Raycasting

Raycasting is part of the underlying physics system that Orx uses.

What is Raycasting?

Raycasting is simply the act of sending a ray from one point to another point, and returning any object that the ray passes through.

What can Raycasting be used for?

Raycasting can be very useful for many gaming scenarios. Here are some examples:

1. Sending rays to determine if a guard can see the hero sneaking around
2. Sending rays to determine what surface object a hero might be about to land on, and use that information to switch to the correct animation.
3. Searching a room full of wall to determine what area needs to be lit up.

Requirements and Rules

In order to successfully use raycasting, there is a minimum number of things that you need:

1. Objects that are to be searched using rays need to have a physics body.
2. Those objects also must have their `SelfFlags` and `CheckMask` properties defined.
3. Two vector points are required to make a line: from point A to point B.
4. Two two points are not allowed to be the same, or very close together, or your game will assert.
5. Point A is not allowed to begin within an object. Otherwise the object won't be found.
6. orxObject_Raycast is used to make the raycast.

Testing with the Bounce Demo

Before trying raycasting in our code, let's first play around with it in the Bounce Demo. The Bounce Demo is available with all compiled orx libraries.

Enable the physics debugging in the Bounce Demo by using the command, or by adding to the /orx/code/bin/Bounce.ini file:

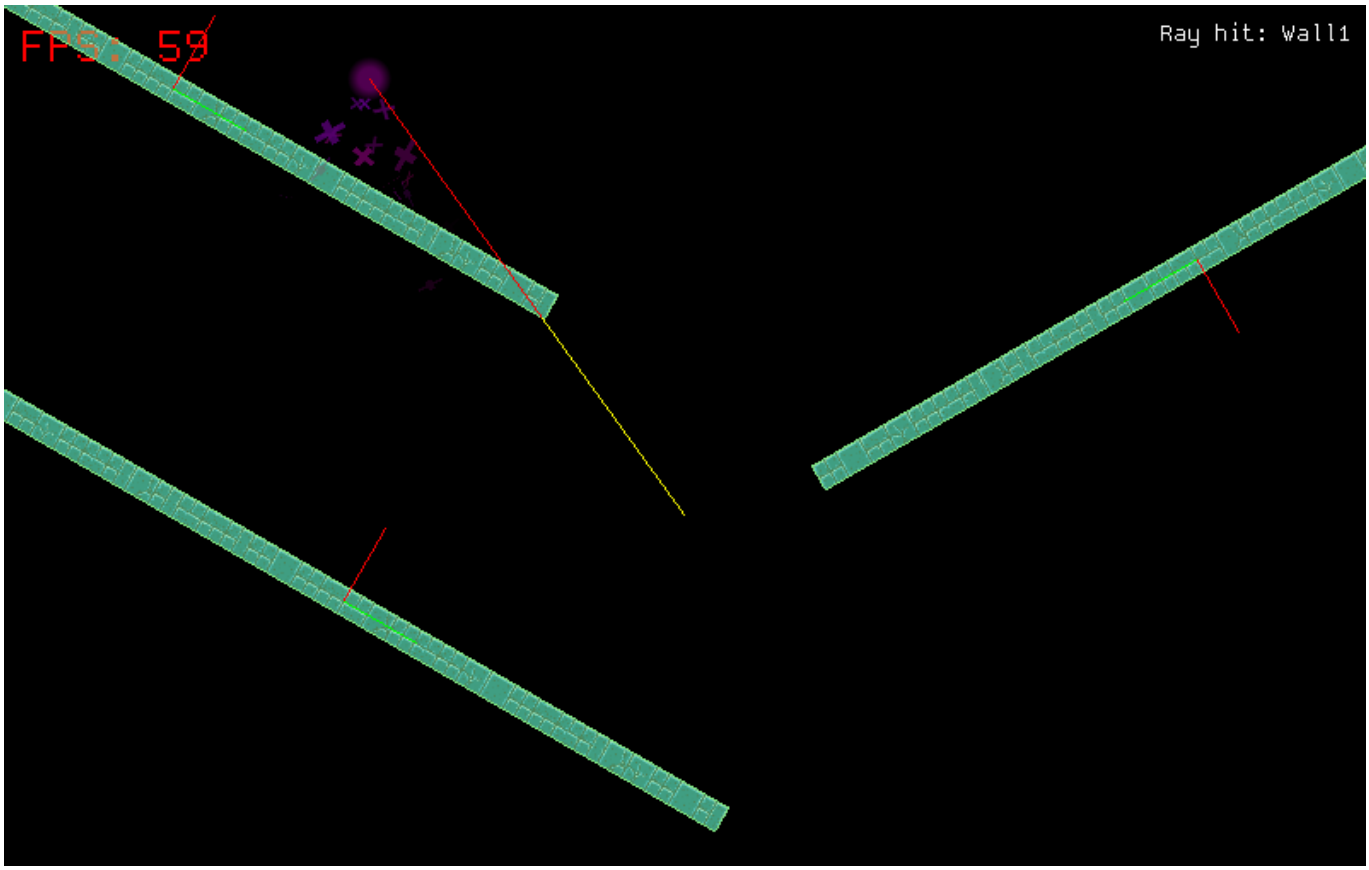
```
[Physics]
ShowDebug = true
```

Then run the debug version of Orx at:

```
orx/code/bin/orxd.exe
```

If you want to turn on the debug using the orx command, press the backtick key (`) and type: `Set Physics ShowDebug true`

While the demo is running, hold the Space key down.



