PPEM 2020

Librerías de física

Resumen

Comunicación usando

- Arduino
- TCP, UDP
- Requests HTTP
- Websockets
- Protocolo MIDI
- OSC
- OOCSI

Bibliotecas de física en Processing

Box2D para Processing: motor de física en 2D, especialmente recomendable para colisiones.

Fisica: implementación de JBox2D orientada a objetos.

Toxiclibs: el módulo vertletphysics es un simulador físico sencillo para 3D, recomendado para simulaciones de sistemas conectados.

LiquidFunProcessing: LiquidFun es una biblioteca de C++ que extiende Box2D para proporcionar física de partículas y dinámica de fluidos. Está optimizado para generar y renderizar simulaciones de fluidos basados en partículas.

PixelFlow: simulación de fluidos, dinámicas de cuerpos blandos, flujo óptico, procesamiento de imagen, sistemas de partículas, física, ...

Box2D para Processing

Box2D es un motor de física escrito en C++.

Su port a Java es JBox2D y se puede usar en Processing.

Existe la librería “Box2D for Processing” que ayuda en el manejo de JBox2D en Processing.

https://box2d.org/documentation/index.html

https://natureofcode.com/book/chapter-5-physics-libraries/

Mundo de Box2D

- Unidades
- Sistema de coordenadas y ángulos
- Diferencias con Processing en definiciones de vectores, rectángulos y vértices
- Elementos: World, Body, Shape, Fixture, Joint, Vec2D

Box2D está optimizado para trabajar con unidades de **metros-kilogramo-segundo** (MKS). En concreto, Box2D fue afinado para funcionar bien con formas **móviles** entre **0,1** y **10** metros. Las formas **estáticas** pueden tener hasta **50 metros** de largo sin problemas.

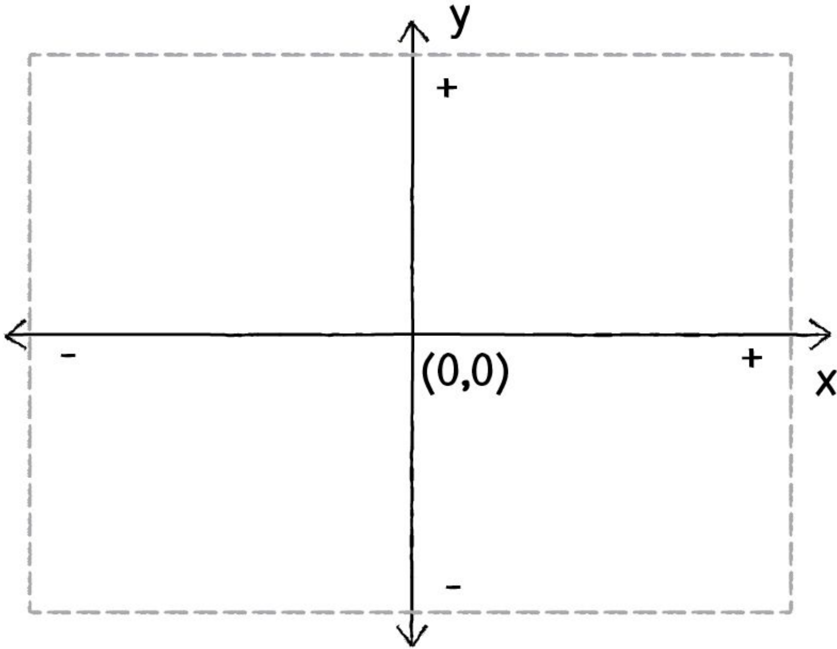
Como vemos no se debería trabajar directamente con pixeles en las simulaciones ya que en entrada de 200px será interpretada como 20m (!). Box2D para Processing dispone de funciones que permiten mapear entre el mundo de pixeles y el mundo de la simulación de física.

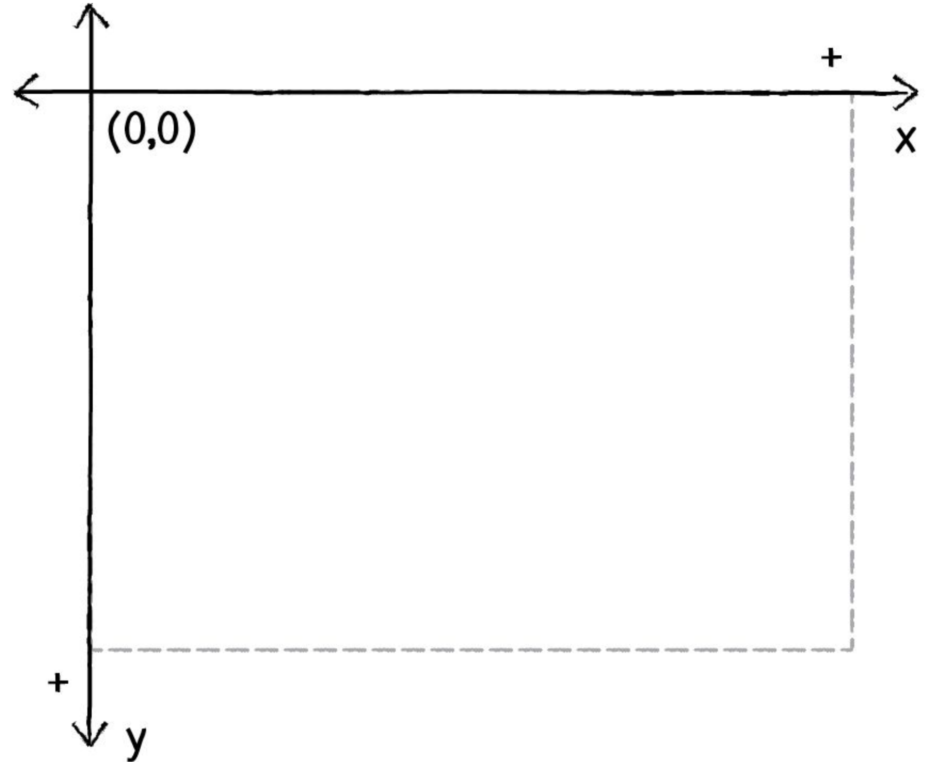
Se recomienda ejecutar el sketch a **60 fps**, ya que es la velocidad a la que se simula la física. La simulación de la física no está conectada al frameRate del sketch, así que incluso si su boceto se ejecuta a 30 fps, la física seguirá avanzando 1/60 de segundo cada frame. Puede cambiar el paso del tiempo haciendo physics.getSettings (). Hz = newFrequency; pero tenga en cuenta que este motor sólo está ajustado para trabajar con 60 Hz, por lo que puede estar inestabilidad o los resultados mala calidad si uno lo hace.

Se usan **radianes** para definir ángulos.

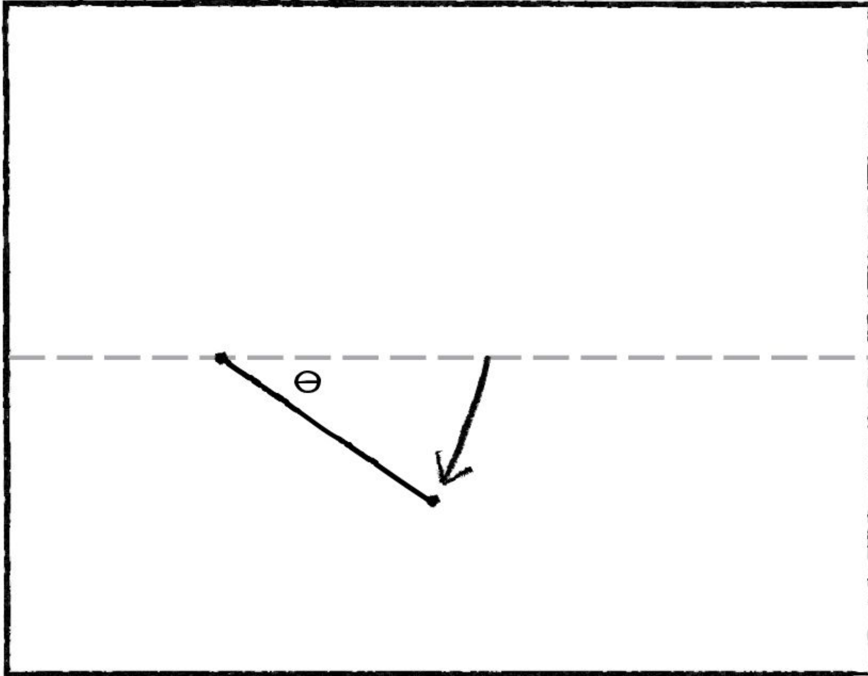
La **gravedad** por defecto es: b2Vec2 gravity(0, -9.8); // gravedad de la tierra, 9.8 m/s/s hacia abajo

Sistema de coordenadas en Box2D

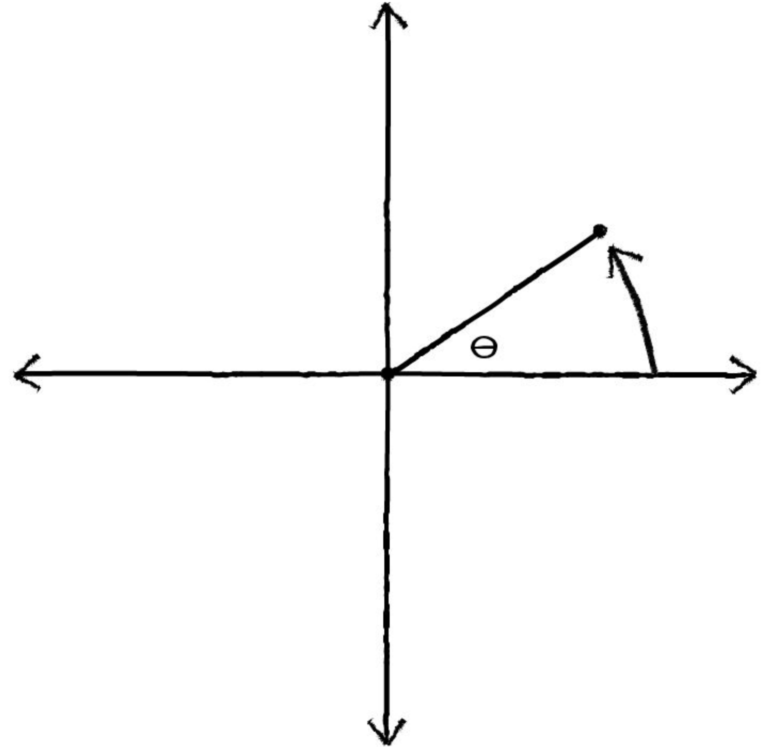




Ángulos



clockwise rotation in Processing



counterclockwise rotation in Box2D

Coordenadas Box2D y Processing

Posiciones y dimensiones en el mundo de Processing -> convertir a coordenadas y dimensiones en mundo físico ("world") -> pedir los cálculos a Box2D -> convertir los resultados de la simulación a pixeles

Funciones auxiliares:

coordWorldToPixels(Vec2 world) // Convertir la posición de World a píxeles

coordWorldToPixels(float worldX, float worldY) // Convertir la posición de World a píxeles

coordPixelsToWorld(Vec2 screen) // Convertir la posición de píxeles a World

coordPixelsToWorld(float pixelX, float pixelY) // Convertir la posición de píxeles a World

scalarPixelsToWorld(float val) // Escalar una dimensión (ancho, alto, o radio) de píxeles a World

scalarWorldToPixels(float val) //Escalar una dimensión de World a píxeles

Por defecto 10 píxeles = 1m

Vectores

Definición de los rectángulos

PolygonShape compuesto por vértices

Diferencias PVector y Vec2

PVector	Vec2

`PVector a = new PVector(1,-1);`
`PVector b = new PVector(3,4);`
`a.add(b);`
	`Vec2 a = new Vec2(1,-1);`
	`Vec2 b = new Vec2(3,4);`
	`a.addLocal(b);`

