\_right+"\n");

  fill(0,0,255);

  rect(560,right\_paddle-30,20,60);

  // Draw ball

  fill(0,255,0);

  rect(ballx-5,bally-5,10,10);

  if (ballx<0) {dx=-2; ballx=550; serve=2; score\_left++;}

  if (ballx>width) {dx=2; ballx=40; serve=1; score\_right++;}

  if(serve==0) // in play

  {

   // If ball is "out" hits either end

   ballx+=dx;

  // Bouce off top/bottom edge

  bally+=dy;

  if ((bally>height)||(bally<0)) {dy=-dy; bally+=dy;}

  // if balle hits left paddle

  if ((abs(left\_paddle-bally)<30)&&((ballx>30)&&(ballx<=(30+abs(dx))))) {dx=-dx; ballx+=dx;}

  // if balle hits right paddle

  if ((abs(right\_paddle-bally)<30)&&((ballx>560-abs(dx))&&(ballx<=(560)))) {dx=-dx; ballx+=dx;}

  }

  if(serve==1)

  {

   bally=mouseY;

   if (mousePressed==true) serve=0;

  }

  if(serve==2)

  {

   bally=right\_paddle;

   if (right\_click==1) serve=0;

  }

  fill(0);

  text(""+score\_left,40,40);

  text(""+score\_right,530,40);

}

And the client code is as follows...

// PongClient

import processing.net.\*;

Client c;

String input;

int data[];

int right\_paddle=150;

int left\_paddle=150;

int click=0;

int ballx=300;

int bally=150;

int score\_left=0;

int score\_right=0;

void **setup**()

{

  size(600, 300);

  background(204);

  textSize(40);

  //frameRate(15); // You may need to slow it down a little

  // Connect to the server&apos;s IP address and port

  c = new Client(this, "127.0.0.1", 12345); // Replace with your server&apos;s IP and port

}

void **draw**()

{

  background(255);

  // Receive position of left paddle from server

  if (c.available() > 0)

  {

    input = c.readString();

    int i=input.indexOf("\n");

    if(i>0) // Check there is data

    {

    input = input.substring(0, i); // Only up to the newline

    data = int(split(input, ",")); // Split values into an array

    // Update game state sent from server

    if(data.length==5) // A full line should contain 5 pieces of data

    {

    left\_paddle=data[0];

    ballx=data[1];

    bally=data[2];

    score\_left=data[3];

    score\_right=data[4];

    }

    }

  }

  fill(255,0,0);

  rect(10,left\_paddle-30,20,60);

  right\_paddle=mouseY;

  if (mousePressed==true) click=1; else click=0;

  fill(0,0,255);

  rect(560,right\_paddle-30,20,60);

  c.write(right\_paddle +","+click+"\n");

  // Draw ball

  fill(0,255,0);

  rect(ballx-5,bally-5,10,10);

  fill(0);

  text(""+score\_left,40,40);

  text(""+score\_right,530,40);

}

Answer the following questions in your word document under the heading “Pong Game”.

Why did the server code contain no IP address?

Why has code been added to test if the string received is incomplete, how could this fault occur?

**Part 2:** Multiple balls using Object Oriented Programming

At this stage, you should have a working ball bouncing program up and running. If not here is a basic version sufficient to get started.

// Physical Constants

float g=-9.81; // Gravity

float k=0.0; // Friction const (Try 0.02)

float m=0.25; // Mass kg

float dt=0.01; // Time step 10ms

// Boundary (starting) conditions

float vy=0; // Initial vertical velocity

float y=0.9; // Initial vertical position

float x=0; // Initial horizontal position

float vx=0.8; // Initial horizontal velocity

float t=0; // Initial time

void **setup**()

{

  size(500, 500); // but region of interest [0,0],[1,1]

}

void **draw**()

{

  background(255); // White background

  vy=vy+(g-((k/m)\*vy))\*dt;  // Euler Cromer Advance

  y=y+(vy\*dt);

  vx=vx+(-((k/m)\*vx))\*dt;

  x=x+(vx\*dt);

  t=t+dt;

 if (y<=0) {vy=-vy; y=y+(vy\*dt);} // Change direction (and step back)

 if (x<=0) {vx=-vx; x=x+(vx\*dt);}

 if (x>=1) {vx=-vx; x=x+(vx\*dt);}

  float sx=map(x,0,1,0,width); // Scale x on [0,0][1,1] to the screen

  float sy=map(y,0,1,height-1,0);

  fill(255,0,0); // Draw the ball

  ellipse(sx,sy,10,10);

