MonocleGL Physics Documentation

Description:

MonocleGL uses a popular 2D physics engine called chipmunk to efficiently simulate real world physics on pretty much any device. Feel free to view some the samples at http://code.google.com/p/chipmunk-physics/ to get a good feel of the power of this engine. In monocleGL physics take effect only on nodes that are not being dragged or tweened. While dragging/tweening the physics engine is updated of the node position but it will not apply any physics magic to the node. Then when control is returned to the physics engine everything is where is suppose to be and the physics magic can continue as normal.

To initialize the physics:

To use the physics in a demo you must first create the physics object. You may also call physics on each node which negates the need to pass in the nodes ID. To create a physics object pass the plugin reference like so:

```
var physics = new Physics(p);
```

Function setEnvironment:

Set environment will set global settings for the whole physics space. By default gravity is set to point 200 units down but with this function it can be pointed in any direction at any magnitude. For mobile devices this function can link gravity to the accelerometer. Unfnortunately since the desktop does not have accelerometer this can not be used in demos.

```
physics.setEnvironment(command);
```

setEnvironment commands:

The setEnviroment command use the string format “variable:value,variable:value,...” where variable is a command and value is the desire value the physics engine will use. As an example to set gravity to be 200 units up use the command “gravity_y:-200”. Here’s a list of command and members for setEnvironment:
gravity_x - (number default 0) The gravity force in the x-axis.
gravity_y - (number default -200) The gravity force in the y-axis.
accelerometer - (boolean default false) If true then the accelerometer control the gravity.
pause - (boolean default false) If true then the physics engine will stop updating.

command - This is the command string full of commands and the members to be affected. (See below for more detailed explanation of the commands)

Function addPhysics:

Add physics is the main function for giving node physics data. The the plugin will parse the passed command string so white spaces and case do not matter. Once addPhysics can be called multiple times on the object. If physic information already exists then the plugin use those values as defaults and uses the extra commands sent in the command string to overwrite them.

```
node.addPhysics(command);
physics.addPhysics(node.id, command);
```

node - This is the object to have physics added to it.
command - This is the command string full of commands and the members to be affected. (See below for more detailed explanation of the commands)

addPhysics commands:

The add physics command use the string format “variable:value,variable:value,...” where variable is a command and value is the desire value the physics engine will use. As an example set the mass to 30, elasticity to 0.9 use the command “mass:30,elasticity:0.9”. Here’s a list of command and members for addPhysics:

mass - (number default 100.0) How heavy the object is.
elasticity - (number default 0.8) How elastic the object is, 1.0 is the “perfect” bounce.
friction - (number default 0.5) How much the object resists sliding on another object.
torque - (number default 0.0) The angular force on a rotating object.
velocity_x - (number default 0.0) The x-axis linear force on an object.
velocity_y - (number default 0.0) The y-axis linear force on an object.
sensor - (boolean default false) If true the object will only trigger callbacks.
static - (boolean default false) If true the object will not have gravity affect it.

type - (string default circle) the type of shape the physics body should be.
- circle - (default) Uses the objects width as the radius of the object (faster)
- segment - Draws a line from the object (x,y) to (x+width,y+height) (fast)
- box - Draw a box around the objects height and width (slower)

Function removePhysics:

Remove physics will search for any physics objects that are attached to the passed node. This command completely removes all physics data from the node but the node will still exist in the system.

```
node.removePhysics();
physics.removePhysics(node.id);
```

node - This is the object which will have it's physics removed from.

Function addSegment:

Add segment is used to add static line segments to the physics space. Typically this is used to create a border around the demo to keep all the objects to stop falling to infinity. Because these line segments are static they don't eat up much cpu so are a great way to build up more complex scenes.

```
physics.addSegment(name, x1, y1, x2, y2);
```

name - This is the name of the line segment, can share names with other segments
x1 - The x axis of the start point
y1 - The y axis of the start point
x2 - The x axis of the end point
y2 - The y axis of the end point

Function removeSegment:

Remove segment from the physics space. This function goes though all the segments and remove any segment's with matching names.

```
physics.removeSegment(name);
```

name - This is the name of the line segment that will be removed

Function addJoint:

Add joint is use to place constraints on two objects. The type of joint will greatly affect how the object will react some it would be worth while to check out this video of

chipmunk joints in action. http://youtu.be/ZgJJZTS0aMM (Note: a value of “NULL” can be passed for one of the nodes. This means the non null node will be attached to a static body.)

```
nodeA.addJoint(nodeB.id, command);
physics.addJoint(nodeA.id, nodeB.id, commands);
```

nodeA - This is the first object to have a joint added to it.
nodeB - This is the second object to have a joint added to it.
command - This is the command string full of commands and the members to be affected. (See below for more detailed explanation of the commands)

addJoint commands:

The add physics command use the string format “variable:value,variable:value,...” where variable is a command and value is the desire value the physics engine will use. As an example to create a slide joint to has a minimum distance of 20 and a maximum distance of 60 use the command "type:slide,min:20,max:60". For the joints the type of the joint dictates what parameters you will need. Here’s a list of command and members for addJoint:

Pin joint (“type:pin”):
- anchor_ax - (number default 0.0) Anchor point x-axis on object A, 0.0 for center.
- anchor_ay - (number default 0.0) Anchor point y-axis on object A, 0.0 for center.
- anchor_bx - (number default 0.0) Anchor point x-axis on object B, 0.0 for center.
- anchor_by - (number default 0.0) Anchor point y-axis on object B, 0.0 for center.

