Physics based animations

- Julien Vulliet (Vux)
- Natan Sinigaglia (Dottore)
Physics based animations

- Introduction
- Terminology
- Box2d Introduction
- Box2d in Depth
- Pratical exercises
Physics based animations

Introduction
Physics based animations

Introduction

Why using a physics engine?

- Realistic Motion
- Objects Interaction (Collisions)
- Efficient handling of Multiple Objects
- Create constraints between Objects
Physical based animations

Terminology

- Time steps
- ABB, OBB
- Shape, Body, Joint
- Broadphase, Narrowphase
- Sleeping Bodies
- Tunneling, CCD, TOI
Box2d

http://www.box2d.org

- Box2d is an open source 2D physics engine, primarily designed for games.
- Written by Erin Catto (physics developer at Blizzard)
- Licensed under zlib: http://en.wikipedia.org/wiki/Zlib_License
- Rigid body engine. Object geometry can't be altered.
- Contains most modern physics engine features (Broadphase, CCD, Compound Bodies, Collision filtering, Joints).
- Initially written in C++, but ports are available in Flash, Java, Objective C, C#...
Physics based animations

Box2d for VVVV

- Box2d plugin is a Wrapper against Box2d (c++ version). It's not a port.
- For license, go to plugins/licenses/license-box2d.txt
- Uses a top-bottom/top-bottom approach for object creation/interaction
- Wraps pretty much all box2d features (apart the few which requires code)

IMPORTANT NOTE:

As box2d (and any physics engine) will struggle with small values, it is really not recommended to use objects of an area of less than 0.1

This will cause some errors (due to floating points), with very low inertia for example (no restitution)

Good practice is to have a projection with a uniform scale of 0.1 so we can used decently sized objects.
Physics based animations

Box2d in depth
World

This is the core object for box2d nodes.

It handles:

- Physics simulation
- Object management (Shape updates/Destruction)
- Collision checks
- Prepare objects so their data can be retrieved.
Physics based animations

World

-- These are the world boundaries, any object going outside this bounds will become invalid. Modifying those values will reset the world.

-- Global gravity Force

-- Time step for each physics step, it is better to keep it constant, do not attach to framerate

-- The more iterations the more accurate the simulation, but of course affects performances

-- Allows the engine to put body to sleep at global level

-- Start/Stop simulating

-- Destroys all objects in the world and recreates a blank one
Physics based animations

World

- World instance, required for nodes creating objects
- How many active controllers the world contains
- Just a flag to notify if world is valid, mainly boundaries.
- This is a default body (static), with no shapes attached. Use for creating joints in world space, or you can attach static shapes
- All body instances active in the world
- All joint instances active in the world
- Bang to indicate that the world has been reset, generally use it to recreate default world geometry (Walls for example)
Physics based animations

Primitives

- Circle
  - Radius
  - Position XY

- Box
  - Size XY
  - Angle

- Polygon
  - Vertices XY
  - Vertices Count per shape
  - Loop

Array of segments. Density has not impact as a segment has no mass by definition.
Physics based animations

Primitives

- **Density**: 1.0000
  - Density of the shape to calculate the mass. A density of 0 will make the body static.

- **Friction**: 0.3000
  - Friction amount, when two objects "slide" on each other. 0 means no friction, 1 means full friction.

- **Restitution**: 0.0000
  - Restitution amount, this is the amount of "bounce" (collision response) when two objects collide. It is possible to have more than 1, which is physically impossible, but can have uses for games. Also when two objects collide, the maximum restitution between both is taken into account.

- **Is Sensor**:
  - Is the shape a collision sensor?

- **Group Index**: 0
  - Used for collision filtering.