}

Imagine now that you had to make a 100 balls bounce around the screen independently you might at first baulk at the amount of code required to do this. However using an Object Oriented approach, it becomes a relatively straightforward process. We create a description of one ball and then create an array of these balls. The object oriented approach makes use of a “class” to group together all the methods (in our case *move()* and *draw()* ) and variables (in our case x, y, vx and vy) that are associated with a ball.

The following listing reorganises the above code into a class; replace the existing code with this version.

// Physical Constants

float g=-9.81; // Gravity

float k=0.0; // Friction const (Try 0.02)

float m=0.25; // Mass kg

float dt=0.01; // Time step 10ms

void **setup**()

{

Program

  size(500, 500);

}

void **draw**()

{

  background(255);

}

class Ball

{

  float vy;

  float y;

Variables

  float x;

  float vx;

  float t;

  Ball()

  {

Class constructor runs once each time a new instance of the class is created.

   vy=0;

   y=0.9;

   x=0;

   vx=0.8;

Class description, consider this the template for making instances of the object. It is the “cookie cutter”

   t=0;

  }

  void advance()

  {

Method to advance the ball belongs to the class.

    vy=vy+(g-((k/m)\*vy))\*dt;

    y=y+(vy\*dt);

    vx=vx+(-((k/m)\*vx))\*dt;

    x=x+(vx\*dt);

    t=t+dt;

    if (y<=0) {vy=-vy; y=y+(vy\*dt);}

    if (x<=0) {vx=-vx; x=x+(vx\*dt);}

    if (x>=1) {vx=-vx; x=x+(vx\*dt);}

  }

  void draw\_ball()

Method to draw the ball belongs to the class.

  {

      float sx=map(x,0,1,0,width);

      float sy=map(y,0,1,height-1,0);

      fill(255,0,0);

      ellipse(sx,sy,10,10);

  }

}

You will notice that if you run this code it does nothing as the runnable code in *setup()* and *draw()* is empty. To create an instance of the ball class called “b”. Change the code at the start of the program to read as follows (add the four lines that are not grey).

// Physical Constants

float g=-9.81; // Gravity

...

Create a reference to a Ball object called “b”. Similar to declaring a variable without giving it a value (e.g. *int x;*)

Ball b;

void **setup**()

Create an instance of the Ball object. The class constructor, *Ball()*, is called and the variables are set to their starting value.

{

  b=new Ball();

  size(500, 500);

}

void **draw**()

{

  background(255);

Call the methods that now belong to the object to advance its position and then draw it. The method is accessed via the “.” dot operator.

  b.advance();

  b.draw\_ball();

}

**To Do:** Run the code and verify that it works.

**Pause and reflect:** The first program was about 20 lines of code that required understanding of physics, plotting, scaling and collisions with boundaries. The second program (if you ignore the class) is just four lines. Declare the object and initialise it, then access its variables and methods via the dot operator. This “abstraction” of the problem allows us to keep at a distance the fiddly technical stuff when we are assembling a bigger program.

**Note:** You used the Gplot object last week to draw graphs.

Imagine if at this point someone asked you to make the program run for 50 balls. It can be done as follows,

Create a reference to an array (list) of Ball objects.

Ball [] b;

void **setup**()

Create an empty array of Ball objects.

{

  b=new Ball[50];

For each ball object call the class constructor so as to set its initial variable values. At this point 50 instances of the ball object have been created. They can be indexed via the integer value inside the square bracket, the first ball object is b[0] and the last b[49].

  for(int i=0;i<50;i++)

  {

    b[i]=new Ball();

  }

  size(500, 500);

}

void **draw**()

{

  background(255);

  for(int i=0;i<50;i++)

We advance and draw each object separately.