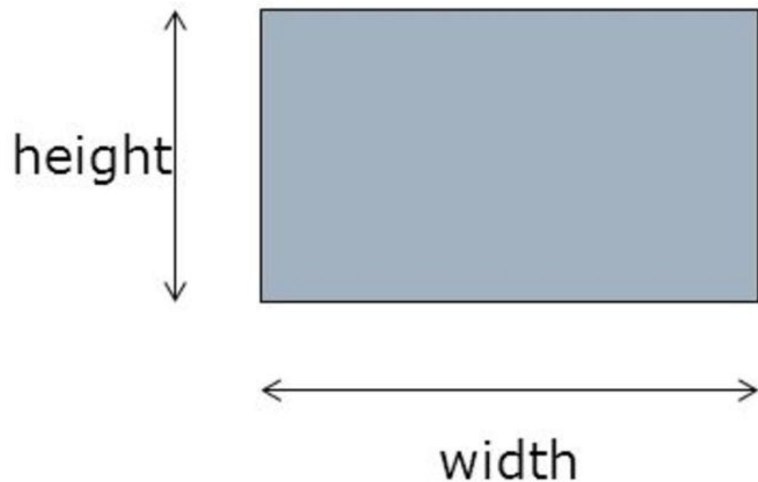
`PVector a = new PVector(1,-1);`
`PVector b = new PVector(3,4);`
`PVector c = PVector.add(a,b);`
	`Vec2 a = new Vec2(1,-1);`
	`Vec2 b = new Vec2(3,4);`
	`Vec2 c = a.add(b);`

`PVector a = new PVector(1,-1);`
`float n = 5;`
`a.mult(n);`
	`Vec2 a = new Vec2(1,-1);`
	`float n = 5;`
	`a.mulLocal(n);`

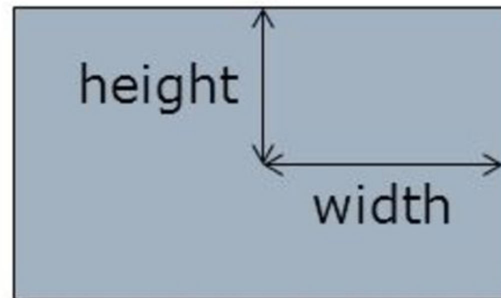
`PVector a = new PVector(1,-1);`
`float n = 5;`
`PVector c = PVector.mult(a,n);`
	`Vec2 a = new Vec2(1,-1);`
	`float n = 5;`
	`Vec2 c = a.mul(n);`

`PVector a = new PVector(1,-1);`
`float m = a.mag();`
`a.normalize();`
	`Vec2 a = new Vec2(1,-1);`
	`float m = a.length();`
	`a.normalize();`

Rectángulo en Box2D



(modo CORNER por defecto)

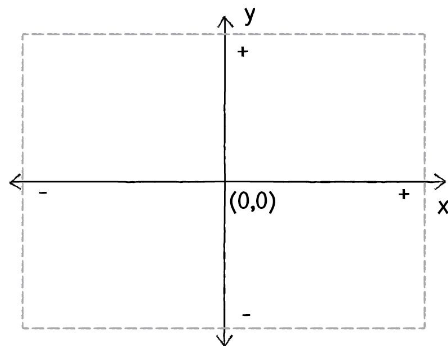


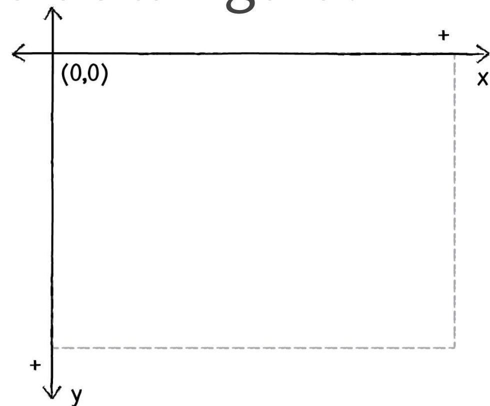
BOX2D

(equivalente a modo RADIUS)

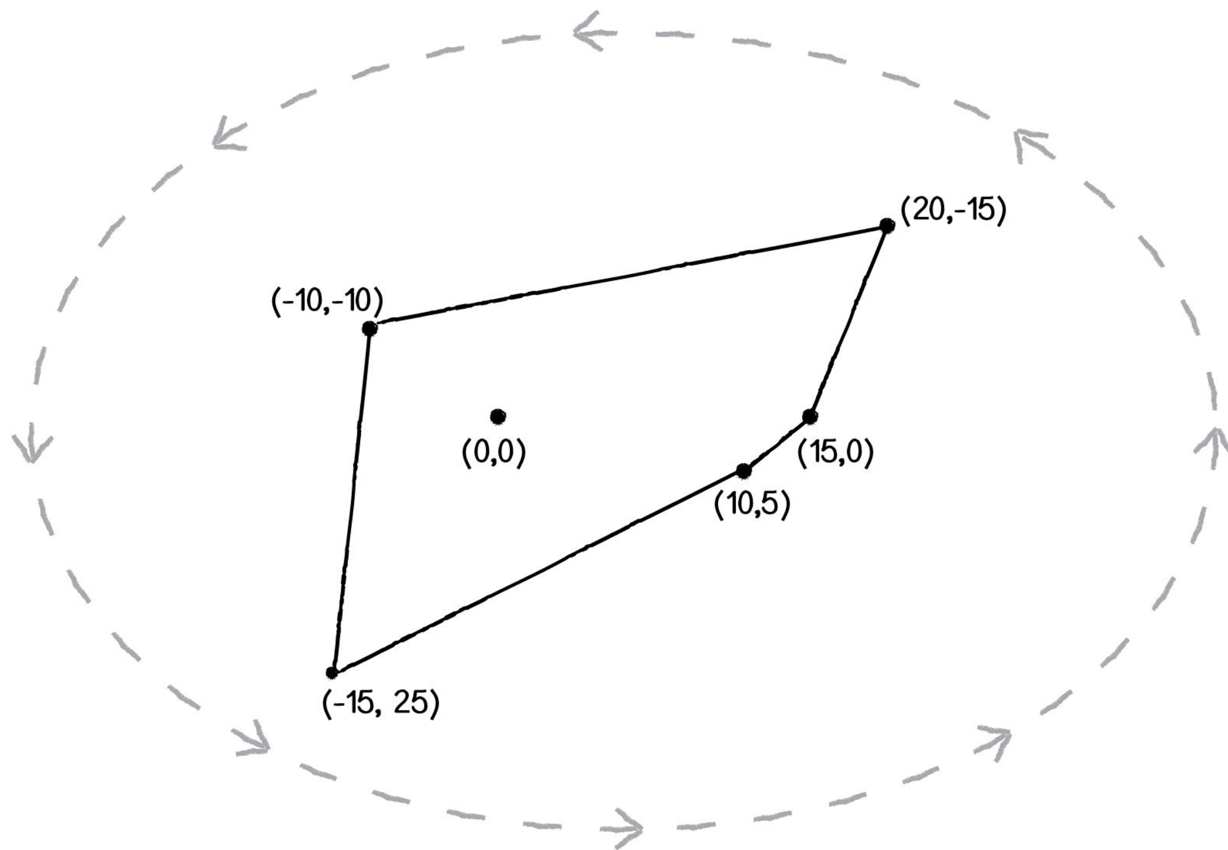
PolygonShape compuesto por vértices

- Se definen los vértices en el sentido horario en Box2D pero como el eje Y está al revés, al nivel de píxeles definimos en el sentido anti-horario.
- Se define los vértices en referencia al centro de la figura!
- Convexo, no concavo!

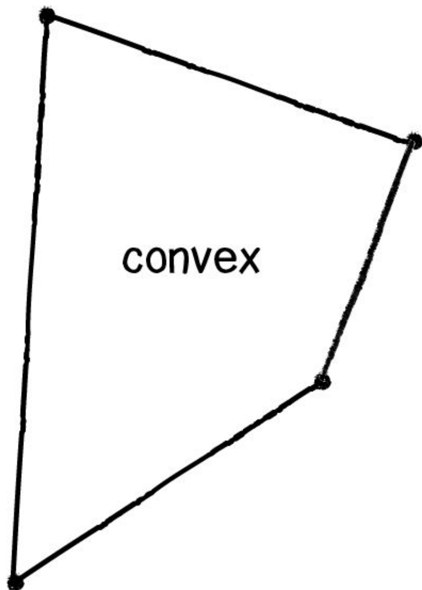


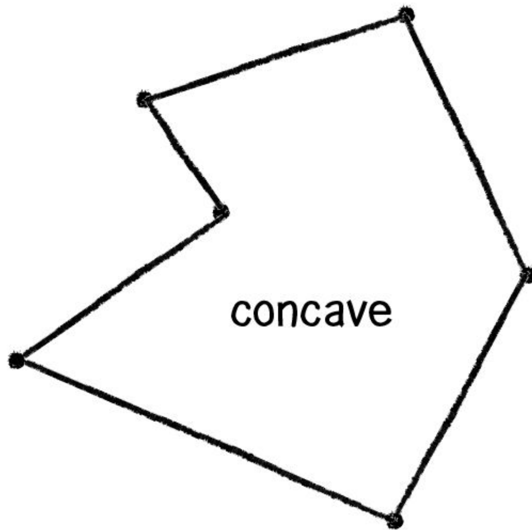


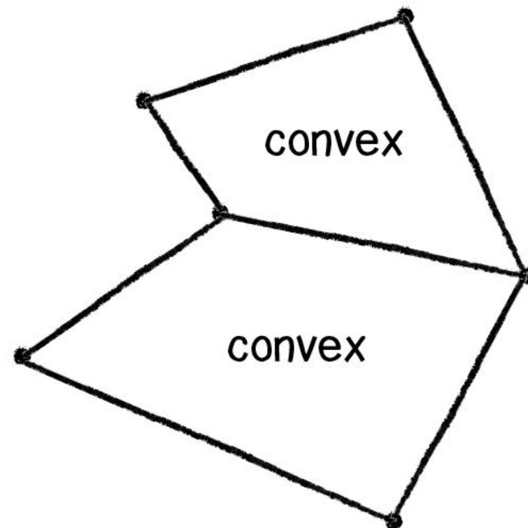
PolygonShape compuesto por vértices



PolygonShape y colisiones







Elementos básicos

- **World**: Gestiona la simulación física. Sabe todo sobre el espacio global de coordenadas y también almacena listas de todos los elementos del mundo.
- **Body**: Sirve como el elemento primario en el mundo de Box2D. Tiene una ubicación y velocidad.
- **Shape**: Define la forma del cuerpo necesaria para calcular las colisiones.
- **Fixture**: Conecta la forma al cuerpo y establece propiedades tales como densidad, fricción y restitución.
- **Joint**: Actúa como una conexión entre dos cuerpos (o entre un cuerpo y el mundo mismo).
- **Vec2**: Describe un vector en el mundo Box2D.

