Joints

What is a joint?

Joints constraint two bodies to the world or between them. One of these bodies can be static. If both bodies are static, the joint is ignored.

They are created using the same procedure as the body and fixture creations, with a joint definition and the corresponding factory.

```
joint = myWorld->CreateJoint(&jointDef);
//... do stuff ...
myWorld->DestroyJoint(joint);
```

When a body is deleted, all joints attached to this body are deleted as well.

Types of joints

Which joints you will used in order to reproduce the flippers and the kicker? Are there more elements in your pinball that need joints?

- ✗ Distance
- ✗ Revolute
- ✗ Prismatic
- ✗ Pulley
- ✗ Gear
- ✗ Mouse
- ✗ Wheel
- ✗ Weld
- ✗ Rope
- ✗ Friction
- ✗ Motor

Distance joint

Live Demo

The most simple one! Keeps two bodies at a constant distance.

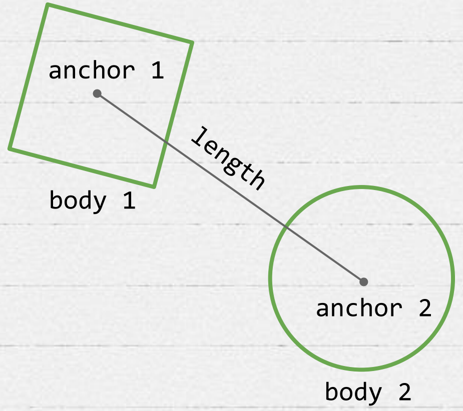
```
b2DistanceJointDef jointDef;
jointDef.Initialize(myBodyA, myBodyB, worldAnchorOnBodyA,
worldAnchorOnBodyB);
```

We can allow or block the collision between them:

```
jointDef.collideConnected = true;
```

Can be transformed into a spring tweaking two parameters on the joint definition:

```
jointDef.frequencyHz = // Less than half FPS
jointDef.dampingRatio = // 0...1 where 1 is no damping
```



Revolute joint (Hinge)

Live Demo

Live Demo

Two bodies connected by an anchor point, that determines a joint angle:

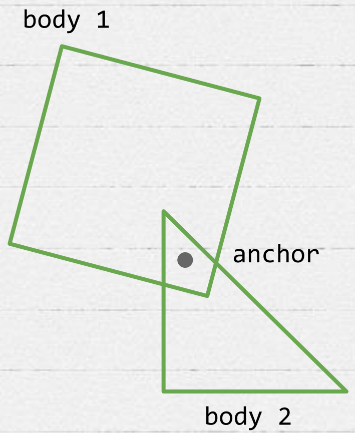
```
b2RevoluteJointDef jointDef;
jointDef.Initialize(myBodyA, myBodyB, myBodyA->GetWorldCenter());
```

Accepts minimum and maximum angles

```
jointDef.enableLimit = true;
jointDef.lowerAngle = -0.5f * b2_pi; // -90 degrees
jointDef.upperAngle = 0.25f * b2_pi; // 45 degrees
```

And the possibility to add a motor (torque) to spin one of the bodies.

```
jointDef.enableMotor = true;
jointDef.maxMotorTorque = 10.0f;
jointDef.motorSpeed = 0.0f;
```



Prismatic joint

Live Demo

Allows relative translation between objects

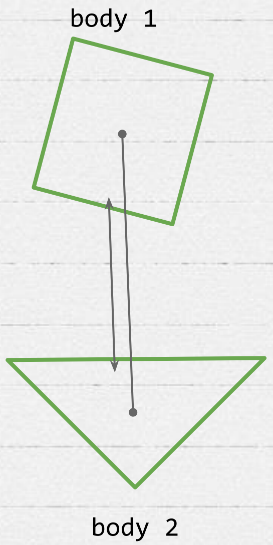
```
b2PrismaticJointDef jointDef; b2Vec2 worldAxis(1.0f, 0.0f);
jointDef.Initialize(myBodyA, myBodyB, myBodyA->GetWorldCenter(),
worldAxis);
```

Accepts minimum and maximum displacement

```
jointDef.enableLimit = true;
jointDef.lowerTranslation = -5.0f;
jointDef.upperTranslation = 2.5f;
```

And the possibility to add a motor (force)

```
jointDef.enableMotor = true;
jointDef.maxMotorForce = 1.0f;
jointDef.motorSpeed = 0.0f;
```