  {

   b[i].advance();

   b[i].draw\_ball();

  }

}

To extend the code to 50 balls required the addition of only three extra lines. Run the program and you will notice that only one ball appears. The problem is that all the balls have the same starting condition and advance identically. Change the class constructor to give each ball a random position and velocity.

Choose random values for the position and velocity. Positions are selected on the interval [0.1,0.9] so the balls don’t start on an edge. Velocities are chosen on the interval [-1,1] to allow starting in all directions.

  Ball()

  {

   y=random(0.1,0.9);

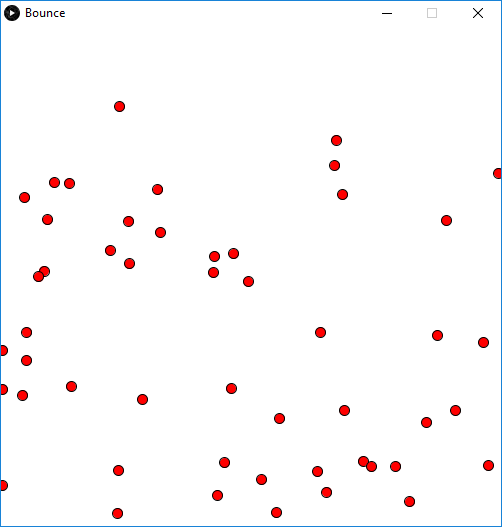
   vy=random(-1,1);

   x=random(0.1,0.9);

   vx=random(-1,1);

   t=0;

  }



**Figure 3:** 50 Balls bouncing around the screen.

We now wish to make the balls different colours and different sizes. To do this you will need to add two new variables to the class *radius* and *col*, 4 lines of code.

class Ball

{

Add variables to define the balls colour and size to the class.

  color c;

  float radius;

  float vy;

  float y;

  float x;

  float vx;

  float t;

  Ball()

  {

   y=random(0.1,0.9);

   vy=random(-1,1);

   x=random(0.1,0.9);

   vx=random(-1,1);

Use random values during construction of object.

   c=color(random(255),random(255),random(255));

   radius=0.01+random(0.04);

   t=0;

  }

In addition we will need to change the *draw\_ball()* method so as to make use of this information. Replace the existing method with the new one.

  void draw\_ball()

  {

     float sx=map(x,0,1,0,width);

     float sy=map(y,0,1,height-1,0);

     float rx=map(radius,0,1,0,width);

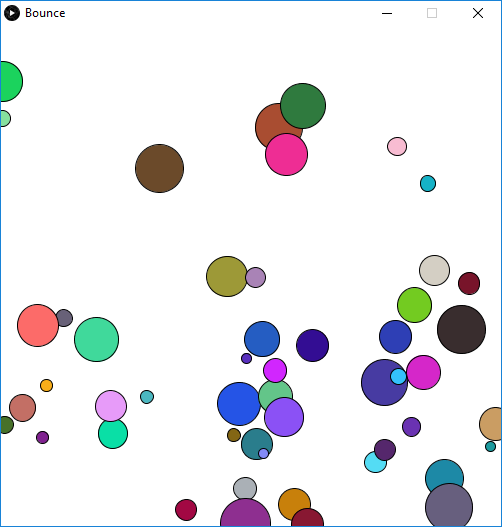
     float ry=map(radius,0,1,0,height);

     fill(c);

     ellipse(sx,sy,2\*rx,2\*ry);

  }

**Note:** the new draw ball method works in terms of one unit of measurement, the physical one of the space the differential equations control, co-ordinates inside [0,0] and [1,1]. We avoid mixing units of measurement such as pixels and meters. Run the program.



**Figure 4:** 50 Balls of different colour and size bouncing around the screen.

At this stage, you may have noticed that things are not realistic because the balls only bounce off the boundaries not each other. To detect and act on a collision is not something you can do within the class (it works on individuals).

To detect a collision you could add the following method before the, *Ball*, class definition but after the *draw(){}* method.

void **draw**()

{

...

Check ball j against all other balls and return -1 if no overlap or the index of the overlapping ball.