1. Crear una definición del cuerpo (BodyDef)
   - Posición (del mundo de física, no del mundo de los píxeles!)
   - Tipo (static, dynamic, kinematic)
2. Crear cuerpo pasandole la definición
3. Crear forma (PolygonShape, CircleShape, EgdeShape, ChainShape)
4. Crear fixture (FixtureDef)
   - Definir densidad (density)
   - Definir fricción (friction)
   - Definir restitución (restitution)
5. Asociar fixture a body

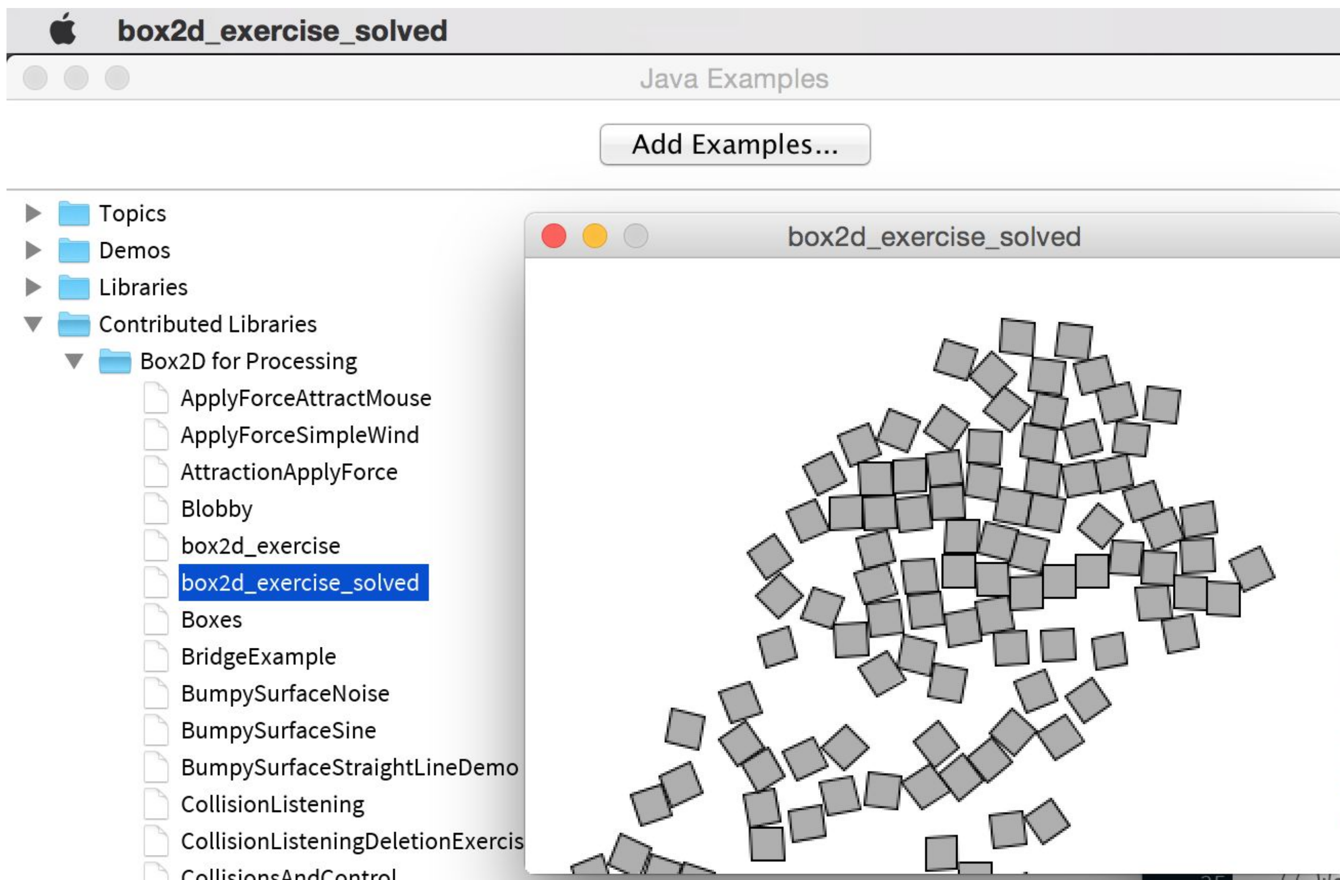
Tipos de cuerpos

- Estático (static): no lo afectan las fuerzas en el mundo, no reacciona a las colisiones, no se mueve
- Dimámico (dynamic): afectado por las fuerzas en el mundo, reacciona a las colisiones
- Cinemático (kinematic): se puede mover manualmente (por el usuario) configurando directamente su velocidad. Los cuerpos cinemáticos colisionan solo con cuerpos dinámicos y no con otros cuerpos estáticos o cinemáticos.

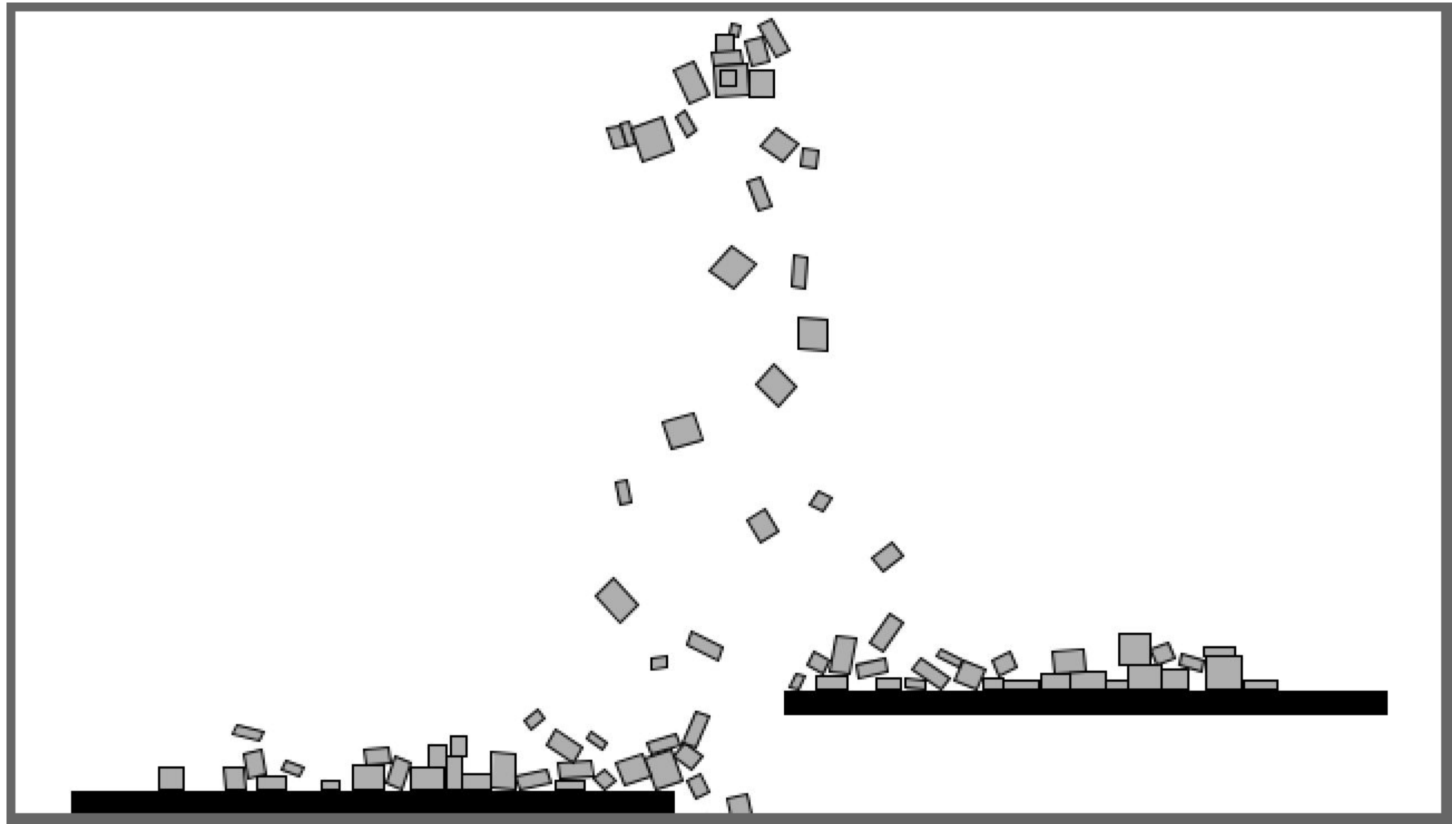
Ejemplo de un PolygonShape

```
import shiffman.box2d.*;
import org.jbox2d.collision.shapes.*;
import org.jbox2d.common.*;
import org.jbox2d.dynamics.*;

Box2DProcessing box2d;
Body body;
int w,h;
void setup(){
  size(400,400);
  box2d = new Box2DProcessing(this);
  box2d.createWorld(); //por defecto la gravedad esta en -10
  w = h = 50;

  // 1. crear "body definition"
  BodyDef bd = new BodyDef();
  bd.type = BodyType.DYNAMIC;
  bd.position.set(box2d.coordPixelsToWorld(100,100));
  // 2. crear "body" pasandole la definición
  body = box2d.createBody(bd);
  // 3. crear "shape"
  PolygonShape ps = new PolygonShape();
  float box2dW = box2d.scalarPixelsToWorld(w/2); //importante!! /2 !!
  float box2dH = box2d.scalarPixelsToWorld(h/2);
  ps.setAsBox(box2dW, box2dH);
  // 4. crear "fixture" para unir "body" y "shape"
  FixtureDef fd = new FixtureDef();
  fd.shape = ps;
  fd.density = 1;
  fd.friction = 0.3;
  fd.restitution = 0.5;
  // 5. asocia "fixture" al “body”
  body.createFixture(fd); // = body.createFixture(ps,1);
}

void draw(){
  box2d.step(); // !!! importante, sin esto la simulación no avanza
  Vec2 pos = box2d.getBodyPixelCoord(body);
  float a = body.getAngle();
  rectMode(CENTER);
  pushMatrix();
  rect(pos.x,pos.y,w,h);
  popMatrix();
}
```