A line will be drawn to represent the ray. Point one will be the centre of the screen and point two will be mouse position. So you can move the ray around.

It is easiest to move the ray over one of the walls to understand what the ray does.

If you move the ray to intersect with a wall, the line will change from a green line, to a yellow and red one.

The point between the yellow and red indicates the point where the wall object has been located.

What data does raycasting give you?

There are several pieces of data that you get from a raycast. These are:

1. The object that the ray has located. There are two modes here. You can either get the first object found, or the nearest. They might not always be the same object.
2. The location where the contact between the ray and the located object occurred. This is different from the location of the object itself.

3. The normal or the direction of the ray where the contact between the ray and the located object occurred.

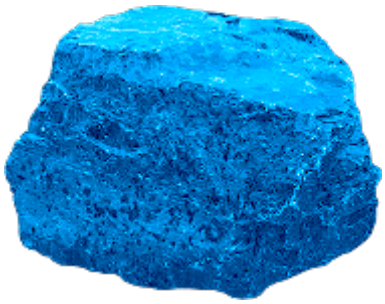
Trying it out

Let's try it out in our own project.

Setting up a new project

To help you work through this tutorial, first create a new blank project using the init script.

Next, rather than use the orx logo all the time, grab the blue rock and copy it to the `data/texture` folder.



Change the default [Object] section to use the ore.png texture (and make it smaller):

```
[Object]
Graphic         = @
Texture         = ore.png
Pivot           = center
AngularVelocity = 18
Scale      = 0.5
```

Change the application to be a standard window so that you can easily get to the console window behind. We'll need to see our logging later.

Add a body to the object so that physics will be enabled on it. Raycasts can only locate objects with bodies:

```
[Object]
Graphic         = @
Texture         = ore.png
Pivot           = center
AngularVelocity = 18
Scale      = 0.5
Body       = OreBody

[OreBody]
```

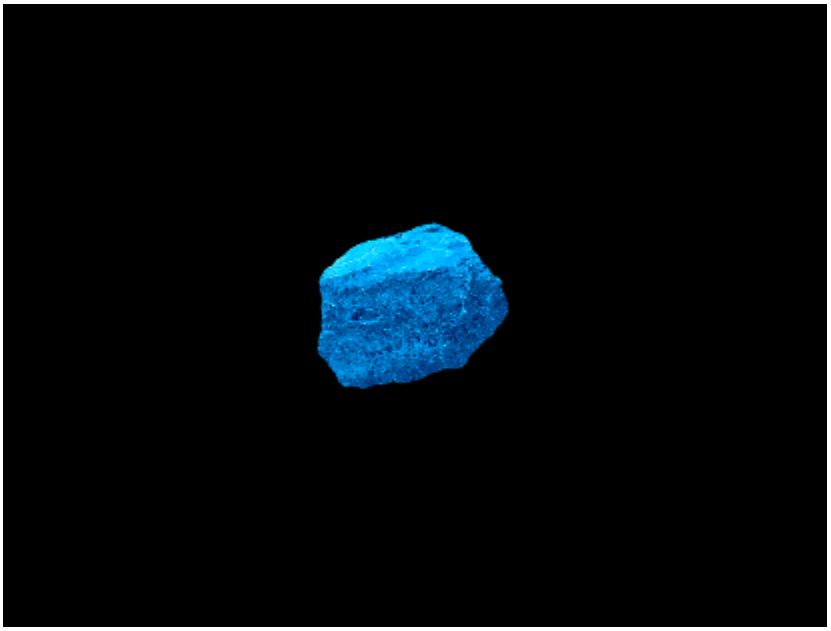
```
Dynamic  = true
PartList = OreBodyPart

[OreBodyPart]
Type      = box
SelfFlags = ore
CheckMask = everythingelse
Solid     = true
```

Enable the physics debug outlining:

```
[Physics]
ShowDebug = true
```

Give that a quick run and you should get a rotating rock with the physics debug border surrounding the object.



Now to do the raycast on every frame. Add to the Update() function:

```
void orxFASTCALL Update(const orxCLOCK_INFO *_pstClockInfo, void *_pContext)
{
    ...
    ...

    orxVECTOR worldPointA = { 0, 0, 0 };
    orxVECTOR worldPointB = { 200, 200, 0 };

    orxOBJECT *collidingFromRayObject = orxNULL;
    collidingFromRayObject = orxObject_Raycast(&worldPointA, &worldPointB,
0xFFFF, 0xFFFF, orxFALSE, orxNULL, orxNULL);

    if (collidingFromRayObject != orxNULL){
        orxLOG("Got something");
```

```
    }
}
```

That's a bit of code, so I'll talk it through. There are two world coordinates defined: one in the centre of the world, and one at 200, 200. So a diagonal line.