}

int Collision(int j,Ball b[])

{

Don’t check if ball overlaps with itself (exclude this)

Cycle through all balls

  int res=-1;

  for(int i=0;i<50;i++)

  {

Euclidian distance between two centre points (balls I and j)

   if (i!=j)

   {

     float sep=sqrt(pow(b[i].x-b[j].x,2)+pow(b[i].y-b[j].y,2));

     float rads=b[i].radius+b[j].radius;

     if(sep<rads){ res=i;}

Sum of the two radii (balls i and j)

   }

  }

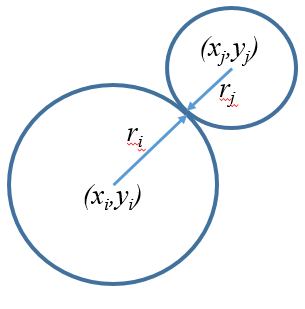
Collision between ball i and ball j detected.

  return(res);

}

class Ball

{



**Figure 5:** If the distance between the centres of the circles is less than sum of circles radii then a collision has occurred.

We can now use the collision method in our main program. The following code turns the colour of any colliding balls black. Add the code between the *advance()* and the *draw\_ball()* methods in *draw().* Run the code. You will notice they go black quite quickly.

   b[i].advance();

   int r=Collision(i,b); // Check if ball i collides with any balls in array b

   if(r!=-1) // -1 means no collision, r is index that identifies ball in b

   {

        b[i].c=color(0,0,0);

        b[r].c=color(0,0,0);

   }

   b[i].draw\_ball();

The modelling of the collision physics between spheres is a little tricky for one week. The following code provides a fair approximation of the velocities of objects before and after a collision. See the notes for more details. Replace the two lines that change the colour of the balls to the ones shown below.

   b[i].advance();

   int r=Collision(i,b); // Check if ball i collides with any balls in array b

   if(r!=-1) // -1 means no collision other

   {

      float nvxj = (b[i].vx \* (b[i].radius - b[r].radius) + (2 \* b[r].radius \* b[r].vx)) / (b[i].radius + b[r].radius);

      float nvyj = (b[i].vy \* (b[i].radius - b[r].radius) + (2 \* b[r].radius \* b[r].vy)) / (b[i].radius + b[r].radius);

      float nvxr = (b[r].vx \* (b[r].radius - b[i].radius) + (2 \* b[i].radius \* b[i].vx)) / (b[i].radius + b[r].radius);

      float nvyr = (b[r].vy \* (b[r].radius - b[i].radius) + (2 \* b[i].radius \* b[i].vy)) / (b[i].radius + b[r].radius);

      b[i].vx=nvxj;

      b[i].vy=nvyj;

      b[r].vx=nvxr;

      b[r].vy=nvyr;

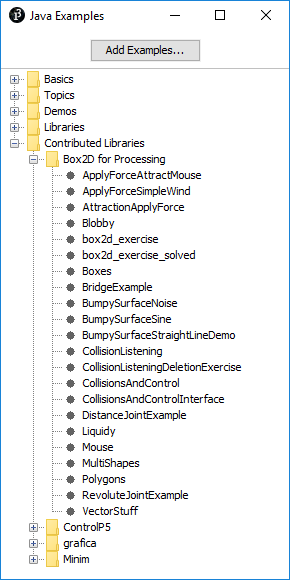
   }

   b[i].draw\_ball();

**To Do:** Run the code. Reduce the number of particle to five and then take a screen shot and add the resulting image to your word document, add a suitable figure title. Well done you have created a reasonable particle simulation using object-oriented design.

**Part 3:** Using Box2D to model physical system

Imagine if at this stage we wanted to improve the physics and add square objects to the scene. We could do it but it would take quite a while to code and even longer to iron out all the bugs. As software engineers we should always look to “code re-use” rather than scratch building code to build our applications. Using properly maintained libraries that have been used and maintained by many people provides a quicker and more reliable result. Box2D is a 2D physics engine that can model some interesting problems similar to those we have just studied. It should be added already to Processing (or you can add it). First, try running the examples that come with it.



**Figure 5:** Examples of Box2D available when it is installed in Processing.

At least look at Boxes, AttractionApplyForce, Liquidy and Polygons.

Box2D is the physics engine behind programs such as “knock down the castle”, “crush the castle” and “angry birds”. The following very short program gives you an idea of what is possible.