Ejemplo con varios elementos interactuando

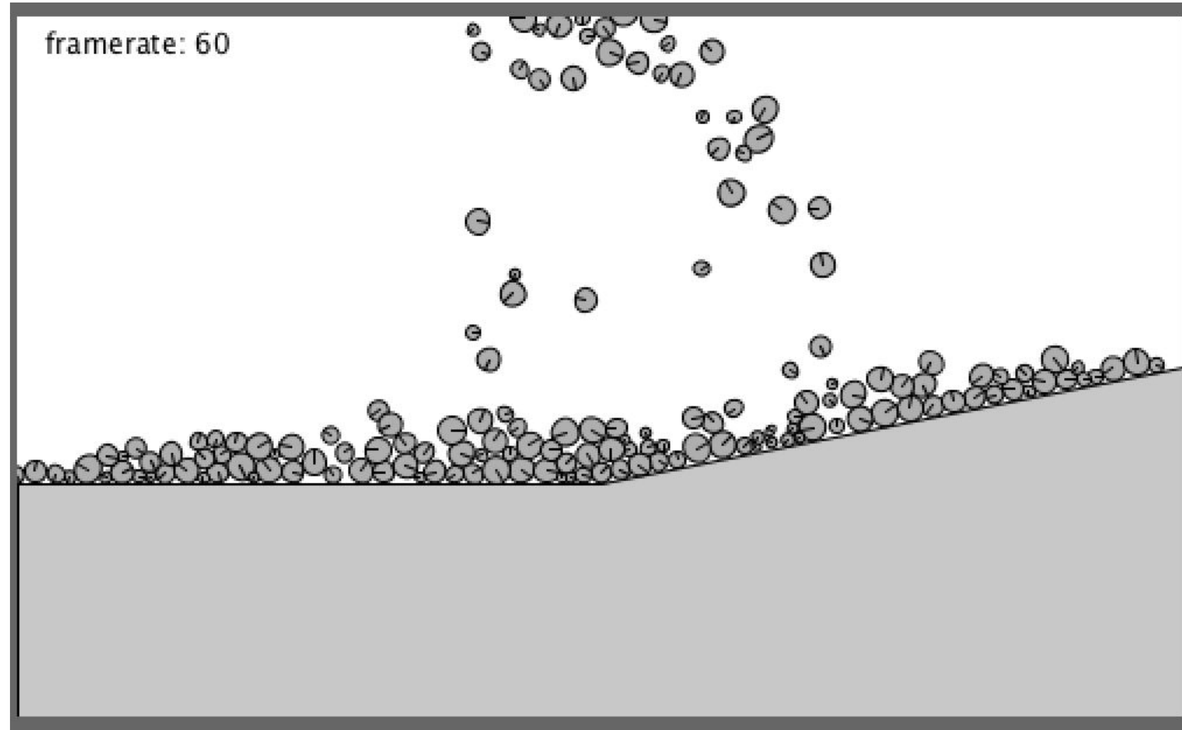


Los cuerpos dinámicos y estáticos / “Boxes”



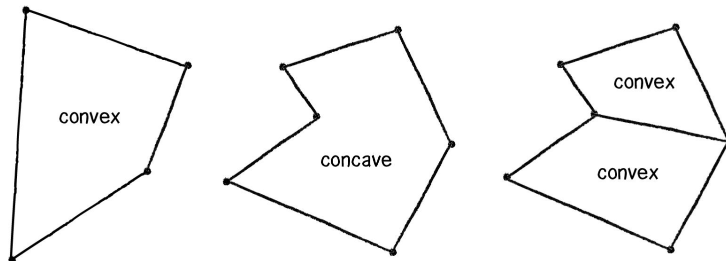
Cuerpo estático definido por vértices

ChainShape y CircleShapes : BumpySurfaceStraightLineDemo

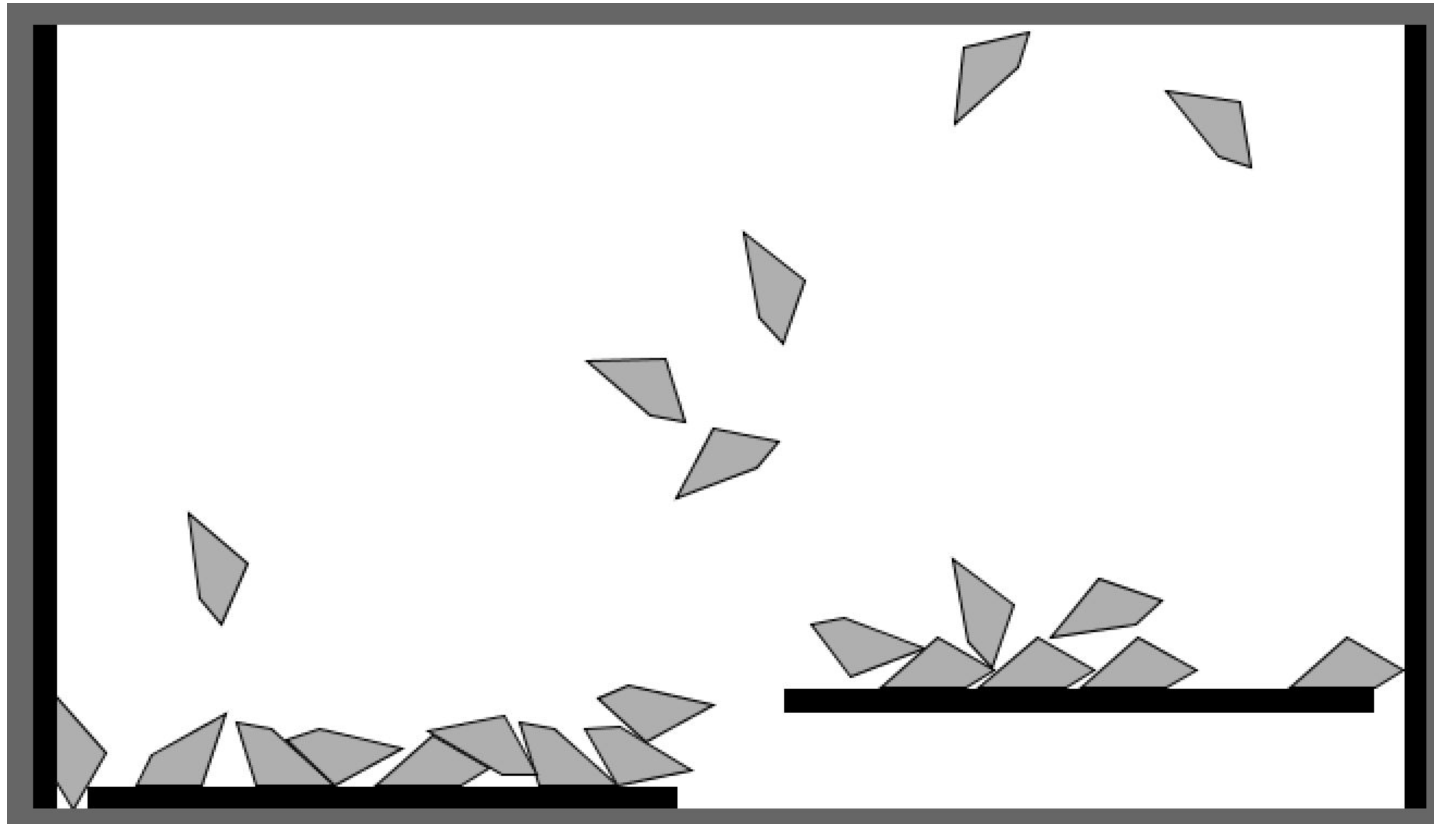


Ejemplos de formas más complejas

- Polígono customizado
  - solo convexo, no concavo!
  - se definen los vértices en el sentido horario en Box2D pero como el eje Y está al revés, al nivel de píxeles definimos en el sentido anti-horario.
  - se define los vértices en referencia al centro de la figura!
- Cuerpos conectados (Ojo! Se alinean los centros de los shapes! Hay que pensar en offsets!)
- Joints: distance joint y revolute joint