Slide joint (“type:slide”):
- anchor_ax - (number default 0.0) Anchor point x-axis on object A, 0.0 for center.
- anchor_ay - (number default 0.0) Anchor point y-axis on object A, 0.0 for center.
- anchor_bx - (number default 0.0) Anchor point x-axis on object B, 0.0 for center.
- anchor_by - (number default 0.0) Anchor point y-axis on object B, 0.0 for center.
- min - (number default 0.0) Minimum allowable distance between both objects.
- max - (number default 1.0) Maximum allowable distance between both objects.

Pivot joint (“type:pivot”):
- anchor_ax - (number default 0.0) Anchor point x-axis in the global coordinates.
- anchor_ay - (number default 0.0) Anchor point y-axis in the global coordinates.
- (Note: Pivot is different from any other joint because it requires a global point for the anchor where as all the other joints use local references.)

Groove Joint (“type:groove”):

groove_ax - (number default 0.0) Groove start point x-axis on object A.
groove_ay - (number default 0.0) Groove start point y-axis on object A.
groove_bx - (number default 0.0) Groove end point x-axis on object A.
groove_by - (number default 0.0) Groove end point y-axis on object A.
anchor_bx - (number default 0.0) Anchor point x-axis on object B, 0.0 for center.
anchor_by - (number default 0.0) Anchor point y-axis on object B, 0.0 for center.

Damping spring joint (“type:spring”):

anchor_ax - (number default 0.0) Anchor point x-axis on object A, 0.0 for center.
anchor_ay - (number default 0.0) Anchor point y-axis on object A, 0.0 for center.
anchor_bx - (number default 0.0) Anchor point x-axis on object B, 0.0 for center.
anchor_by - (number default 0.0) Anchor point y-axis on object B, 0.0 for center.
rest_length - (number default 2.5) The length of the spring when it’s at rest.
stiffness - (number default 5.0) How much the spring will want to return to it’s resting length.
damping - (number default 0.05) How much the spring dampers movement.

Damping rotatory spring joint (“type:rotary_spring”):

rest_angle - (number default 0.0) The length of the spring when it’s at rest.
stiffness - (number default 5.0) How much the spring will want to return to it’s resting angle.
damping - (number default 0.05) How much the spring dampers movement.

Damping rotary limit joint (“type:rotary_limit”):

min - (number default 0.0) The minimum angle difference in radians.
max - (number default 1.0) The maximum angle difference in radians.

Ratchet joint (“type:ratchet”):

phase - (number default 0.0) The starting phase of the joint.
ratchet - (number default 0.0) The step amount for each ratchet in radians.

Gear joint (“type:gear”):

phase - (number default 0.0) The starting phase of the joint.
ratio - (number default 1.0) The gear ratio size between object a to object b.

Motor joint (“type:motor”):

rate - (number default 0.0) The rotational speed of the motor in radians/s.

Function removeJoint:

Remove joint searches the joint list for a joint that matches both nodes passed in. If a match is found the joint is removed from the system.

```
nodeA.removeJoint(nodeB.id);
physics.removeJoint(nodeA.id, nodeB.id);
```

nodeA - This is the first object to have the joint removed.
nodeB - This is the second object to have the joint removed.

Function addCollision:

Many collisions will occur in any physics simulation. If we're interested in triggering a callback when a collision occurs you must specify the two that will trigger an event and the function to called when it does happen. Currently collision can only be detected but in the future look for the ability to detect separations too.

```
nodeA.addCollision(nodeB.id, object, functor);
physics.addCollision(nodeA.id, nodeB.id, object, functor);
```

nodeA - This is the first object to have a collision added to it.
nodeB - This is the second object to have a collision added to it.
object - The JavaScript object to call the function on.
functor - The name of the function to call.

Function removeCollision:

Remove collision will remove the filter that trigger collision events. You must pass it the two nodes that where registered before.

```
nodeA.removeCollision(nodeB.id);
physics.removeCollision(nodeA.id, nodeB.id);
```

nodeA - This is the first object to have the collision removed.
nodeB - This is the second object to have the collision removed.

Function removeAllPhysics

Clean up function that removes all the physics data and frees everything.

```
physics.removeAllPhysics();
```