In Box2D you start by creating a “Body” definition. This consists of defining the location (x,y), linear velocity (defines the direction the Body is moving in), angular velocity (defines how the Body is spinning) and its type (STATIC: fixed like stone, DYNAMIC: free to move using physics engine, KINEMATIC: moveable like a paddle but without physics). The body is at this stage just a point.

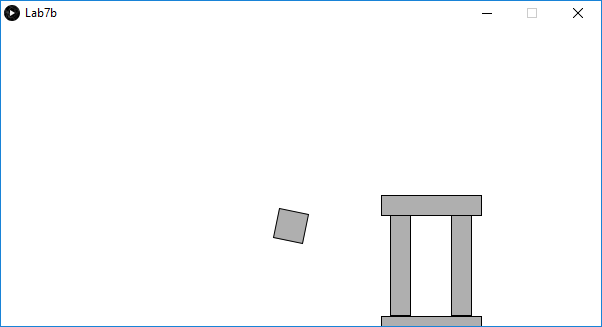
You can then define a geometry (shape) to associate with the Body; typically, this is some form of polygon.

The geometry and the body are brought together using a fixture. The shape is a property of the fixture. The fixture is a property of the body.

In the Box2D examples, the classes used to create the Body are stored in separate pde files contained in the same folder as the main program pde file. This is a good idea as it pushes the “boiler plate” code out of sight to allow you to focus on the use of the Box2D objects.

In the following example, we have moved this code back into the main program pde, which allows us to see it and modify it.

Copy and paste the following code into a new Processing sketch. Compile and run the code.



**Figure 6:** 5 Bodies defined in Box2D, what happens next?

import shiffman.box2d.\*;

import org.jbox2d.collision.shapes.\*;

import org.jbox2d.common.\*;

import org.jbox2d.dynamics.\*;

// A list for all of our rectangles

ArrayList<Box> boxes;

Box2DProcessing box2d;

void **setup**()

{

  size(600,300);

  smooth();

   // Initialize and create the Box2D world

  box2d = new Box2DProcessing(this);

  box2d.createWorld();

  // Turn on collision listening!

  box2d.listenForCollisions();

  // Create ArrayLists

  boxes = new ArrayList<Box>();

  Box p = new Box(400,height-61,20,100,0);

  boxes.add(p);

  p = new Box(460,height-61,20,100,0);

  boxes.add(p);

  p = new Box(430,height-122,100,20,0);

  boxes.add(p);

  p = new Box(430,height-5,100,10,1);

  boxes.add(p);

  p = new Box(40,height-52,30,30,0);

  boxes.add(p);

  p.body.setAngularVelocity(-4);

  p.body.setLinearVelocity(new Vec2(30, 10));

}

void **draw**()

{

  background(255);

  // We must always step through time!

  box2d.step();

  // Display all the boxes

  for (Box b: boxes)

  {

    b.display();

  }

}

// A rectangular box

class Box

{

  // Instead of any of the usual variables,

  // we will store a reference to a Box2D Body

  Body body;

  float w,h;

  Box(float x, float y, float w\_,float h\_, int type)

  {

    w = w\_;

    h = h\_;

    // Build Body

    BodyDef bd = new BodyDef();

    if(type==0)

    {

     bd.type = BodyType.DYNAMIC; // Was DYNAMIC

    }

    else

    {

      bd.type = BodyType.STATIC;

    }

    bd.position.set(box2d.coordPixelsToWorld(x,y));

    body = box2d.createBody(bd);

   // Define a polygon (this is what we use for a rectangle)

    PolygonShape sd = new PolygonShape();

    // Box2D considers the width and height of a

    // rectangle to be the distance from the

    // center to the edge (so half of what we

    // normally think of as width or height.)

    float box2dW = box2d.scalarPixelsToWorld(w/2);

    float box2dH = box2d.scalarPixelsToWorld(h/2);

    sd.setAsBox(box2dW, box2dH);

    // Define a fixture

    FixtureDef fd = new FixtureDef();

    fd.shape = sd;

    // Parameters that affect physics

    fd.density = 1;

    fd.friction = 0.3;

    fd.restitution = 0.5;

    // Attach Fixture to Body

    body.createFixture(fd);

  }

  void display()