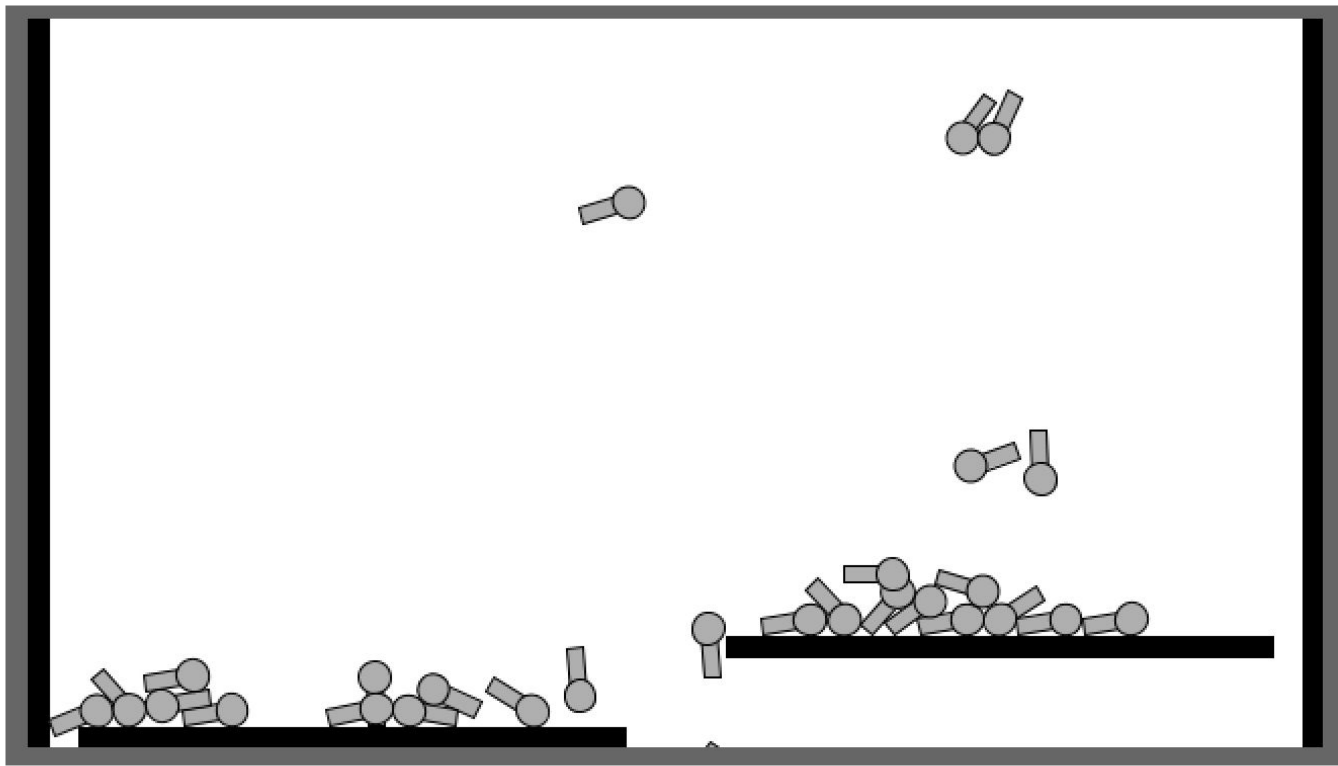


Polígonos customizados // Polygons



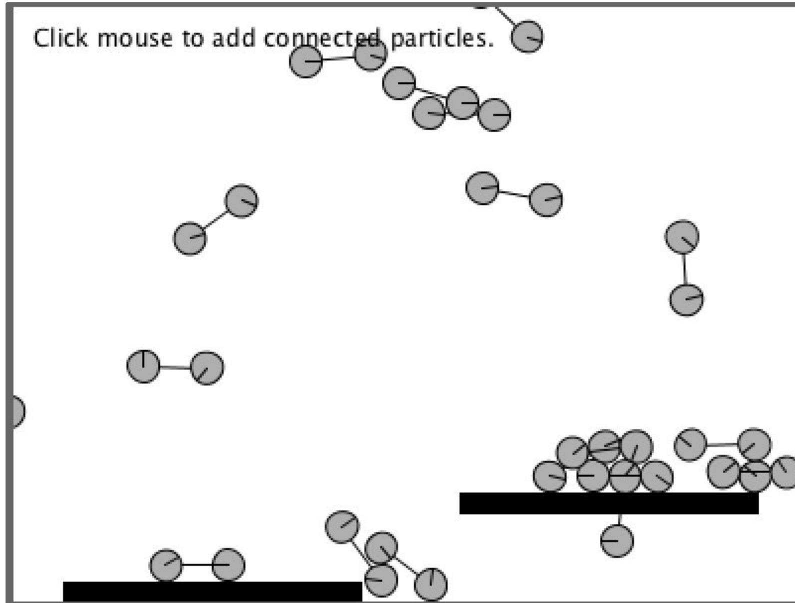
Cuerpos conectados // MultiShapes

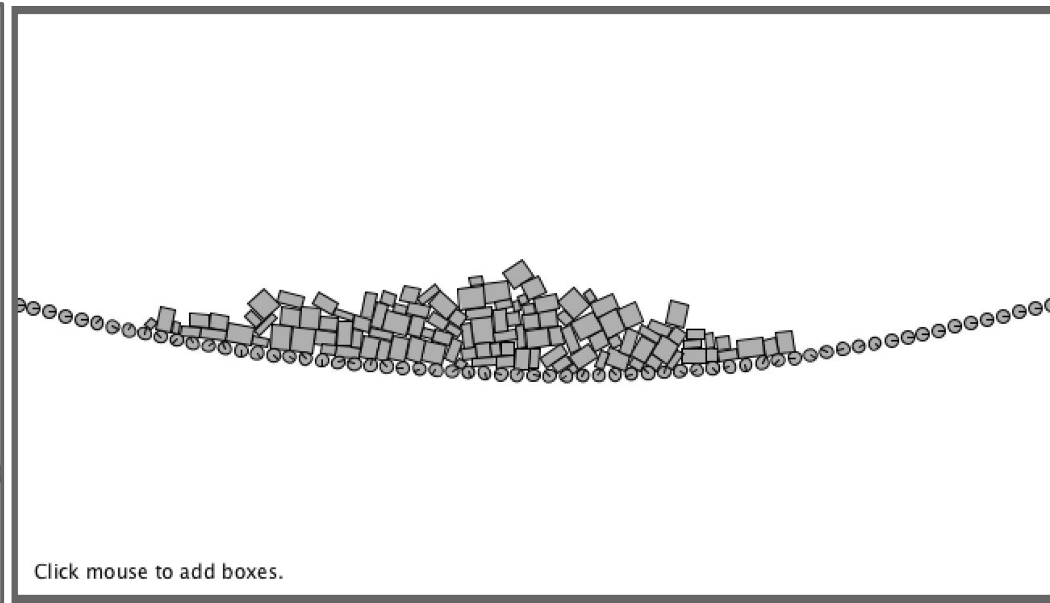
Ojo! Se alinean los centros de los shapes! Hay que pensar en offsets!



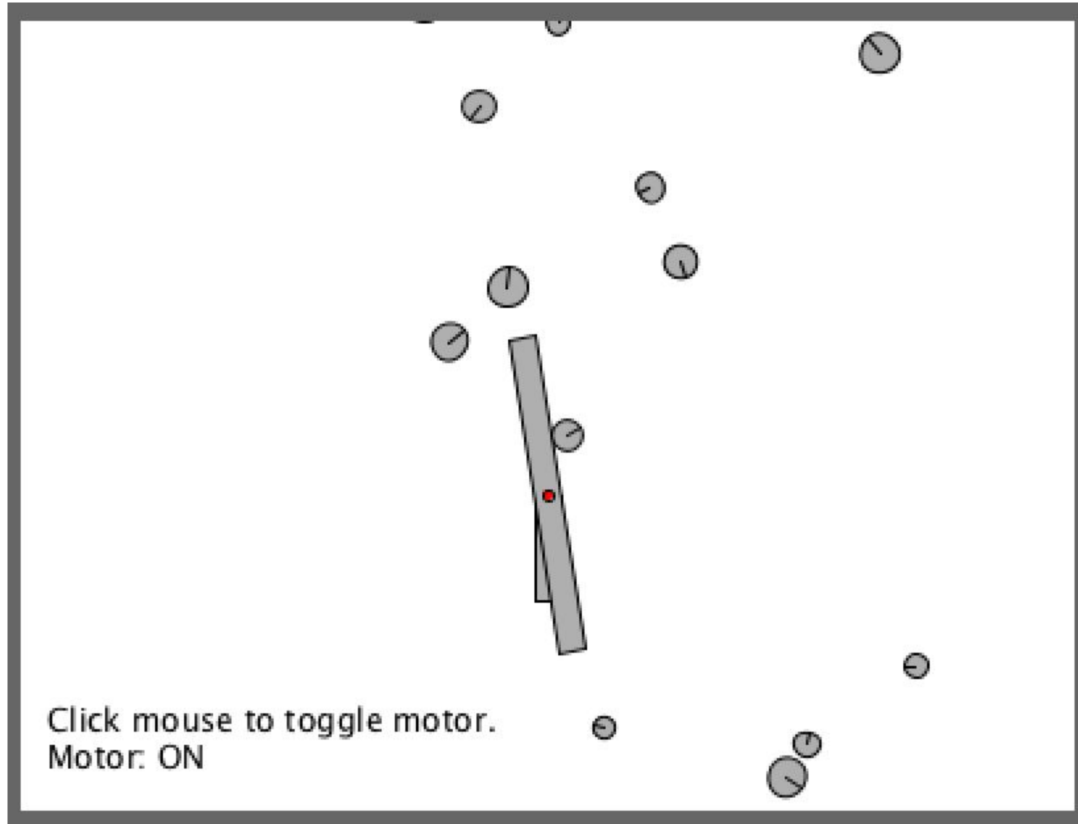
Distance joint

DistanceJointExample y BridgeExample





Revolute joint // RevoluteJointExample

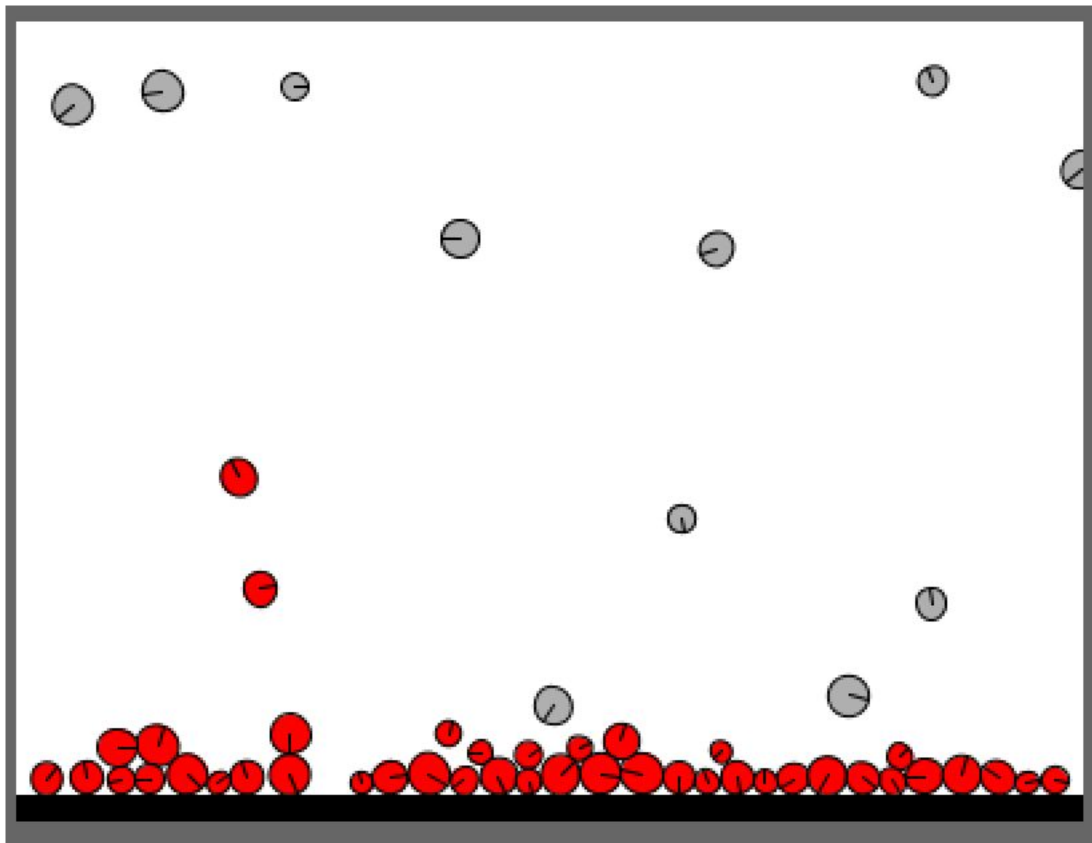


Con el **mouse** mediante MouseJoint: ejemplo Mouse

Controlando **fuerzas externas**: ApplyForceSimpleWind, ApplyForceAttractMouse, AttractionApplyForce

En **colisiones**: CollisionListening

Ejemplo colisión / CollisionListening



Facilita la creación de simulaciones físicas 2D en Processing

- Orientación a objetos
- Valores por defecto
- Implementa casos de uso más comunes
- Gestión de estilos como Processing
- Gestión de eventos como Processing

http://www.ricardmarxer.com/fisica/

Ejemplo de un PolygonShape ...

```
import shiffman.box2d.*;
import org.jbox2d.collision.shapes.*;
import org.jbox2d.common.*;
import org.jbox2d.dynamics.*;

Box2DProcessing box2d;
Body body;
int w,h;
void setup(){
  size(400,400);
  box2d = new Box2DProcessing(this);
  box2d.createWorld(); //por defecto la gravedad esta en -10
  w = h = 50;

  // crear "body definition"
  BodyDef bd = new BodyDef();
  bd.type = BodyType.DYNAMIC;
  bd.position.set(box2d.coordPixelsToWorld(100,100));
  // crear "body"
  body = box2d.createBody(bd);
  // crear "shape"
  PolygonShape ps = new PolygonShape();
  float box2dW = box2d.scalarPixelsToWorld(w/2); //importante!! /2 !!
  float box2dH = box2d.scalarPixelsToWorld(h/2);
  ps.setAsBox(box2dW, box2dH);
  // crear "fixture" para unir "body" y "shape", sin definición de fricción,etc.
  body.createFixture(ps,1);  // paso PolygonShape y densidad en 1
}

void draw(){
  box2d.step();// !!!
  Vec2 pos = box2d.getBodyPixelCoord(body);
  float a = body.getAngle();
  rectMode(CENTER);
  pushMatrix();
  translate(pos.x,pos.y);
  rotate(-a);
  rect(0,0,w,h);
  popMatrix();
}
```

… y lo mismo en fisica

```
import fisica.*;
FWorld world;

void setup() {
  size(400, 400);
  Fisica.init(this);
  world = new FWorld();
  FBox b = new FBox(50, 50); // por defecto en física 20 píxeles == 1 metro
  b.setPosition(100, 100);
  world.add(b);
}

void draw() {
  world.step();
  world.draw();
}
```

Valores por defecto

- El tamaño del mundo es 3 veces más grande que el del sketch. Los cuerpos que salen del mundo dejan de simularse.
- step() considera un incremento de tiempo de 1/60 de segundo
- Los cuerpos móviles se pueden agarrar con el ratón
- 20 píxeles en pantalla == 1 metro en el mundo de la simulación

Propiedades del mundo

Bordes: se pueden modificar algunas de las propiedades de los bordes

mundo.setEdgesFriction(0)

mundo.setEdgesRestitution(1)

Gravedad: el mundo tiene una fuerza de gravedad que afecta a todos los cuerpos móviles. Se puede modificar

mundo.setGravity(0, -100)

Agarrables: los cuerpos del mundo se pueden agarrar con el ratón. Esto se puede evitar

mundo.setGrabbable(false)

Cuerpo en un punto: se le puede pedir el cuerpo (o los cuerpos) que se encuentra en un punto determinado del mundo

mundo.getBody(200, 200)

mundo.getBodies(200, 200)

Cuerpos simples: FBox, FCircle, FPoly, FLine, FBlob

Cuerpos compuestos: FCompound (cuerpos compuestos)

Joints: FDistanceJoint, FRevoluteJoint, … y más joints!

