More about the co-ordinates.... the guide you've all been waiting for, Part Four and the cat is still hanging in there . . .

Eejit's Guide

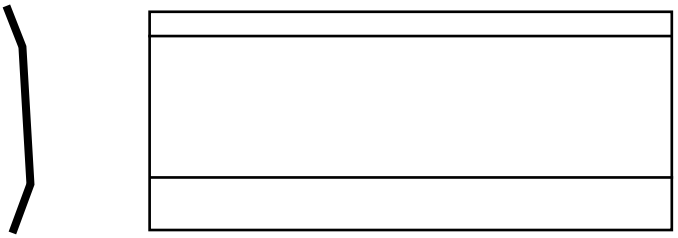
to building object for OpenBVE trainsim routes

by graymac

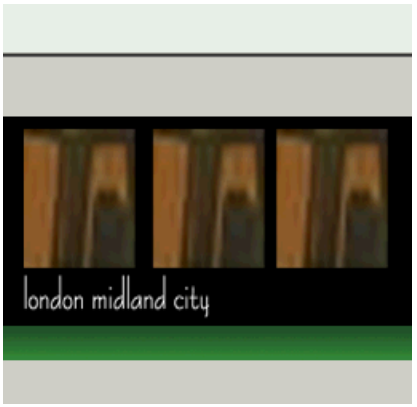
Working with co-ordinates

Using a single face is often restrictive and inefficient. Single mages can be easily attached to multiple faces by defining the co-ordinates correctly. As an example take the side of a typical DMU:

The profile for this side would look some-thing like this:



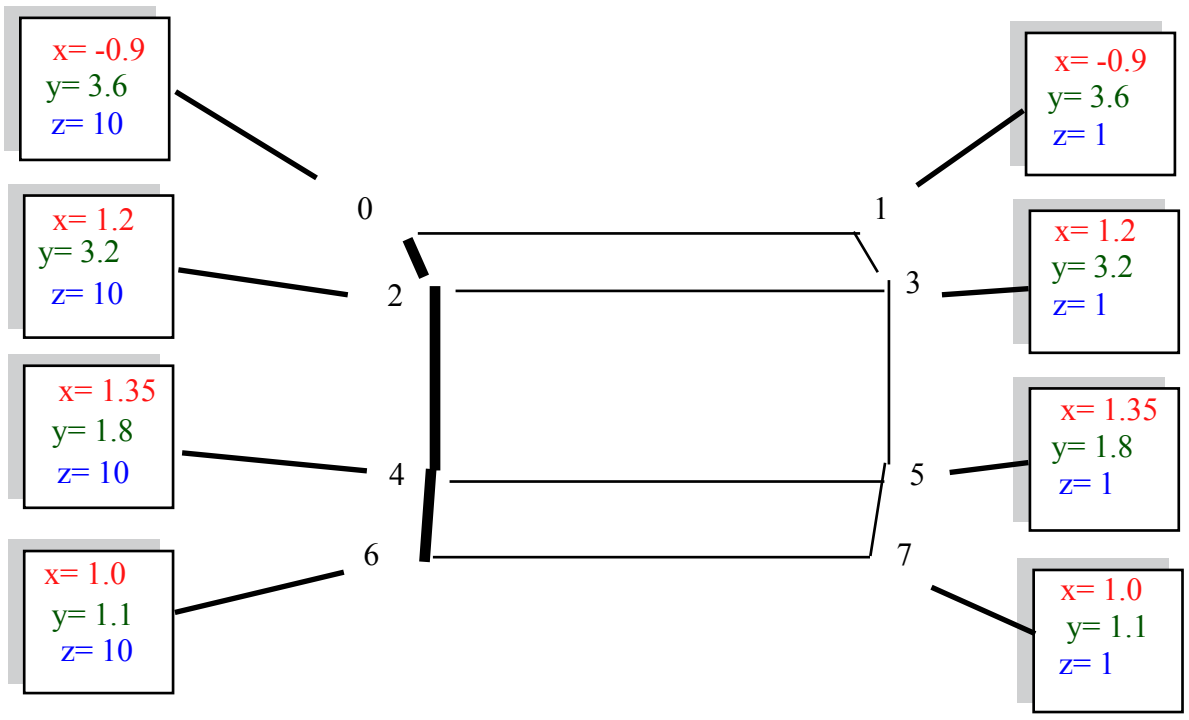
This image, prepared as a 256x256 bit-map, is to be spread across the three faces so that the joints in the faces coincide with the correct parts of the image. The co-ordi-nates are the key to doing this.



