Route Building for OpenBVE
- How it works (a simplified explanation)

The program uses a simplified (for the builder) 3D modelling method. Your "train" moves along a "running rail" from start to finish. The view that you see on your screen is from the running rail, taken at the same height above it as the engine driver's eye-line. Imagine it as a camera where the driver's head would be. In use, you are normally looking forward in the running direction.

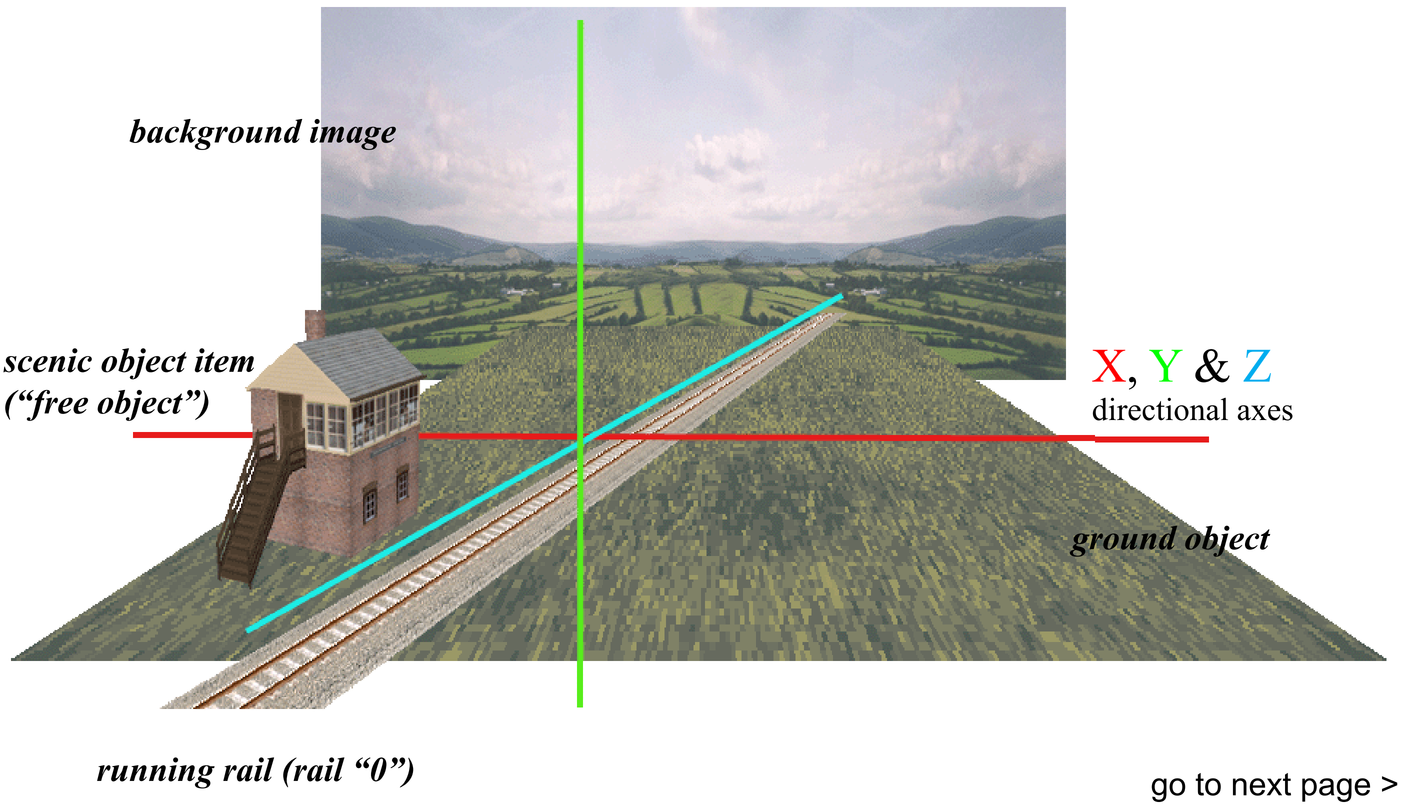
At its most basic all a route has to have is a starting station and a finishing station, plus a "running rail" to connect them. Of course this alone would hardly be sufficient for most users.

start
finish

Like reality, the route has space in three dimensions. Width, height and distance. These three values are the key to everything. Your running rail starting point will be given a value (in metres, it's all based on metric system) Every item or object in the route will be placed relative to a given distance along this running rail. If it is to the left of the running rail it will be a minus value, if to the right a positive value. The height of objects above or below the rail will also be given as plus or minus figures.