Propiedades de la dinámica de los cuerpos

Posición: posición en píxeles

cuerpo.setPosition(100, 200)

Rotación: rotación en radianes

cuerpo.setRotation(PI/4)

Velocidad: velocidad en píxeles por segundo

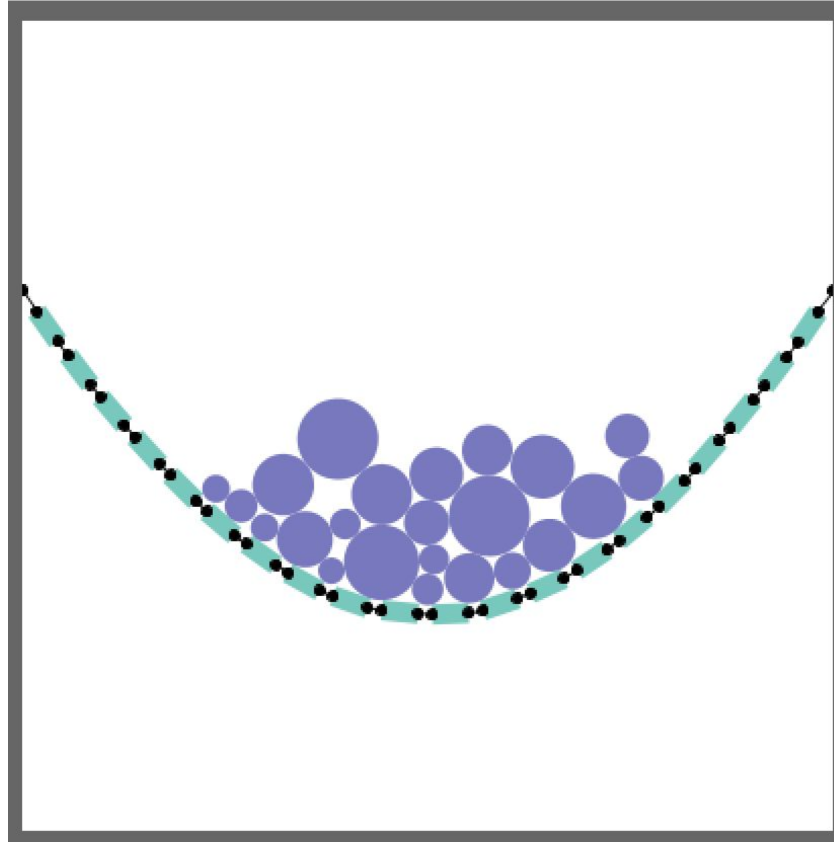
cuerpo.setVelocity(50, 0)

Velocidad angular: velocidad de rotación en radianes por segundo

cuerpo.setAngularVelocity(PI/16)

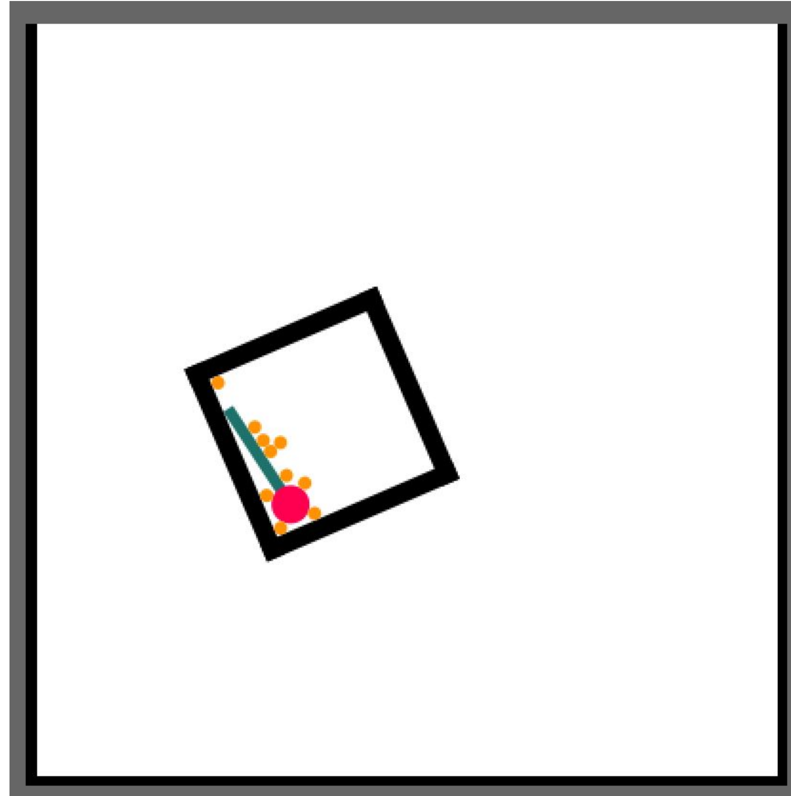
Ejemplo con FBox, FCircle y FDistanceJoint

Ejemplo “Anchors”

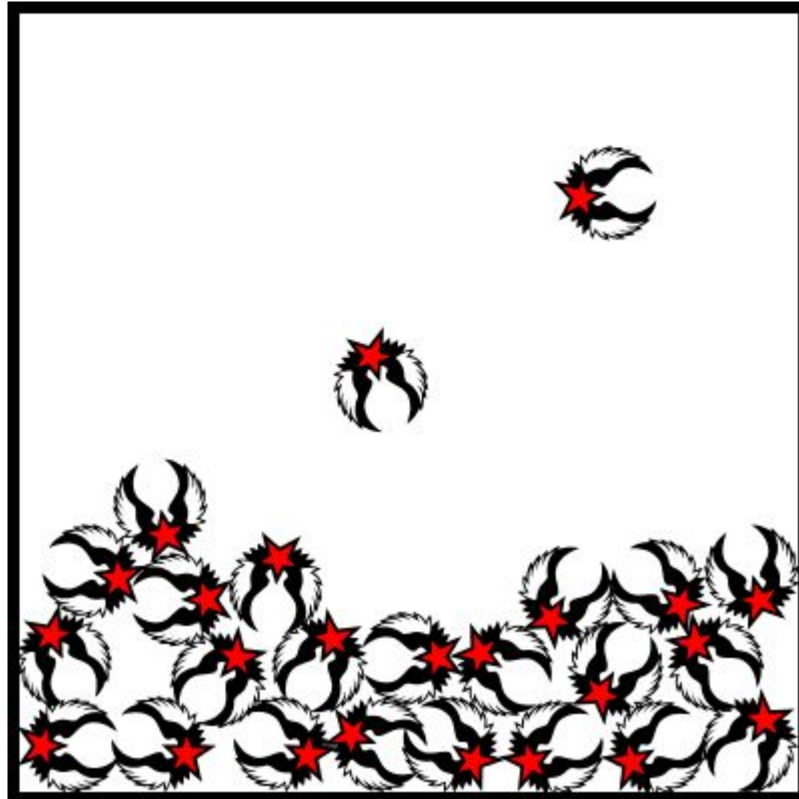


FCompound // Ejemplo Compound

setBullet(true)



Imágen en el mundo físico // WingedStar



Densidad: densidad de masa del objeto en gramos por píxel al cuadrado

cuerpo.setDensity(0.3)

Restitución: pérdida de velocidad durante el rebote (componente perpendicular de una colisión) [valor de 0 a 1 (no pierde)]

cuerpo.setRestitution(0.3)

Fricción: pérdida de velocidad durante el rozamiento (componente tangencial de una colisión) [valor de 0 a 1]

cuerpo.setFriction(0.3)

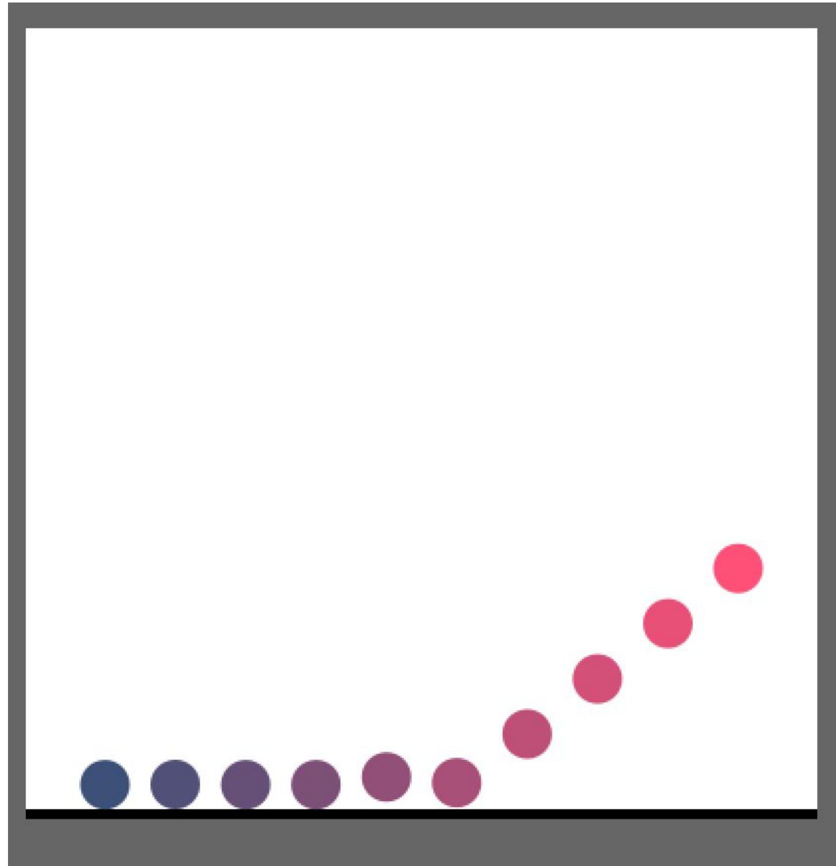
Amortiguación: pérdida de velocidad durante el movimiento [valor de 0 a 1]

cuerpo.setDamping(0.3)

Amortiguación Angular: pérdida de velocidad durante la rotación [valor de 0 a 1]

cuerpo.setAngularDamping(0.3)

Restitución // Restitution



Propiedades de dibujo

Color de Relleno: color de relleno del cuerpo, mismo formato que Processing

cuerpo.setFill(100, 20, 130)

Color de Trazado: color de trazado del cuerpo, mismo formato que Processing

cuerpo.setStroke(100, 20, 130)

Grosor de Trazado: grosor de trazado del cuerpo, mismo formato que Processing

cuerpo.setStrokeWeight(3)

Imagen: dibujar el cuerpo mediante una imagen

PImage imagen = loadImage("canica.png");

cuerpo.attachImage(imagen)

Los contactos como eventos

```
void contactStarted(FContact contacto) { // código a ejecutar  }

void contactPersisted(FContact contacto) { // código a ejecutar }

void contactEnded(FContact contacto) { // código a ejecutar }
```

Acceso a los contactos

Los contactos se le pueden pedir al cuerpo

cuerpo.getContacts()

También se le puede pedir si está tocando un determinado cuerpo

cuerpo.isTouchingBody(otroCuerpo)

O bien pedirle los cuerpos que está tocando

cuerpo.getTouching()

ContactResize

toxiclibs es una **colección de bibliotecas** de código abierto para tareas de diseño computacional con Java y Processing desarrollado por Karsten "toxi" Schmidt. Las clases se mantienen razonablemente genéricas con el fin de maximizar la reutilización en diferentes contextos que van desde el diseño generativo, animación, interacción / diseño de interfaz, visualización de datos a la arquitectura y la fabricación digital, el uso como herramienta de enseñanza y mucho más.

