Moving a ScrollObject along a curve segment

In this tutorial we'll go work through how to make an object travel along a curve segment.

Set up an orx/Scroll project

You can follow this tutorial to help you set up a new orx/Scroll project. Once set up, create a ScrollObject class for your ship. We will make use of the Update function in order to update the ship's position every frame.

Assets

Here is a ship graphic that can be used:

The config for the ship which will fly along the curve is:

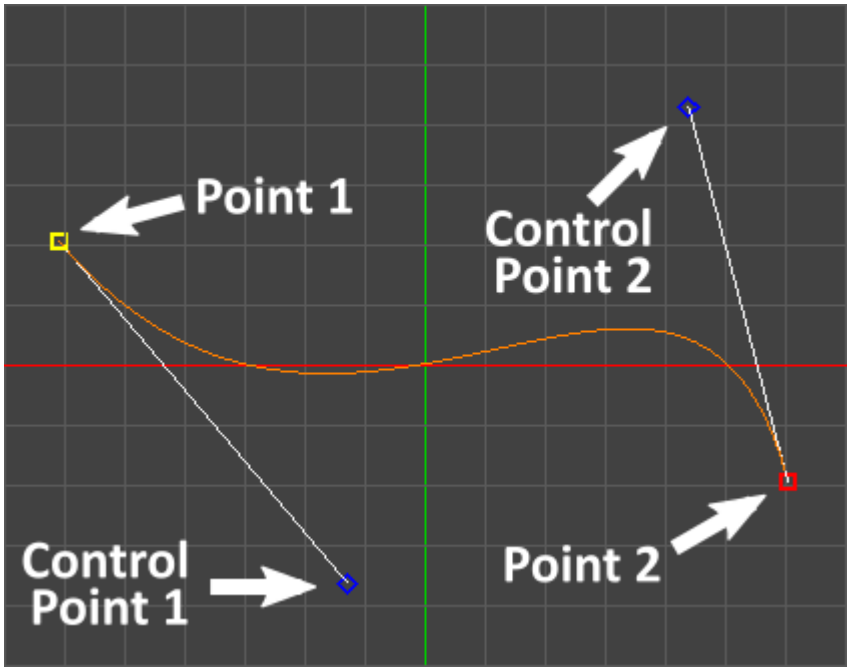
```
[Ship]
Graphic   = @
Texture   = fighter.png
Pivot     = center
Smoothing = true
```

The Curve

We'll need a curve for the ship to travel along. A bezier curve segment is made up of:

- 2 Points, for the start and end, and;
- 2 Control Points, for adjusting the size of the curve

Here is an illustration of a curve from the Orx Curve Editor:



This tutorial centers around the use of the `orxVector_Bezier` function. This function takes four vectors, 2 for the points, and 2 for the control points. The signature of the function looks like this:

```
orxDLLAPI orxVECTOR* orxFASTCALL orxVector_Bezier (orxVECTOR *_pvRes,
        const orxVECTOR *      _pvPoint1,
        const orxVECTOR *      _pvPoint2, //control point
        const orxVECTOR *      _pvPoint3, //control point
        const orxVECTOR *      _pvPoint4,
        orxFLOAT      _fT
    )
```

Before we discuss the function further, lets get some vectors for the curve segment first.

The curve segment we'll use in our config is the following:

```
[Curve]
Points = (-243.072, -82.3369, 0) # (-51.0597, 145.47, 0) # (175.771,
-171.898, 0) # (241.819, 77.2513, 0)
```

This is the format that is outputted from the Orx Curve Editor, but you can use any collection of four vectors you like.

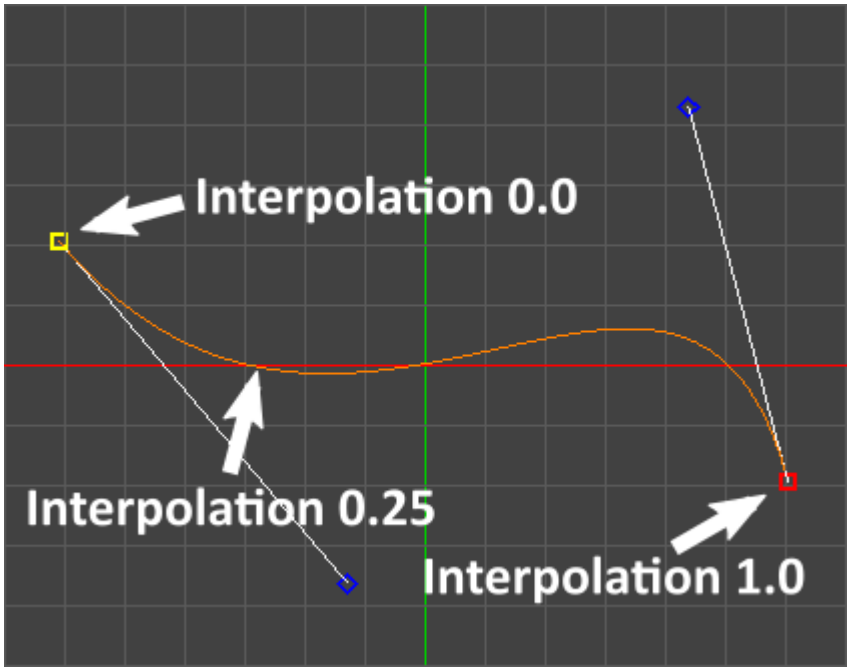
To retrieve the four vectors from the config, you can use:

```
    orxVECTOR vectors[4];

    if (orxConfig_PushSection("Curve")) {
        for (int i = ; i < 4; i++) {
            orxConfig_GetListVector("Points", i, &vectors[i]);
```

```
        }
        orxConfig_PopSection();
    }
```

The last parameter of the `orxVector_Bezier` function takes the interpolation position. This is a value between 0.0 and 1.0. 0.0 is the very beginning of the curve segment. 1.0 is the very last point of the curve segment. A value between 0.0 and 1.0 is a position somewhere along the curve.