Technically, these directions are called "axes" (plural of 'axis') and they correspond to "X" - left to right; "Y" - up and down; & "Z" - backwards and forwards.

Here it is in a simple diagram. You should be able to visualise this idea, if you can do so then you will see what you're aiming for, this is the "baseboard" where we make and place the numerous items that will be seen on your train ride. All I've shown here is a single rail, a background image and a "ground" item (bit of old plain grass!) and one scenic "object" - a signal cabin placed to the left of the rail.



go to next page >

I said it was “simplified” 3D modelling. The “instructions” for everything are entered in plain text (I only use windows notepad, nothing fancy) using a simple code. The file is saved as “.csv” format (“comma-separated values”) but it’s simply a limited number of instructions to place an item accurately in a particular place. There are classes of objects, again a limited number of these, and every one has its own special identity which you give it, entered in a list of “Structures” (this list can be included IN the route.csv file as it always was in the earlier BVE4 routes, or made as a separate .txt list for OpenBVE, but you will find all about that later).
Some classes of object include “free objects”; rail objects; ground objects and others.
“Free Objects” are the things you make to put in the route, such as signal boxes, houses, trees, platforms etc. You make them separately and give them a unique number in the structures list to identify them and tell the program where they are stored (folder location) then when you come to place them in your route you simply enter a line such as:

```
.freeobj 0;99;x;y;z
```

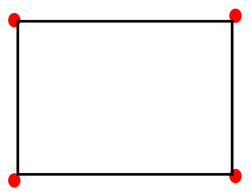
- where 0 is the ‘index’ of the running rail, 99 is the ident number you gave your object and x,y & z is the value of distances from the rail in metres (the running rail is always”0” in the above - don’t worry about ‘other’ rail numbers at this stage). And that’s the main “bones” of it, some similar instructions exist to place the other items where you want them (rails, walls, dikes, ground and poles are the other common item categories).

There’s full documentation available for building a .csv routefile and there’s a lot in it. Though mostly it is as easy as I’ve described above. Start simple, you have to walk before you can run. If you think you have understood this explanation, so far, then you might see to looking into making your own route and building items to put in it.

Another brief explanation. Object making uses a different but similarly simplified code, based on the same principles, by defining **points** (we call them *‘vertices’*, plural of *‘vertex’*) in a space as X,Y,Z values - same as the routefile. That

For example, consider a flat, rectangular wall standing upright. It can be described (mathematically - don’t get worried!) by four vertices two up & down left, two up & down right. *(see red dots)*

By “connecting” the four vertices we can create a “face” . . . think of a face as a solid planar surface



That rectangular surface (face) can have a solid colour applied to it. Or, better, a “texture” “Texture” is a word by which we mean a picture file used to cover a surface.
You’ve probably seen the simpler Hollywood film sets (think of the old “westerns”) where a street full of buildings was just a big board painted with a picture of the sheriff’s jail, saloon and the old whorehouse, or whatever. At it’s most basic, that is all a simple OpenBVE route and object really is.

It’s not too hard to learn to code the basic shapes for the object file. They’re done in a text editor (windows notepad is all you really need) and saved in one of two formats, either .csv or .b3d. Principles are the same for both, but .b3d is cleaner and quicker for the newbie than .csv and less prone to errors with “syntax” - you don’t need to worry about it just now, trust me on this. I can use either but personally I prefer .b3d as being quicker to write. In a route with hundreds of objects included you can see why that’s important.

Hope this insight helps those in the OpenBVE community who want to try their hand at putting something back in.