- **Custom**:
  - Just a standard string, so you can attach anything you want.
Physics based animations

Creating Bodies

- World instance to create the body to
- Body geometry definition
- Body position in world space
- Body initial rotation
- Body initial linear velocity
- Body initial angular velocity
- Bang here to create the body

CreateBody

- Instance of the created body, only alive for the frame it has been created.
- Can be used to create joint/assign controller right upon creation

Body Instance

- Unique Id for the created body

Body Id
Physics based animations

Creating Bodies

- Linear Damping
  - 0.0000
  - Damping ratio. 0 is no damping. 1 is Critical damping

- Angular Damping
  - 0.0000

- Fixed rotation
  - If you turn this flag on, the body will always keep the same angle

- Is Bullet
  - Enable CCD for fast moving objects. This avoids tunneling, but uses performances

- Body custom
  - Identical to shape custom pin but at body level
Physics based animations

Retrieve Body information
Physics based animations

Retrieve Body information

- Is Sleeping: If sleeping, this body is not simulating
- Is Frozen: Body will be set to frozen when it goes outside world boundaries, should be destroyed
- Mass: 0.0000
- Inertia: 0.0000
- Custom: Custom value that was set on creation
Physics based animations

Retrieve Shapes information

--- Shape instances

- Circles
- Polygons
- Edge Chains

- Radius
- Centers XY
- Vertices XY
- Vertices Count

- World coordinates/Coordinates relative to parent body
- Local Coordinates
- Automatically close the polygon by adding an extra point

- Upon creation, each edge chain creates one shape per segment
Physics based animations

Retrieve Shapes information

-All the pins above are common to every single shape
Physics based animations

Find objects (AABBQuery)

-AABBQuery performs a query from the supplied ABB to shapes ABB. So positive result does not necessarily mean that the shape overlaps.

-As AABBQuery performs ABB checks, it can be easily done in the Broadphase, hence it’s a very fast way to find objects which might intersect.

-Query is done against shapes, but as very often we want to interact with its parent body, it is returned as a convenience.
Physics based animations

Find objects (Ray Casts)
Physics based animations

Find objects (Test Points)
Physics based animations

Update Object properties

- UpdateBody
- DestroyBody
- UpdateShape
- ScaleShape
Physics based animations

This node is fully spreadable, so you can set multiple bodies at the same time upon conditions.

Putting a body to sleep will also Set all velocity components to 0.

Setting new position is generally not recommended, as it can break the simulation. More efficient methods for moving objects will be presented later.
This node is also fully spreadable, so you can set multiple bodies at the same time upon conditions.

Technical note:

Internally and for stability purposes. A body is not immediately destroyed. Instead a flag tells it has to be destroyed instead.

The world node, before to execute time step, will check for all bodies marked for deletion and destroys them before the time step.

If we immediately destroy the body, we could have a case when we try to destory/update properties in the same frame, which would result in a crash.
This node is also fully spreadable, so you can set multiple shapes at the same time upon conditions.

Setting a density to zero/non zero can switch the body from static/dynamic.
Physics based animations

- This node is also fully spreadable, so you can scale multiple shapes.
- Incremental makes the new scale calculations as follows: if turned off, new scale = current size * scale factor. If turned on, new scale = current size + scale factor.
- Scale shape is a convenience. Eg: internally shape is destroyed/recreated while keeping same other properties.
- It is not possible to scale by dimension for boxes (unique interface for every shape type)
Physics based animations

Collisions – Retrieve Contact informations

As for AABBQuery, only shape collide, but parent body is returned as convenience.

A contact count is provided as two shapes can have multiple contact points (two boxes with the same angle for example).
Physics based animations

Collisions – Sensors and Filters

Common sensor uses: Proximity detection, Check if an object is in an area, Bullets in tank game (we don't want the object to bounce, just know when if reaches another player)
- Position XY is in Body space, so a position of (0,0) will apply the force to the center of the object.
- Applying a force non central to the object will also impact torque (force on object rotation)
- ApplyForce impacts on the body acceleration.
Physics based animations

Body Interactions : Impulse

- Position XY is in Body space, so a position of (0,0) will apply the impulse to the center of the object.
- Applying an impulse non central to the object will also impact torque (force on object rotation)
- ApplyImpulse directly impacts on the body velocity, so very small values can have so drastic changes
Physics based animations

Body Interactions : Torque

- Torque impacts object rotation (not any linear vector parameters)
- Torque can be positive or negative of course.
Physics based animations

Compound Bodies : Create Shape

- Bodies can contain several shapes
- Collision between shapes belonging to the same body will always be disabled
Physics based animations

Compound Bodies : Destroy Shape

- Please note that when we destroy shapes, some bodies can end up with no shape at all. This need to be taken into account, but is up to the user.
Joints

Joints allow you to create constraints between bodies.