There is an object pointer declared and set to orxNULL.

Next, the actual first go using the `orxObject_Raycast` function. The two points are used in parameters one and two.

Parameters three and four are the self flags, and check flags. Notice we set these to `ore` and `everythingelse` on an object in the data config. But using 0xFFFF here means don't filter by any of them. But a reminder, even though we are using 0xFFFF, setting `SelfFlags` and `CheckMask` must be set on any object you might want a raycast to find.

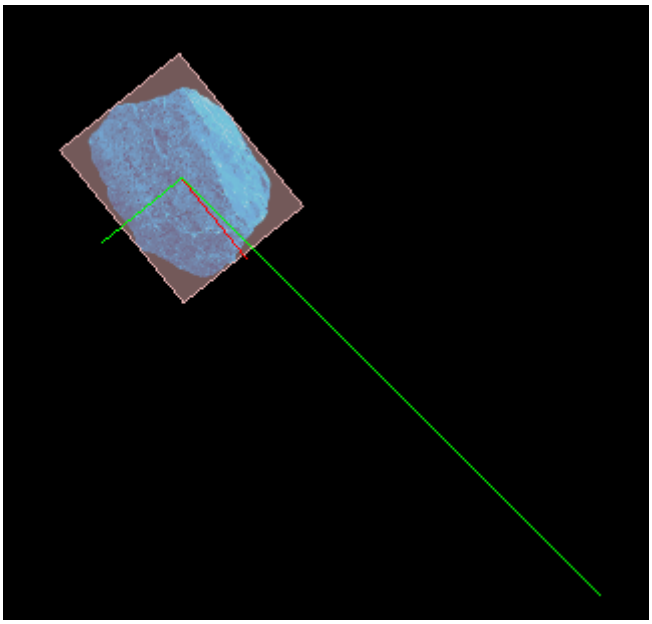
Parameter five is `_bEarlyExit` and `true` on this option means to stop seeking for objects as soon as one is found. However, any object found may not be closest one to point A of the raycast. By saying `false`, the raycast will find the closest object. This might work the physics a tad harder though.

The last two parameters are output parameters. `_pvContact` is the location as a vector where the contact between the ray and the located object occurred. `_pvNormal` is the normal or the direction vector at that point of contact.

Finally, the function will return the object that is found. Or orxNULL if nothing is found.

In the case of the code above, we'll log out a message if an object is found.

But also, when running this code, you'll see the physics debug overlay, the ray is drawn for you so that you can see what is going on.



Now what is going on here?

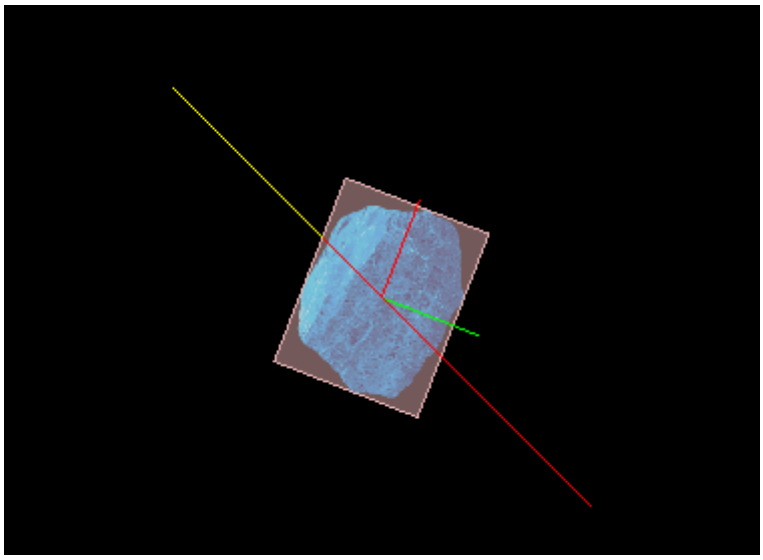
Notice the raycast is starting from the centre of the object, and the centre of the screen. It finishes at 200,200. And the whole line is green. The green line indicates that no object has been located by the

raycast.

This is because of one of the rules listed above is that a cast can't start inside an object. If so, it won't find that object. Therefore, let's move the starting position of the object outside of the starting postion of the raycast:

```
[Object]
Graphic         = @
Texture         = ore.png
Pivot           = center
AngularVelocity = 18
Body       = OreBody
Scale      = 0.5
Position   = (100, 100, 0)
```

Run this and notice a difference in the raycast. The line colour is yellow until it meets the object, and from the point on, the rest of the line is coloured red.



Raycasting is super handy. I hope you can use it to do some pretty cool things.