To illustrate, see the following:



In the image above, by setting 0.25 as the interpolation position, you will locate a vector position a quarter of the way along the curve segment.

The first parameter of the `orxVector_Bezier` function is the vector to store the result.

Now we have everything we need.

Making the object move

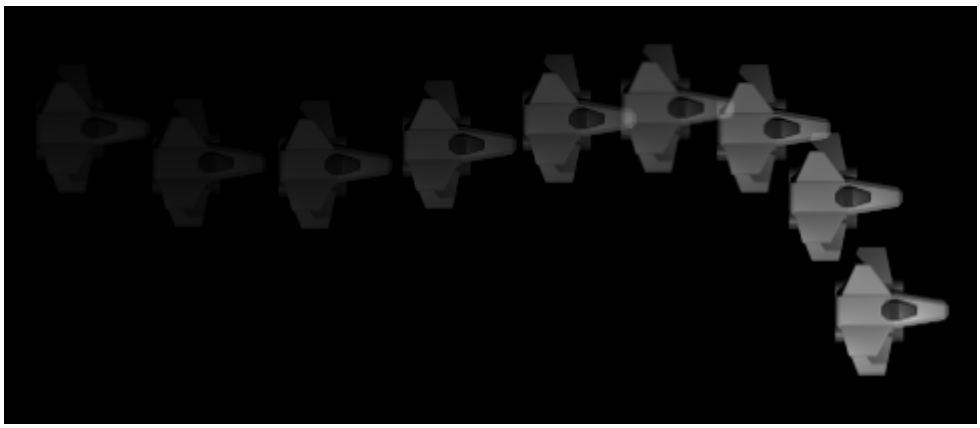
Now that we have the four vectors that define the curve segment, and an output vector to store the result, then we can loop over values between 0.0 and 1.0 and set our object position to whatever vector is returned from `orxVector_Bezier`.

We can do this in the `Update` function of our ScrollObject:

```
void Ship::OnCreate()
{
    if (orxConfig_PushSection("Curve")) {
        for (int i = ; i < 4; i++) {
            orxConfig_GetListVector("Points", i, &vectors[i]); //define
```

```
vectors[4] orxVECTOR array in the header
        }
        orxConfig_PopSection();
    }
}

void Ship::Update(const orxCLOCK_INFO &_rstInfo)
{
    orxVECTOR shipPosition = orxVECTOR_0;

    orxVector_Bezier(&shipPosition,
        &vectors[],
        &vectors[1],
        &vectors[2],
        &vectors[3],
        curveIndex
        );

    SetPosition(shipPosition, orxTRUE);
    SetRotation(orxMATH_KF_PI_BY_2, orxTRUE); //face right

    curveIndex += 0.001; //define curveIndex orxFLOAT in the header
    if (curveIndex > 1.0) {
        curveIndex = ;
    }
}
```

This will have the ship fly smoothly over the curve. So... success!



Of course, the current effect would work well for a character who's rotation isn't intended to change. But for a ship, the rotation should follow the curve. There is a simple method to achieve this:

1. Take the current vector position along the curve
2. Take the next vector position along the curve
3. Do an ATAN with the two vectors
4. Set the result as the rotation.

Our updated `Update` function will look like this:

```
void Enemy::Update(const orxCLOCK_INFO &_rstInfo)
{
    const orxFLOAT unit = 0.001;

    orxVECTOR shipPosition = orxVECTOR_0;

    orxVector_Bezier(&shipPosition,
        &vectors[],
        &vectors[1],
        &vectors[2],
        &vectors[3],
        curveIndex
        );

    SetPosition(shipPosition, orxTRUE);

    if (curveIndex + unit < 1.0) {
        orxVECTOR nextShipPosition = orxVECTOR_0;
        orxVector_Bezier(&nextShipPosition,
            &vectors[],
            &vectors[1],
            &vectors[2],
            &vectors[3],
            curveIndex + unit
            );

        orxFLOAT rotation = orxMath_ATan((shipPosition.fX -
nextShipPosition.fX), (shipPosition.fY - nextShipPosition.fY));
        SetRotation(-rotation, orxTRUE);
    }

    curveIndex += unit;
    if (curveIndex > 1.0) {
        curveIndex = ;
    }
}
```

Now the ship should follow the curve both in position and orientation.

Complex curves contain more than one segment. If you wish to make your object travel along a more complicated curve, you'll need to code a routine that moves the object through a list of points and control points, but the principle is the same.

Its worth noting that very soon, Orx will be able to take a direct list of points through a Position FX and do this job for you. This tutorial will be updated once that feature is ready.

Enjoy working with objects on curves!