First, find your dimensions and number your vertices. Follow the example shown here as it is typically the way most developers find best for ordering the numbers. Remember, the faces are drawn in a clockwise direction, but that doesn't mean the vertices have to have clockwise con-secutive order.

Never be afraid to sketch out your intended faces as you work. A representation of how it looks when its flattened out can be helpful to visualise the job and see the logical order to number the vertices.

The dimensions used here are not intended as an accurate representation of the particular Class. For the purpose of seeing how this coordinate stuff works it will be fine. Obviously, your meticulous research will allow the correct ones when you make your real roject.



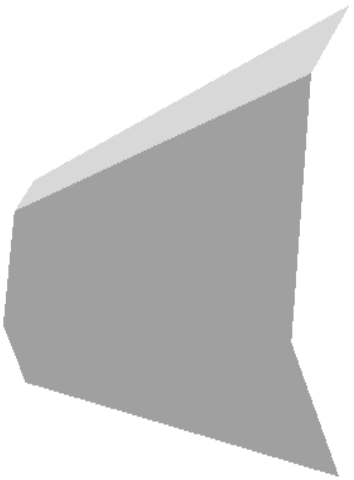
Using the dimensions given we can write the mesh code like so:

```
[meshbuilder]
vertex -0.9,3.6,10
vertex -0.9,3.6,1
vertex -1.2,3.2,10
vertex -1.2,3.2,1
vertex -1.35,1.8,10
vertex -1.35,1.8,1
vertex -1.0,1.1,10
vertex -1.0,1.1,1
```

Define the three faces, ordering itheir veretx numbers in a clockwise direction

```
face 0,1,3,2
face 2,3,5,4
face 4,5,7,6
```

Take a look at the result in the object viewer. You should see something similar to the picture here:



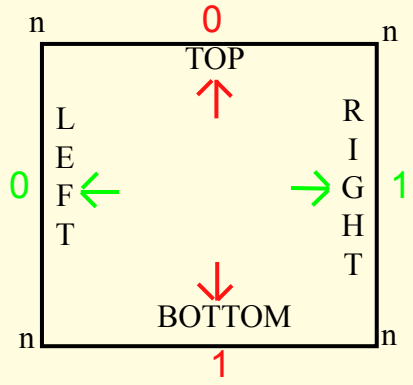
Now we have a mesh to hang the image on. The next bit is how the image relates to the mesh and how to work out the coordinates.

The prepared image will conform dimensioally to the rule of >2, that is to say it will have width and height dimensions, in pixels, of 32, 64,128, 256 etc. This may make it look distorted compared to a normal photo but it won't matter as the program takes care of the scaling.

From the previous guides, remember . . .

```
coordinates n,x,y
```

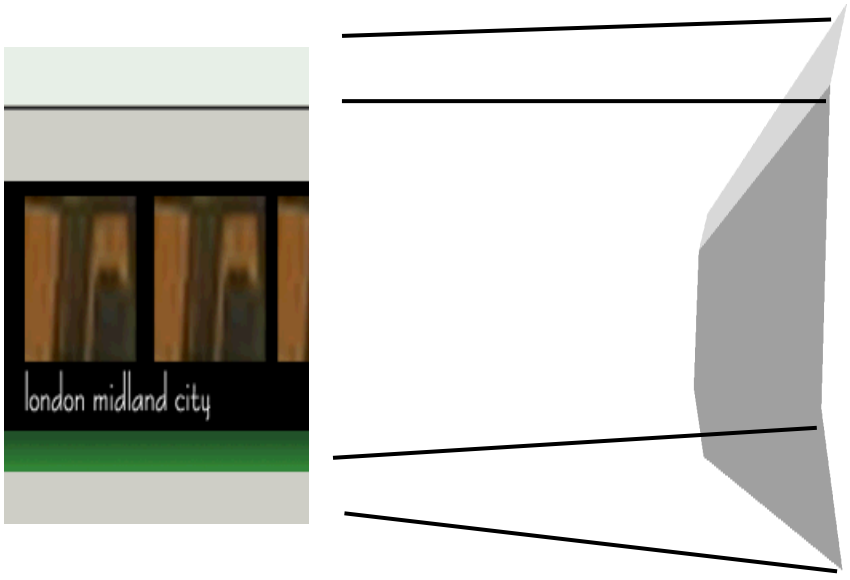
The three parameters are n=vertex number; x= horizontal position and y= vertical position.
A x number of 0 places the left of the image, a 1 places the right.
A y number of 0 places the top of the image, a 1 places the bottom.