El módulo vertletphysics es un simulador físico sencillo para **3D**, recomendado para simulaciones de **sistemas conectados**.

Comparación con Box2D

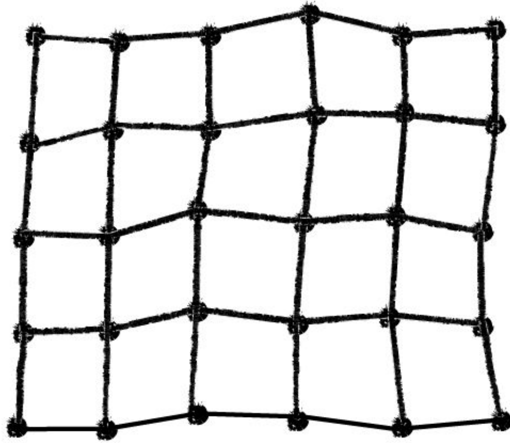
Feature	Box2D	toxiclibs VerletPhysics
Collision geometry	Yes	No
3D physics	No	Yes
Particle attraction / repulsion forces	No ?	Yes
Spring connections	Yes	Yes
Other connections: revolute, pulley, gear, prismatic	Yes	No
Motors	Yes	No
Friction	Yes	No

Comparación con Box2D

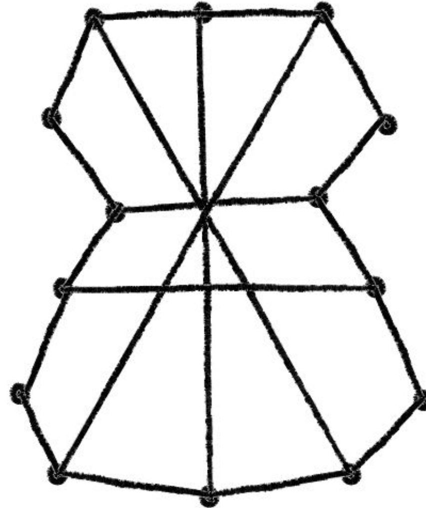
Box2D	toxiclibs VerletPhysics
World	VerletPhysics
Body	VerletParticle
Shape	Nothing! toxiclibs does not handle shape geometry
Fixture	Nothing! toxiclibs does not handle shape geometry
Joint	VerletSpring

Sistemas conectados en toxiclibs

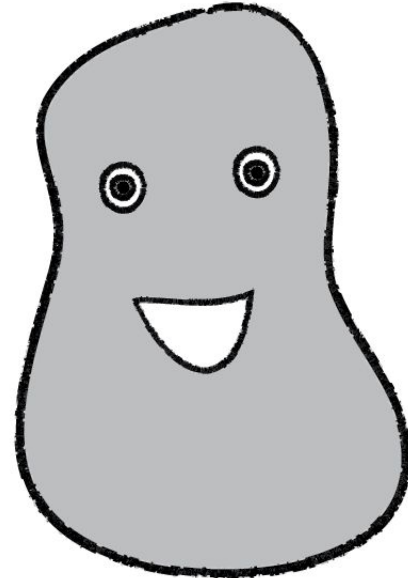
string



blanket

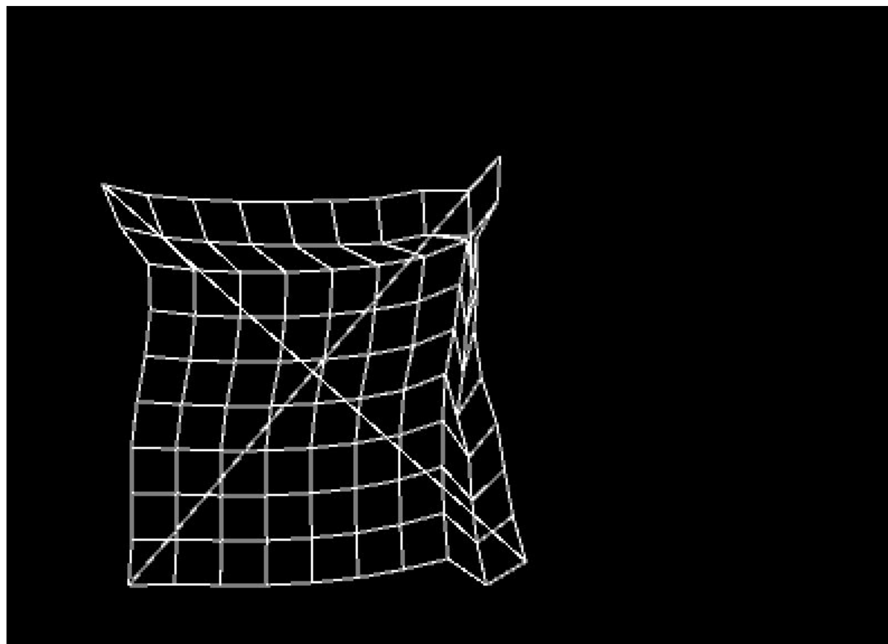


skeleton

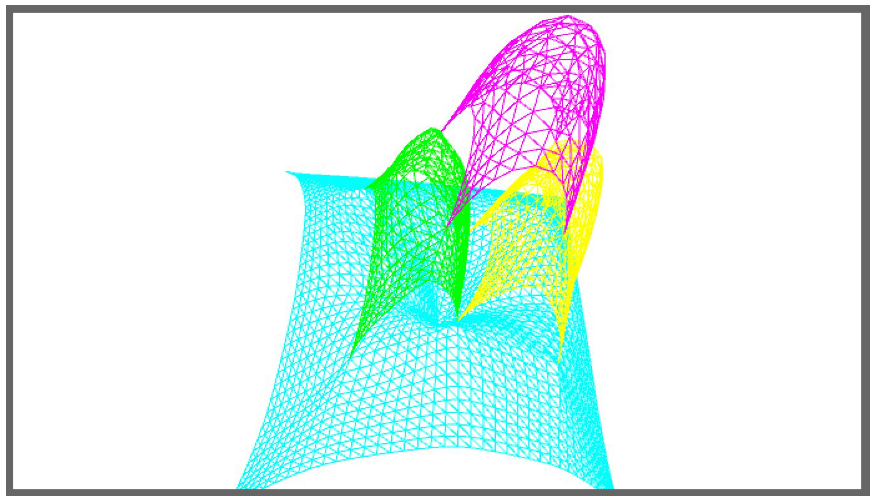


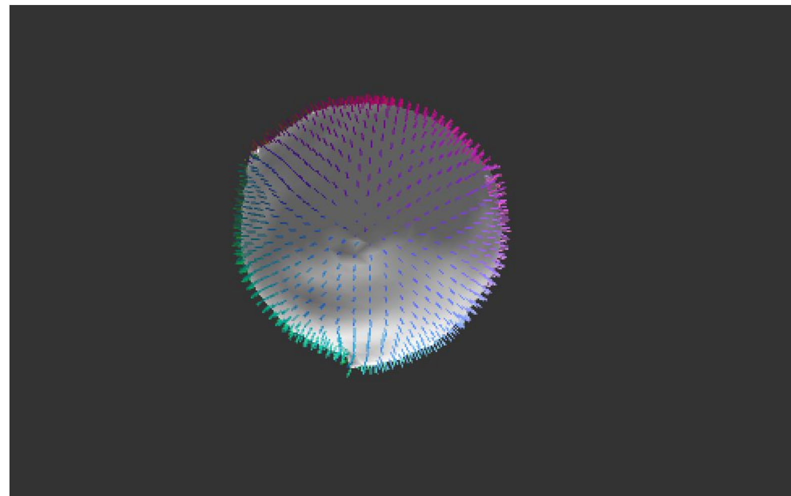
cuddly friend

Ejemplo de sistema conectado //SoftBodySquare



Simulación 3D // JoinedCatenary, InflateMesh



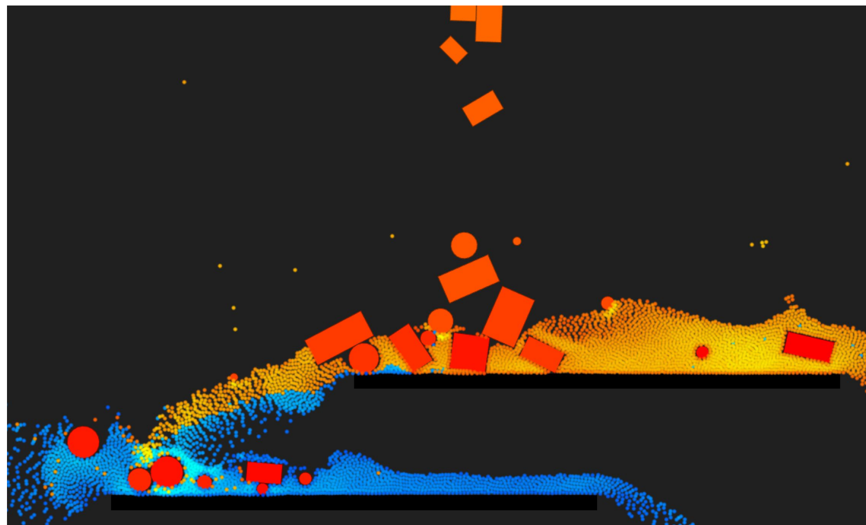


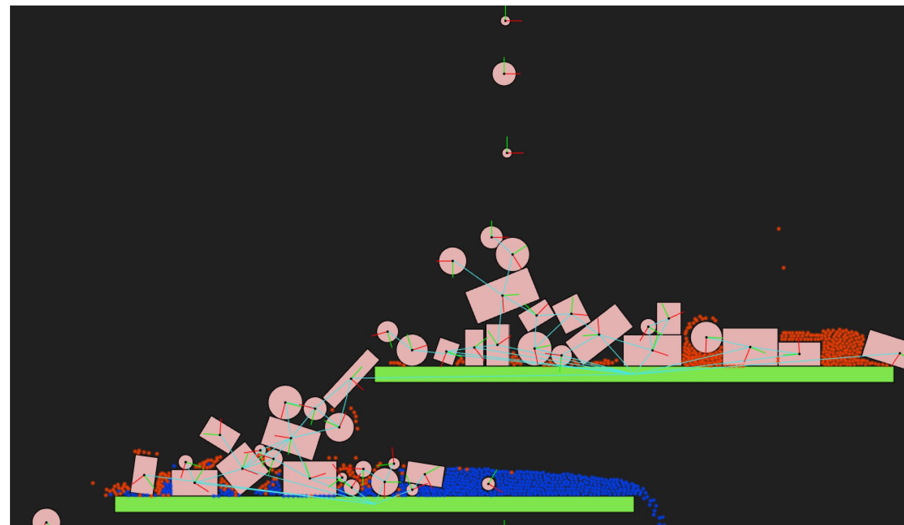
https://github.com/diwi/LiquidFunProcessing

LiquidFun es una biblioteca de C++ que extiende Box2D para proporcionar **física de partículas** y **dinámica de fluidos**. Está optimizado para generar y renderizar simulaciones de fluidos basados en partículas.