  {

    // We need the Body’s location and angle

    Vec2 pos = box2d.getBodyPixelCoord(body);

    float a = body.getAngle();

    // Using the Vec2 position and float angle to

    // translate and rotate the rectangle

    pushMatrix();

    translate(pos.x,pos.y);

    rotate(-a);

    fill(175);

    stroke(0);

    rectMode(CENTER);

    rect(0,0,w,h);

    popMatrix();

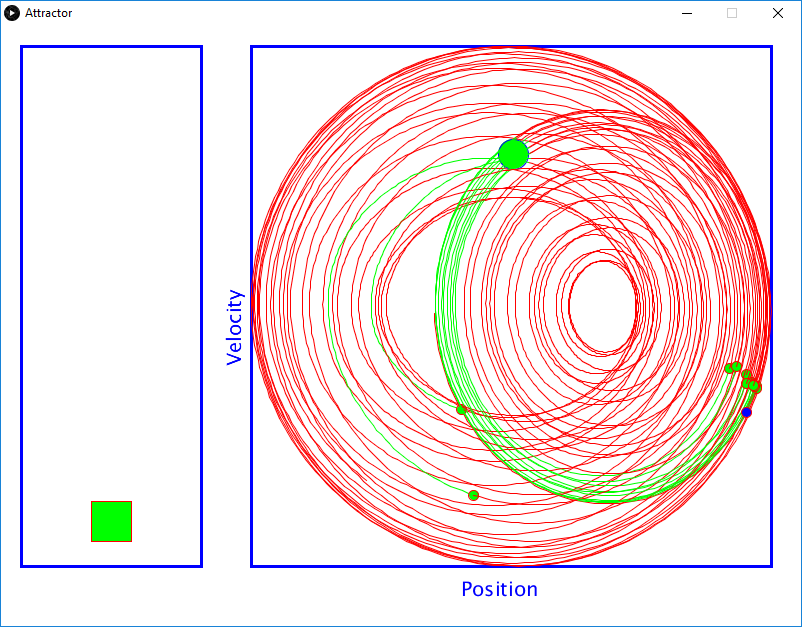
  }

}  // End of the Box Class description

**To Do:** Modify the program supplied to do something slightly different or include a different structure. Take a screen shot of the program running and include it in your Word document. Use the “Copy HTML” option, copy and paste the code from your project to the word document. You may wish to add the line *frameRate(5);* at the start of your program to slow your code down so that you have enough time to press Alt+PrintScreen to capture the image.

**Part 4:** Finally, we are developing VPL code to provide feedback on a program based on its graphic output rather than the source code or run-time console input/output. To help us test this idea write a short program in processing to draw 4 non-overlapping squares on the screen. Just submit the code as four lines of code, you don’t need to add the *setup(){}* and *draw(){}* methods. In VPL enter the code into *upload.pde* and then evaluate it to see the score.

**Extra:** I have added the code at the end of this handout for the Simple Harmonic Oscillator shown in lectures. You could run it at and search for chaotic and non-chaotic regions in the attractor. Just run the code and click on the attractor. To see the Lorentz attractor just uncomment this section of the program and comment out the SHM bit, see lecture notes.



**Figure 7:** Kicked Simple Harmonic Motion

I have also added a zipped project showing how to attach sprites to a Box2D body. This will need to be run using an earlier version of Processing in the Callan Lab; it will run in the Eolas Lab on either version. The code is part of a slightly bigger program that allows the user to catch falling leaves in their arms under the control of the Kinect.



**Figure 8:** Box2D combined with Sprites, catch the falling leaves....

// SHM

int N=10000;

float [] x\_pts=new float[N];

float [] y\_pts=new float[N];

int [] col=new int[N];

float min\_x;

float max\_x;

float min\_y;

float max\_y;

float mx=0;

float my=0;

float m=0.1; // 100 grams

float D=0;

float k=-0.05;

float dt=0.05; //0.008; // Advance in steps of 5ms

float w=sqrt(abs(k/m));

float x=0.0; // Pull string back to 10cm

float v=0.0; // let go, initial velocity is 0

float t=0; // time=0;

int count=0;

void **setup**()

{

  size(800, 600);

  background(255,255,255);

  strokeWeight(1);