Generally they are used to limit the degree of freedom between objects.

For example, think of two objects attached by a rope.

Joints can provide different features:

- **Limits**: This allows you to control the degree of freedom.
- **Motors**: You can drive the joint at a specified speed.
- **Connection point**: From a joint to an object is called an Anchor.
- **You can either connect a body to another one. For some joints you might only want to connect it to an arbitrary point in space. In this case, we can use the ground output from the world node.**
- **Most joints also provide a flag indicating if connected bodies should collide with each other.**
Joints

Before to see every joints in details, here are the two nodes allowing to interact with existing ones:

- Destroy joint works exactly the same way as destroy body.
- If a body is destroyed, all joints attached to it are also destroyed.
Physics based animations

Joints: Distance

This creates a constraint so objects have to stay at a fixed distance to each other.

Frequency and Damping ratio are used to "loosen" the constraint, and make it behave more like a spring.

Frequency is measured in Hertz (oscillator)

Damping ratio says how "thick" is the spring (0 means fully rigid, 1 fully loose)

Distance joint allows you to attach an object to the ground
Physics based animations

Joints : Revolute

This creates a common anchor point, limiting objects rotation. This is like a "pin"

- As we pin both objects, only one anchor is necessary.
- We can specify rotation limits, so this is useful to create objects like ragdolls.
- Motor can be used for this type of joints
Joints : Prismatic

This works like revolute, but creates a distance limit, and also prevents any relative rotation.

- As we pin both objects, only one anchor is necessary.
- We can specify translation limit (relative), so make sure lower is negative and upper is positive.
- Motor can be used for this type of joints.
Objects Controllers

-A controller is an instance to seamlessly process a number of bodies in a certain way.

We can assign/unassign bodies to controllers at will, bodies can be attached to multiple controllers at once.

It's an easy way to manage a lot of objects at once.

-Every controller node also contains a "Clear" pin, to remove all bodies that this controller contains at once, provided as a convenience.

-Every controller parameter can be of course changed in real time
Assigning/Unassigning bodies to controllers is pretty straightforward, just using the two node (please note they are fully spreadable).

The engine will take care that no multiple assignments have been done, and will take care of body destruction.
Objects Controllers : Constant force

Constant force applies a force to each body it contains. It's equivalent to using ApplyForce every frame to a set of bodies.
Objects Controllers : Constant acceleration

Constant acceleration is modifying object acceleration. This is how gravity works eg:

Body.Velocity = Body.Velocity + DeltaTime * A
Objects Controllers : Gravity

Unlike ConstantAcceleration, this controller does not apply a uniform gravity.
Each body contained in this controller will exert a gravitational force against each other.
Gravitational force depends on objects distance between each other and their mass.
Force can be positive (attractor) or negative (repulsor)
Objects Controllers: Buoyancy

This simulates objects moving in water. See usage example.
More on Joints: Mouse Joints

Mouse joint is a simple facility which was used to facilitate debugging, but turned out into a nice feature. We can assign a position and move an object to that position. Internally this is like a "moveable" distance joint.

As a specific case, this joint only has one body.
Physics based animations

More on Joints : Motors

As some joints allow motor, we can also update the motor speed.

Engine takes care if a joint doesn't support motor, but it is recommended to do some filtering beforehand.
Physics based animations

Practical box2d cases
Physics based animations

Retrieve objects (Advanced)

Retrieving objects can be done applying different techniques:

- We can use the body unique Id: in this case, we simply need to use GetSlice (Node)

- One more advanced technique comes by using the Custom pin. We can assign an arbitrary string to an object, then use Sift (String) + GetSlice (Node) to filter object instances.

- If we need to store more than a single information inside the custom pin, we can use either semicolon separated strings, or xml data (make sure all objects create an xml with the same structure to simplify)
Texturing/Coloring

Coloring/Texturing objects can be done multiple ways. Both share the same concepts so any technique applies to both.

Case 1: Non deterministic

In that case we only need the object not to change color, it will remain the same all it lifetime, and we do not mind which color.

Simple way is to create a simple static color pool (Random color spread for example), then use the body/shape id with getslice color.