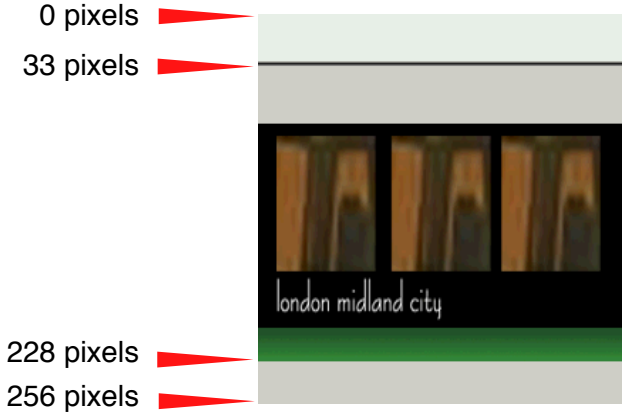
This is easy when there is only one image and one face. If an image needs to be split over two or more faces then the coordinates will be values which need to be calculated and for this exercise at least will be found somewhere between 0 and 1.

With this example the horizontal (x) coordinates will be either 0 or 1, as the whole image is used from side to side undivided. The vertical (y) coordinates are split because the faces join at points on the vertical , coinciding with the gutter line and just at the bottom of the green stripe.

Looking at the side and at the image let's work out at what points the mesh joints coincide in the picture.
Take a line from the edge of the face to the part of the picture where the join should be:



You will have to use you image editor now. You need to count the pixels from the top of the image to the place where the face joins occur. Once you know these values it will be easy to calculate the co-ordinate values from them. In this example the vertical pixel count is shown:



Now, to calculate the coordinate value it is simply a matter of **dividing** the pixel value of the point you have determined by the **whole pixel number** of the direction to be split. In this case it is vertical, 256.
So, at the top of the image the pixel count is 0 divided by 256 = 0
At the point of the gutter line the pixel count is 33 divided by 256 = 0.129
At the point below the green stripe the pixel count is 228 divided by 256 = 0.890
At the bottom of the image the pixel count is 256 divided by 256 = 1

Whatever the dimension of any image you want to split, the formula is the same, take your found number of pixels and divide by the whole pixel count for the direction you're dealing with. If its vertical, count from the top, if its horizontal, count from the left.

So, let's fill in the values:
In this exampl there will be eight coordinates, corresponding to the vertices.

```
coordinates 0,
coordinates 1,
coordinates 2,
coordinates 3,
coordinates 4,
coordinates 5,
coordinates 6,
coordinates 7,
```

The horizontal values are easy, 0 to the left, 1 to the right. Look at the diagram of the vertices and see how neatly it works out, as all the left vertices were numbered evenly and all the right vertices have the odd numbers.

```
coordinates 0,0,
coordinates 1,1,
coordinates 2,0,
coordinates 3,1,
coordinates 4,0,
coordinates 5,1,
coordinates 6,0,
coordinates 7,1,
```

And now the vertical values are inserted, according to our calculations. The way the vertices were numbered means it is easy to see how the values shift from top to bottom, each pair on a particular height.

```
coordinates 0,0,0
coordinates 1,1,0
coordinates 2,0,0.129
coordinates 3,1,0.129
coordinates 4,0,0.89
coordinates 5,1,0.89
coordinates 6,0,1
coordinates 7,1,1
```

Now take a look at he result in the object viewer. It may not be absolutely realistic, but as long as you have got this far with the example you now understand how to command the coordinates to do what you want them to. Researching the proper dimensions from the real thing, and making up your image textures, well, now that's down to you . . .