Creating an Object with many Dynamic Body Parts

Creating a complex object made of many parts, like a boss monster or end-of-level-alien requires some careful planning when designing your config.

There are many ways to achieve certain behaviours in your large object. For example, you may want your boss to completely break apart on contact with a dynamic body like a missile. Or perhaps you might want the whole to stay together, and just parts of the main object to be destroyed piece by piece.

In the weld joint tutorial, I showed how an object and its children (with dynamic bodies) can be joined together using weld joints so that they don't break apart on collision with another body.

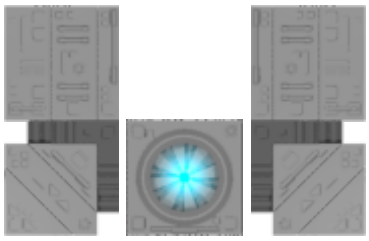
The only problem with this approach is that as the object become more complex or too large, the physics simulations becomes unstable and unreliable. You may see some strange effects at the object distorts and freaks out, to which it can't recover.

The article will cover a more simple, and rock solid approach. The basis is simple: don't use joints, have the main object contain all the bodies, and have all the child not have any bodies.

During the physics contact event (collision), simply check the name of the body part that has collided, and destroy the correct child and body separately in code.

Sound complicated? Not really, let just work through it.

Here are our assets, three sections of the mothership:



The boss' core, the left section and the right section. The rules for our boss object are:

1. All three parts must stay together if colliding with another object
2. Physics with dynamics are needed so there is restitution (ie, some jolt when colliding with another object)
3. One part can be destroyed but the others remain intact together.

Start with the Boss' core object config and get it displaying on the screen:

```
[Boss]
Graphic     = ShipCoreGraphic
Position    = (0, 0, 0.5)

[ShipCoreGraphic]
```

```
Texture       = boss-core.png
Pivot         = center
Smoothing     = true
```

And add the code to display it into the init() function:

```
orxObject_CreateFromConfig("Boss");
```

Great. Next to define the left and right child sections:

```
[LeftChild]
Graphic     = LeftGraphic
Position    = (-57, 0, 0)

[LeftGraphic]
Texture       = boss-left.png
Smoothing     = true

[RightChild]
Graphic     = RightGraphic
Position    = (58, 0, 0)

[RightGraphic]
Texture       = boss-right.png
Smoothing     = true
```

And then add them as children to the Boss' core object:

```
[Boss]
Graphic     = ShipCoreGraphic
Position    = (0, 0, 0.5)
ChildList   = LeftChild # RightChild
```

The next step is to define a body, and three body parts. Normally a body and part would be defined on each child, but in our case, the core object will own all three body parts, and will “overlay” the parts onto the left and right children:

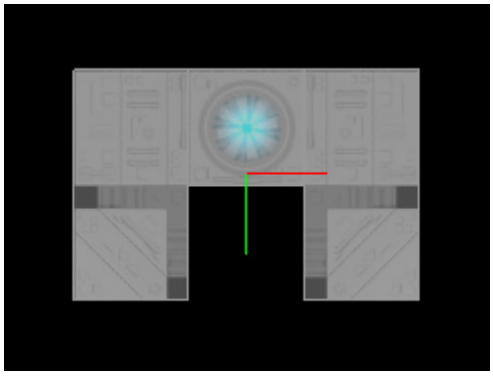
```
[BossBody]
Dynamic        = true
PartList   = BossBodyPart # LeftBodyPart # RightBodyPart

[BossBodyPart]
```

```
Type        = box
Solid       = true

[LeftBodyPart]
Type        = box
Solid       = true
TopLeft          = (-57,0,0)
BottomRight      = (0,115,0)

[RightBodyPart]
Type        = box
Solid       = true
TopLeft          = (58,0,0)
BottomRight      = (115,115,0)
```

Then add the body to the main object:

```
[Boss]
Graphic          = ShipCoreGraphic
Position    = (0, 0, 0.5)
ChildList   = LeftChild # RightChild
Body        = BossBody
```

Run that with physics debug mode on and you'll all three sections covered by a part (owned by the main object).



We'll need a projectile to hit a section of the ship. Here is an asset you can use:

And the config for it:

```
[Projectile]
Graphic      = ProjectileGraphic
Position     = (200, 200, 0.5)
```

```
Body        = ProjectileBody
Speed       = (-300, -480, 0)

[ProjectileGraphic]
Texture       = projectile.png
Pivot         = center
Smoothing     = true

[ProjectileBody]
Dynamic         = true
PartList    = ProjectileBodyPart
LinearDamping  = 1
AngularDamping = 2
FixedRotation  = false

[ProjectileBodyPart]
Type        = sphere
Solid       = true
Friction    = 1.2
Density         = 20
```

Have it created in the init() function with:

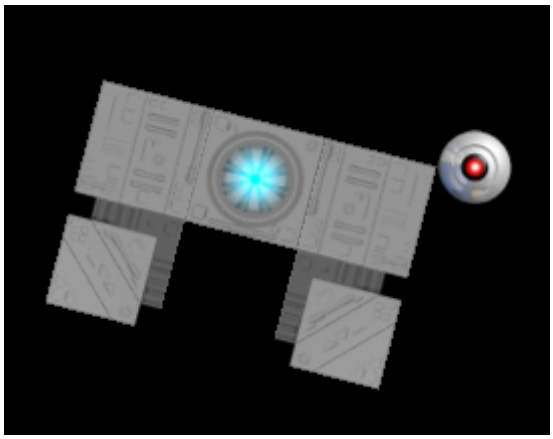
```
orxObject_CreateFromConfig("Projectile");
```