As the object id is constant through all the object lifetime, the object will never change color (unless we change the color pool)
Texturing/Coloring

Case 2: Deterministic

In that case we only need the object not to change color, it will remain the same all its lifetime, but we want to specify the color upon creation.

- Technique 1: Store color as Hex in custom pin. This works and is very easy, but this will not apply to texturing. It also of course allows to change color using UpdateBody.

- Technique 2: Create a color pool as previously, and store the color slice index in the custom pin. This will also apply easily to texturing.

- Technique 3: Create a color pool using a pair structure (code/color). Code is just a string to represent the color. We can then use Sift (String) and GetSlice (Color).
Device/User interactions

Box2d offers a lot of features to interact with an existing world. As there is a lot of different devices, doing a generic system is almost impossible.

Altough some devices are more adapted for some interactions than others.

We will speak about device types and discuss different ways of integrations.
Physics based animations

Device/User interactions : MIDI

They generally offer a standard interface (notes/controls), so building a midi system is generally quite straightforward.

Some implementations cases:

- Create objects using midi notes (Keyboards/Pads).
- Update gravity/controller data (Any 2d vector works pretty well with any device which has an x/y slider), we can also easily apply factors using sliders/knobs. Enabling/disabling controllers can also easily be achieved using toggles.
Physics based animations

Device/User interactions : Accelerometers

Some examples include:
- Wii remotes
- Iphones

Some general good uses for them:
- controllers (update forces)
- Applyforce (so we can interact with single objects)
Multi touch integration with box2d is a vast subject. Some different possible interactions are:

- Object Creation/Destruction
- Move objects (Mouse joints/Applyforce works pretty well)

The main difficulty with multitouch is we often need to track if an object is assigned to a touch. This can be easily achieved by storing touch id in the custom object pin.

Simple ways of processing hit tests can be done using AABBQuery, HitTest (polygon/quad), then processing the data.

We also need some modules to tell us when a touch appears/disappear.
Physics based animations

Build complex models

Once we get past the simple cool examples, we then often want to create more complex models. This requires more efforts and forward design thinking to keep patching as clean as possible.

Some general considerations:

- Know your tools: The key to build a model is to know what feature to use in what case. Some joints type work better than other in certain cases for examples. So it is important to test as many node/parameter as possible to be able to know which one to use.

- Build a framework: Generally a model will have 3 main parts, which are initial model creation, interactions, display. Having those in separate subpatches helps a lot once the model starts to be quite big.

- Try to keep interactions as device independent as possible: Having some hardcoded midi in the interaction patch is not a good idea. So having subpatches with pins for every interactions will make things easier.

- Build a lot of generic modules: modules like getbodybycustom are fairly general purpose and helps a lot.

- For evolving worlds/games, nodes like automata will simplify your life greatly.
Box2d can easily adapt to projection mapping. Techniques to build a model depends greatly on the complexity of the shape.

For simple models, boxes/circles/polygons will offer nice features. Building a simple "drawing tool" will make you life slightly easier and save your time (common storage can be xml/sqlite databases)

For more complex models edge chains will come to mind. Image processing will often be needed in that case. Please also note that edges are expensive so you will want to simplify your polygons (to keep a good performance/accuracy balance).
Building box2d models in a boygroup environment is quite straightforward.

Generally a good practice is to separate the server/client engine that way:

- Server: World, object creation, interactions
- Client: Keep getbodydetails in the server, then send coordinates to clients, and process display

This offers nice performances as the server will take care of all the simulation.

We can do custom filtering in the clients to avoid drawing objects which doesn't belong to the view.
Development of box2d is still ongoing, and some new features are already in development.

- No more world/shape limit
- Improved AABB/Raycasts
- Enable/Disable bodies on the fly
- New joints (Friction/Weld/Rope)
- Edge chain revamp (bi directional by default)
- Compound shape system revamp (ability to group shapes before to add them to a body)
- Fully integrated with new dynamic plugins, so you can access object properties, apply interactions in your own code. (You can write your own controller for example, create some complex collision filtering logic...). That also means that it will be beta24 required.
- Simplified retrieval system. More custom nodes to simplify patching, major changes in shapes retrieval.
- Features request welcomed :)