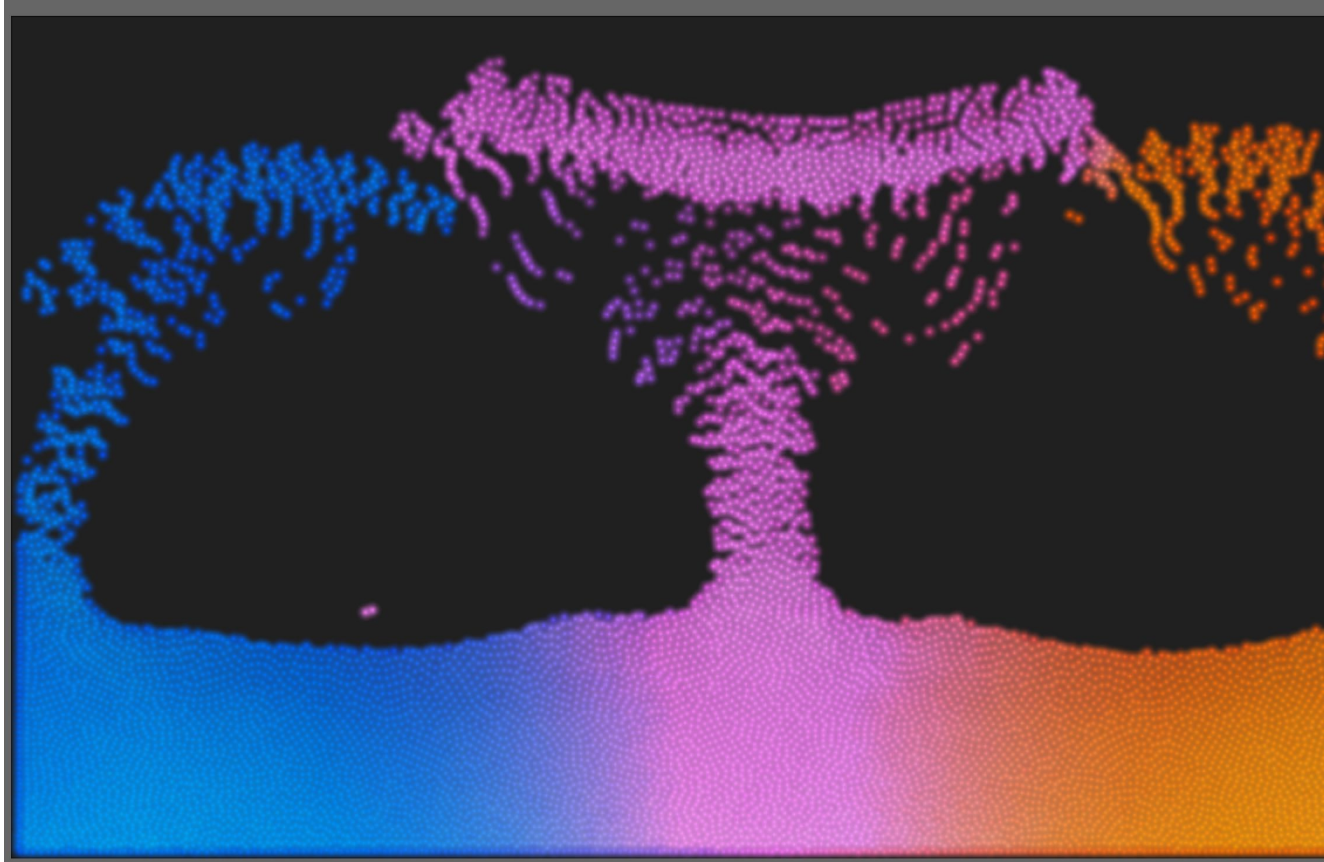
http://google.github.io/liquidfun/Programmers-Guide/html/index.html

Basic_Tutorial

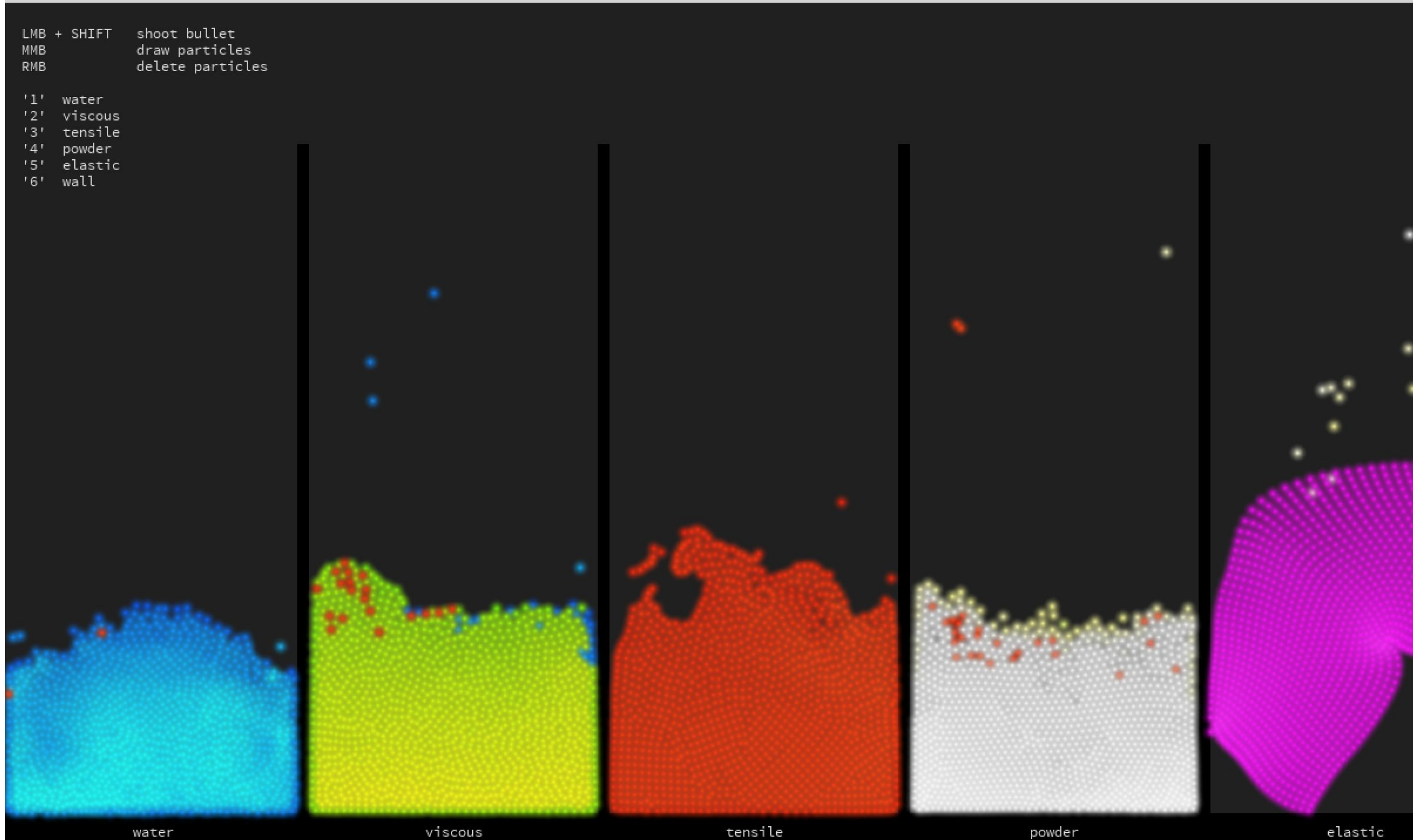




liquidfun_DamBreak



liquidfun_ParticleTypes

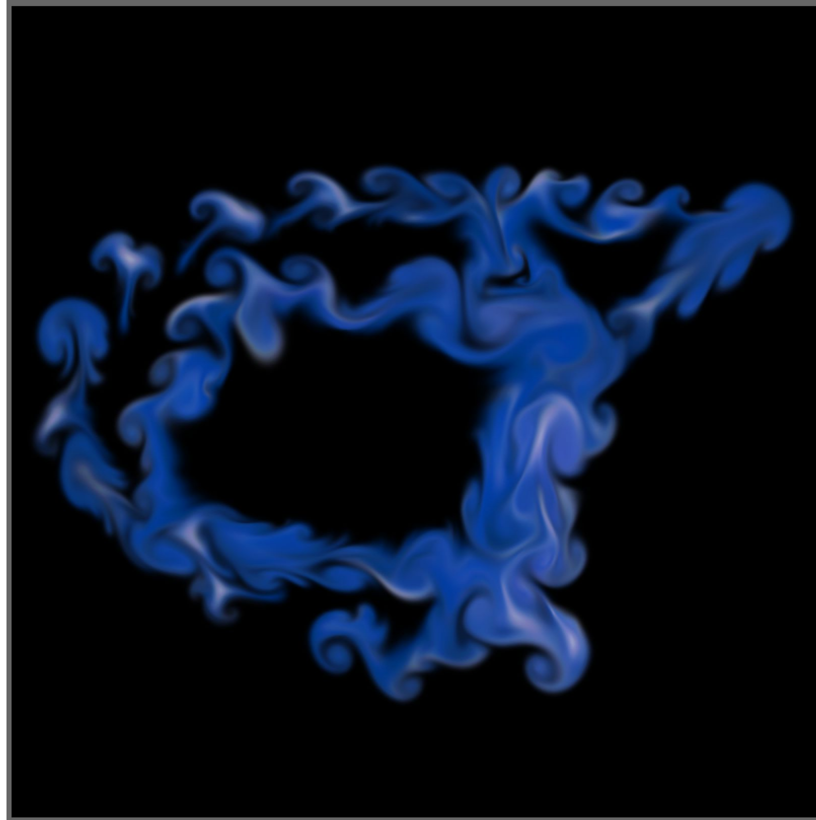


Librería PixelFlow

Librería para computos en GPU (GLSL) de alta performance. Para simulación de fluidos, dinámicas de cuerpos blandos, flujo óptico, procesamiento de imagen, sistemas de partículas, física,...

https://github.com/diwi/PixelFlow

Fluid2D/Fluid_GetStarted



PixelFlow en acción

https://vimeo.com/184850333

Entrega 4 - Juego de física con comunicación

Una simulación de caída de caramelos del cielo (imágenes que se dibujan en la posición definida por la simulación). Al nivel del piso hay un personaje que se mueve a la derecha y izquierda con las flechas del teclado (cuerpo cinemático a cual aplicamos fuerzas para moverlo?). Si el personaje logra agarrar el caramelo suma un punto (el contador se despliega en la ventana) y el caramelo desaparece del mundo de la simulación, si el caramelo toca el piso, desaparece del mundo de la simulación.

1. **Sketch uno**: ahí se **obtiene** los cambios de los controladores y **manda** los valores obtenidos al sketch dos. Tiene que contener controladores (cada controlador manda un mensaje/el valor nuevo cuando cambia su estado):
    - Para subir o disminuir la velocidad de la caída de los caramelos (por ejemplo en slider o dos botones)
    - Para subir o disminuir la frecuencia con la cual aparecen caramelos nuevos
    - Para resetear el contador de caramelos agarrados
2. **Sketch dos**: ahí se **recibe** los valores y **actualiza** la visualización y la simulación física con los nuevos valores recibidos.

Fecha de entrega: **viernes** 10.04.20 hasta las 23:59.

Como me lo imagino

Puntaje: 11