  // Calculate the attractor and min max values once

  //float x=1,y=10,z=10;

  for(int i=0;i<N;i++)

  {

  float kick=100\*abs(cos(0.236\*w\*t)); // float kick=0; for normal SHM

    v=v+(((D\*v)+(k\*x)+kick)/m)\*dt;

    x=x+(v\*dt);

  //https://judithcurry.com/2014/05/30/the-astonishing-math-of-michael-ghils-climate-sensitivity/

  // float P = 10, R = 28, B = 8/3; // Lorentz Attractor

  // float dx=P\*(y -x)\*dt;

  // x=x+dx;

  // float dy=((R\*x)-y-(x\*z))\*dt;

  // y=y+dy;

 // float dz=((x\*y)-(B\*z))\*dt;

  // z=z+dz;

   //x\_pts[i]=y;

   //y\_pts[i]=z;

   x\_pts[i]=x;

   y\_pts[i]=v;

   t=t+dt;

  }

  min\_x=x\_pts[0];  // Find max and min

  max\_x=x\_pts[0];

  min\_y=y\_pts[0];

  max\_y=y\_pts[0];

  for(int i=0;i<N;i++)

  {

    if(x\_pts[i]>max\_x) max\_x=x\_pts[i];

    if(x\_pts[i]<min\_x) min\_x=x\_pts[i];

    if(y\_pts[i]>max\_y) max\_y=y\_pts[i];

    if(y\_pts[i]<min\_y) min\_y=y\_pts[i];

  }

}

void **draw**()

{

  background(255,255,255);

  // Frame the plots

  stroke(0,0,255);

  noFill();

  strokeWeight(3);

  float fx=250,fy=20,fw=520,fh=520;

  rect(fx,fy,fw,fh);

  rect(20,fy,180,fh);

  textSize(20);

  text("Position",fx+(fw/2)-50,fy+fh+30);

  float x=fx-10;

  float y=60+fy+fh/2;

  pushMatrix();

  translate(x,y);

  rotate(-HALF\_PI);

  translate(-x,-y);

  text("Velocity", x,y);

  popMatrix();

  strokeWeight(1);

  // When mouse is down highlight 100 iterations that pass through

  if(mousePressed)

  {

  mx=mouseX;

  my=mouseY;

  int flag=0;

  for(int i=0;i<N;i++)

  {

     col[i]=0;

     float screen\_x=fx+(fw\*(x\_pts[i]-min\_x)/(max\_x-min\_x));

     float screen\_y=fy+(fh\*(y\_pts[i]-min\_y)/(max\_y-min\_y));

     if ((pow(mx-screen\_x,2)+pow(my-screen\_y,2))<225)

     {

       flag=100;

     }

     if(flag>0)

     {

      flag--;

      col[i]=1;

     }

  }

  }

  fill(0,255,0);

  ellipse(mx,my,30,30);

  for(int i=1;i<N;i++)

  {

    if(col[i]==0)

    {

       stroke(255,0,0);

      if(col[i-1]==1)

      {

          fill(0,255,0);

          ellipse(fx+(fw\*(x\_pts[i-1]-min\_x)/(max\_x-min\_x)),fy+(fh\*(y\_pts[i-1]-min\_y)/(max\_y-min\_y)),10,10);

      }

    }

    else

    {

      stroke(0,255,0);

    }

    line(fx+(fw\*(x\_pts[i-1]-min\_x)/(max\_x-min\_x)),fy+(fh\*(y\_pts[i-1]-min\_y)/(max\_y-min\_y)),fx+(fw\*(x\_pts[i]-min\_x)/(max\_x-min\_x)),fy+(fh\*(y\_pts[i]-min\_y)/(max\_y-min\_y)));

  }

  rect(90,fy+2+(fh-44)\*(x\_pts[count]-min\_x)/(max\_x-min\_x),40,40);

  fill(0,0,255);

  ellipse(fx+(fw\*(x\_pts[count]-min\_x)/(max\_x-min\_x)),fy+(fh\*(y\_pts[count]-min\_y)/(max\_y-min\_y)),10,10);

  count++;

  if(count>=N) count=0;

}   //Redraw the new screen