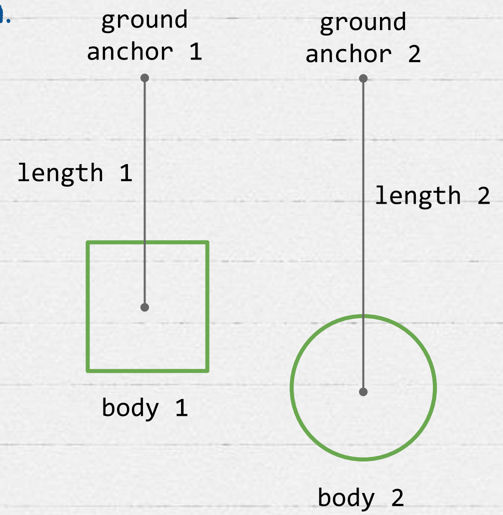


Pulley joint

Live Video

A pulley is used to create an idealized pulley. The pulley connects two bodies to ground and to each other. As one body goes up, the other goes down.

```
b2Vec2 anchor1 = myBody1->GetWorldCenter();
b2Vec2 anchor2 = myBody2->GetWorldCenter();
b2Vec2 groundAnchor1(p1.x, p1.y + 10.0f);
b2Vec2 groundAnchor2(p2.x, p2.y + 12.0f);
float32 ratio = 1.0f;
b2PulleyJointDef jointDef;
jointDef.Initialize(myBody1, myBody2, groundAnchor1,
groundAnchor2, anchor1,
anchor2, ratio);
```



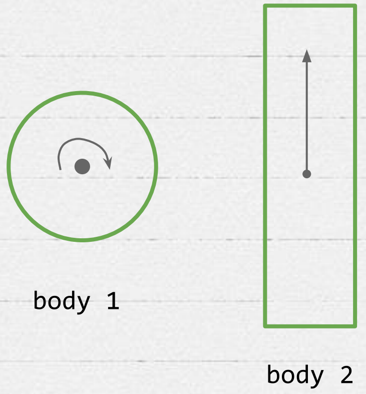
The ratio can be modified.This causes one side of the pulley to extend faster than the other.

Gear joint

Live Demo

Useful to create sophisticated mechanical contraption systems.

```
b2GearJointDef jointDef; // Only accepts revolute and prismatic joints
jointDef.bodyA = myBodyA;
jointDef.bodyB = myBodyB;
jointDef.joint1 = myRevoluteJoint;
jointDef.joint2 = myPrismaticJoint;
jointDef.ratio = 2.0f * b2_pi / myLength;
```



It accepts ratios too!

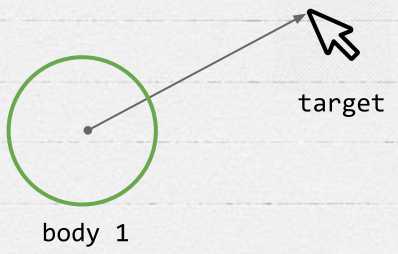
Caution! Gear joint contains other joint references (revolute or prismatic), so when removing, delete first the gear joint.

Mouse joint

Live Demo

Only used for debugging purposes! It makes a body to track a specific world point:

```
b2MouseJointDef def;
def.bodyA = ground;
def.bodyB = body_clicked;
def.target = mouse_position;
def.dampingRatio = 0.5f;
def.frequencyHz = 2.0f;
def.maxForce = 100.0f * body_clicked->GetMass();
mouse_joint = (b2MouseJoint*) world->CreateJoint(&def);
```

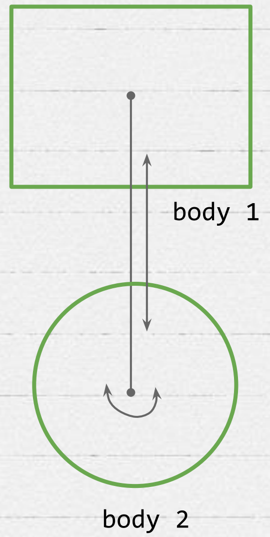


Wheel joint

Live Demo

Useful to simulate car wheels with suspension.

```
b2WheelJointDef jointDef;
jointDef.localAnchorA.SetZero();
jointDef.localAnchorB.SetZero();
jointDef.localAxisA.Set(1.0f, 0.0f);
jointDef.enableMotor = false;
jointDef.maxMotorTorque = 0.0f;
jointDef.motorSpeed = 0.0f;
jointDef.frequencyHz = 2.0f;
jointDef.dampingRatio = 0.7f;
```