Compile and run and the projectile will go straight through the boss. There has been no self/masks defined to say what collides with what. Change each part so that the projectile collides with everyone one, and everyone else collides with the projectile:

```
[ProjectileBodyPart]
Type        = sphere
Solid       = true
SelfFlags   = projectile
CheckMask   = boss # left # right
Friction    = 1.2
Density         = 20

[BossBodyPart]
Type        = box
Solid       = true
SelfFlags   = boss
CheckMask   = projectile

[LeftBodyPart]
Type        = box
```

```
Solid       = true
SelfFlags   = left
CheckMask   = projectile
TopLeft          = (-57,0,0)
BottomRight      = (0,115,0)

[RightBodyPart]
Type        = box
Solid       = true
SelfFlags   = right
CheckMask   = projectile
TopLeft          = (58,0,0)
BottomRight      = (115,115,0)
```

Compile and run. A projectile will knock the boss flying. It stays together. Great.



The next step is to add a physics handler. Within the handler the name of the part will be checked. For example if the `LeftBodyPart` is hit, that body part will be destroyed, and so will the `LeftChild` object.

Add the handler to the init() function:

```
orxEvent_AddHandler(orxEVENT_TYPE_PHYSICS, PhysicsEventHandler);
```

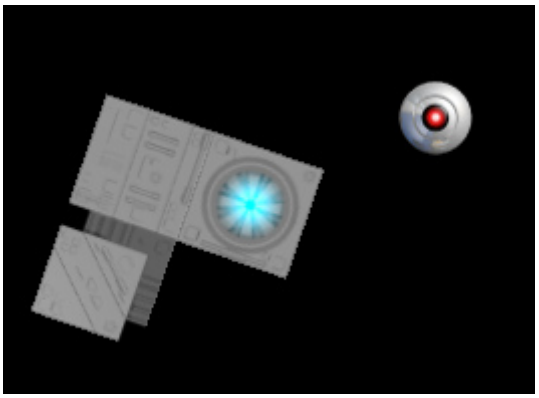
Then the function itself. Apologies for the longish code, but the routine will test collisions between the projectile (and only the Boss' right hand section). If true, the body will be deleted from the man boss object, and the child object will be deleted:

```
void DeleteChildFromObjectByName(orxOBJECT *object, orxSTRING childName) {
    for (orxOBJECT *child = orxOBJECT(orxObject_GetChild(object)); child;
child = orxOBJECT(orxObject_GetSibling(child))) {
        if (orxString_Compare(orxObject_GetName(child), childName) == 0) {
            orxObject_SetLifeTime(child, 0);
        }
    }
```

```
}

void DeleteBodyFromObjectByName(orxOBJECT *object, orxSTRING partName) {
    orxBODY *body = orxOBJECT_GET_STRUCTURE(object, BODY);

    for (orxBODY_PART *part = orxBody_GetNextPart(body, orxNULL); part; part
= orxBody_GetNextPart(body, part)) {
        if (orxString_Compare(orxBody_GetPartName(part), partName) == 0) {
            orxBody_RemovePart(part);
        }
    }
}

orxSTATUS orxFASTCALL PhysicsEventHandler(const orxEVENT *_pstEvent) {

    if (_pstEvent->eType == orxEVENT_TYPE_PHYSICS) {

        orxPHYSICS_EVENT_PAYLOAD *pstPayload;
        pstPayload = (orxPHYSICS_EVENT_PAYLOAD *)_pstEvent->pstPayload;

        if (_pstEvent->eID == orxPHYSICS_EVENT_CONTACT_ADD)
        {
            orxOBJECT *pstRecipientObject, *pstSenderObject;

            /* Gets colliding objects */
            pstRecipientObject = orxOBJECT(_pstEvent->hRecipient);
            pstSenderObject = orxOBJECT(_pstEvent->hSender);

            const orxSTRING senderName = orxObject_GetName(pstSenderObject);
            const orxSTRING recipientName =
orxObject_GetName(pstRecipientObject);

            if (orxString_Compare(senderName, "Projectile") == 0 &&
                orxString_Compare(pstPayload->zRecipientPartName,
"RightBodyPart") == 0
                ) {

                DeleteChildFromObjectByName(pstRecipientObject,
"RightChild");
                DeleteBodyFromObjectByName(pstRecipientObject,
"RightBodyPart");
            }

            if (orxString_Compare(recipientName, "Projectile") == 0 &&
                orxString_Compare(pstPayload->zSenderPartName,
"RightBodyPart") == 0
                ) {
```

```
                DeleteChildFromObjectByName(pstSenderObject, "RightChild");
                DeleteBodyFromObjectByName(pstSenderObject,
"RightBodyPart");
            }
        }
    }

    return orxSTATUS_SUCCESS;
}
```

Compile and run. The projectile will strike the right hand side of the boss ship, destroy that side and the force will send the ship flying off. At the same time the remaining parts will stay joined together.



This is a great technique for complex objects. Thanks to Iarwain for suggesting it in the chat room.