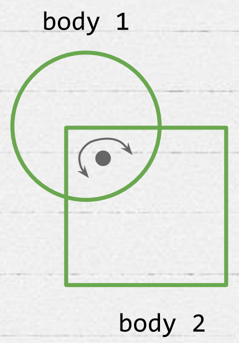
We restrict body2 to a line of body1. It also accepts some spring-like behaviour (like car suspension) and a motor

Weld joint

Live Demo

It will constrain all relative motion. The result is similar of having a body with different fixtures. But with weld joint, we can add spring parameters

```
b2WeldJointDef jointDef;
jointDef.localAnchorA.Set(0.0f, 0.0f);
jointDef.localAnchorB.Set(0.0f, 0.0f);
jointDef.referenceAngle = 0.0f;
jointDef.frequencyHz = 0.0f;
jointDef.dampingRatio = 0.0f
```

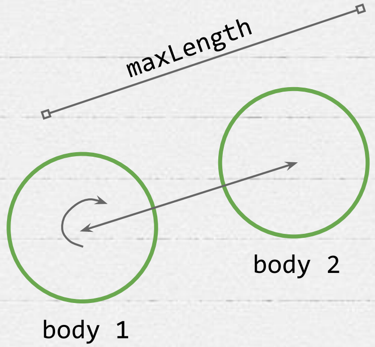


Rope joint

Live Demo

The rope joint restricts the maximum distance between two points. It's similar to distance joint but there is no minimum distance. Useful to prevent stretching.

```
b2RopeJointDef jointDef;
jointDef.localAnchorA.Set(-1.0f, 0.0f);
jointDef.localAnchorB.Set(1.0f, 0.0f);
jointDef.maxLength = 1.5f;
```

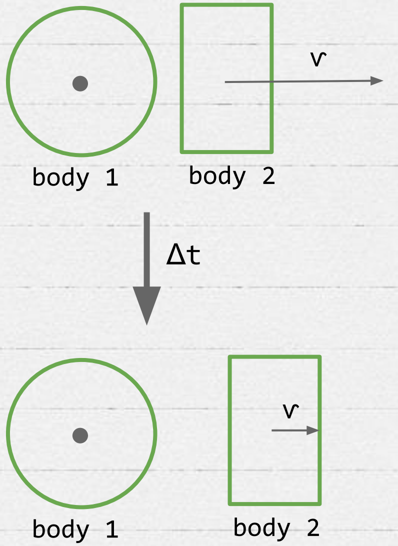


Friction joint

Live Demo

The friction joint is used for top-down friction. It attempts to drive the relative motion between the bodies to zero. The maximum force and torque members are used to limit the rate at which the motion is driven to zero.

```
b2WeldJointDef jointDef;
jointDef.localAnchorA.SetZero();
jointDef.localAnchorB.SetZero();
jointDef.maxForce = 0.0f;
jointDef.maxTorque = 0.0f;
```

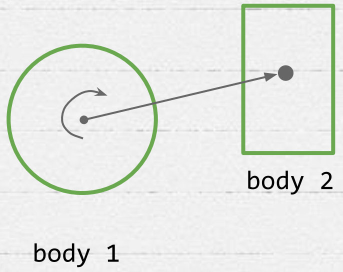


Motor joint

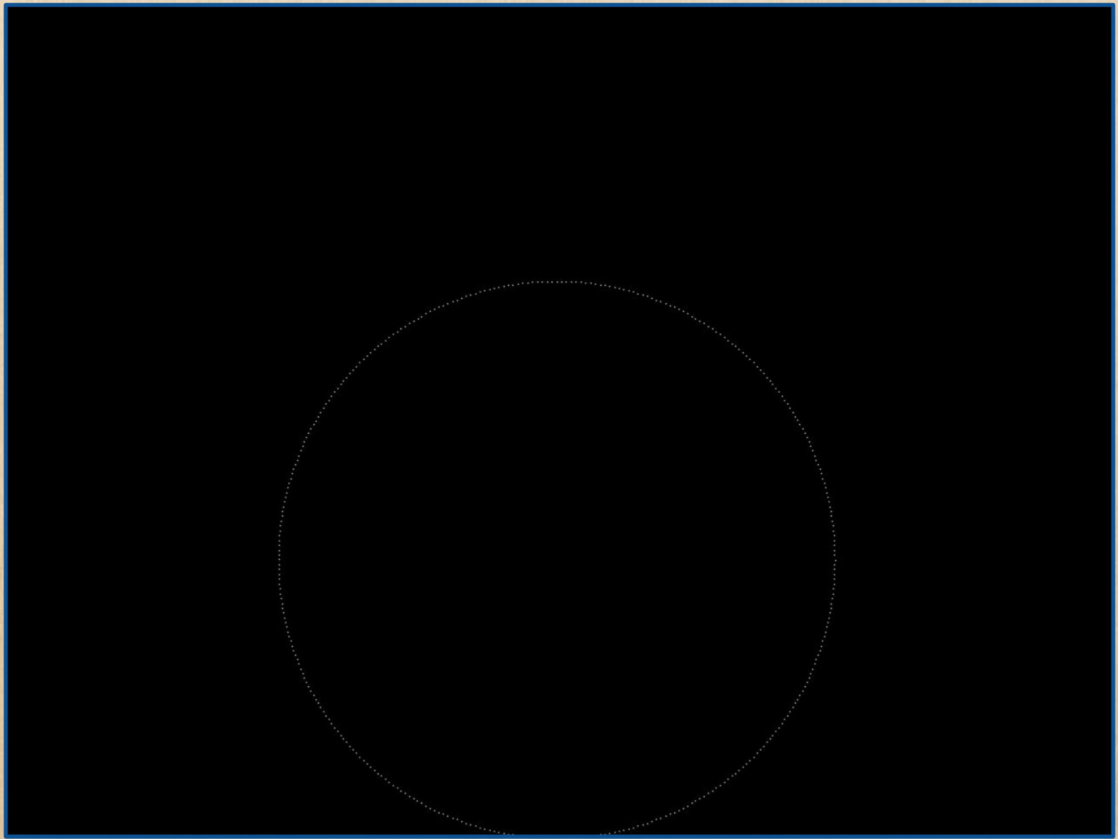
Live Demo

A motor joint lets you control the motion of a body by specifying target position and rotation offsets.

```
b2MotorJointDef jointDef;
jointDef.linearOffset.SetZero();
jointDef.angularOffset = 0.0f;
jointDef.maxForce = 1.0f;
jointDef.maxTorque = 1.0f;
jointDef.correctionFactor = 0.3f;
```



Simply pushes a *body 1* to *body 2* position and rotation. For instance, it could simulate a magnet.



Our goal

YOUR TURN !

Implementation of a mouse joint

It will be very useful to move our ball around the level. To do that, we will follow the next steps:

1. If we clicked on a shape, we will create a mouse joint
2. If we keep pressing the mouse, we will update the target joint position
3. If we release the mouse, the joint will be destroyed.

TODO Nº 1

```
// TODO 1: If mouse button 1 is pressed ...
// if (App->input->GetMouseButton(SDL_BUTTON_LEFT) == KEY_DOWN)
// test if the current body contains mouse position
```

- ✗ When mouse clicking, check whether the cursor is inside one shape
- ✗ If that's true, keep that body on a pointer and stop the iteration
- ✗ At the end, you will have a pointer with the body clicked or NULL otherwise

TODO No 2

```
// If a body was selected we will attach a mouse joint to it
// so we can pull it around
// TODO 2: If a body was selected, create a mouse joint
// using mouse_joint class property
```

Now, create a mouse joint for this selected body:

```
b2MouseJointDef def;
def.bodyA = ground;
def.bodyB = body_clicked;
def.target = mouse_position;
def.dampingRatio = 0.5f;
def.frequencyHz = 2.0f;
def.maxForce = 100.0f * body_clicked->GetMass();
mouse_joint = (b2MouseJoint*) world->CreateJoint(&def);
```

TODO Nº 3

```
// TODO 3: If the player keeps pressing the mouse button, update
// target position and draw a red line between both anchor points
```

- ✗ If user keeps pressing mouse button, we will update target joint position
- ✗ In order to clearly see the joint, we will draw a line between the selected body and cursor position with App->renderer->DrawLine

TODO Nº 4

```
// TODO 4: If the player releases the mouse button, destroy the joint
```

Remember to *nullify* all pointers used in this feature in order to avoid **dangling pointers**.

Next Week . . .

Pinball Delivery!