The UNOFFICIAL csv Object Handbook for OpenBVE

Patrick Norqvist

The

UNOFFICIAL

csv Object Handbook for OpenBVE

Patrick Norqvist

The Unofficial csv Object Handbook for OpenBVE

Author: Patrick Norqvist
ISBN: 978-91-982993-3-5
Website: www.openbve.net

First edition

Preamble

This handbook is a try to explain the csv object file format for OpenBVE. It is a follow up to the previously published handbook on the x object file format.

The material is mainly derived from the documentation published on the BVEstation Wiki website `http://wiki.bvestation.com/`.

As this is the first edition, there may be errors that have escaped the proof-reading process. The author apologies if this is the case, and kindly ask readers to report any errors found to the e-mail address `error@openbve.net`

The reader is expected to be familiar with the operation of the OpenBVE train simulator, and having installed and be familiar with development programs such as the OpenBVE Object Viewer, and text-file editors.

Credits are also given to:

- BVE Klub, with its website `http://www.bveklub.hu/` , from which the illustration to the Shear statement in chapter 5 is taken.
- Mr. Anthony Bowden, with his website `http://www.railsimroutes.net/` , for his csv object files in the OpenBVE route "Birmingham Cross-City South", another good source of information, and also the source of the tree texture taken as an example is chapter 6.

Table of Contents

Chapter 1

Building a simple object

The csv file format allows us to build objects by writing a number of statements in a text file. Most of the statements used in the csv file format, has its equivalents in the b3d object file format, and in a less extent in the x object file format The file extension should be ".csv".

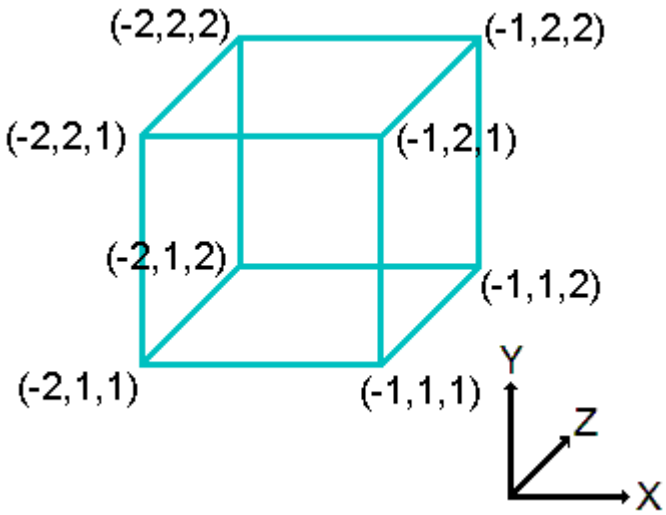
To learn to know the csv object file format, we start with making a simple object, a cube with colored surfaces.

The first we write in the text file is the CreateMeshBuilder statement:

```
CreateMeshBuilder
```

This statement begins a section of definition of a mesh, its faces and colors/textures. There can be more than on CreateMeshBuilder statement in an object file. All other statements refers to the last CreateMeshBuilder section of the object file..

Now we will start to define the vertices for the cube we are going to build.



Each vertex has the coordinates (x,y,z) to define its location in the 3D world. The cube has a side of 1 meter, and is located with its right bottom front corner (-1,1,1) 1 meter to the left on the x-axis (which gives a negative x coordinate), one meter up on the y axis and 1 meter forward on the z axis.

We add the cube's 8 vertices to the csv file using the AddVertex statement:

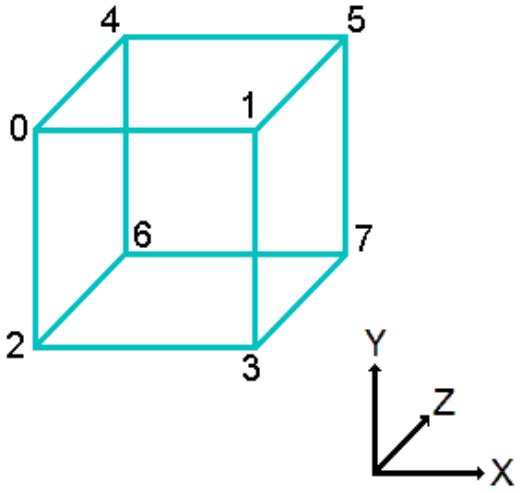
```
CreateMeshBuilder

  AddVertex, -2.000000,2.000000,1.000000, ; Vertex 0
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex 1
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex 2
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex 3
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex 4
  AddVertex, -1.000000,2.000000,2.000000, ; Vertex 5
  AddVertex, -2.000000,1.000000,2.000000, ; Vertex 6
  AddVertex, -1.000000,1.000000,2.000000, ; Vertex 7
```

After the AddVertex statement follow the coordinates x, y, z; for each of the 8 vertices of the cube. The 8 vertices has numbers from 0 to 7

We also make comments in the code. A comment starts with a semicolon "**;**".

There is also a possibility to add another set of x,y,z coordinates as parameters to each AddVertex statement. That is if one want to tweak the normals of the vertex. If no such parameters are given, such in this example, the normals are automatically calculated.

We can pick the vertices in any order, in this case they were picked as this:



Next we define the faces of the cube. Faces are surfaces that are "stretched" between vertices. The cube may have up to 6 faces. If we define all or just a few depends on if all feces well be seen when the object is placed in the simulation.

In this this case, we will define all possible faces of the tube so that we can turn it any way and still see the object as a cube.

We add the cubes 6 faces using the numbers of the vertices that we have already begun with. We define a face using the AddFace statement, and as parameters listing the vertex numbers, in clockwise order seen from the outside of the object:

```
CreateMeshBuilder

  AddVertex, -2.000000,2.000000,1.000000, ; Vertex 0
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex 1
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex 2
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex 3
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex 4
  AddVertex, -1.000000,2.000000,2.000000, ; Vertex 5
  AddVertex, -2.000000,1.000000,2.000000, ; Vertex 6
  AddVertex, -1.000000,1.000000,2.000000, ; Vertex 7

  AddFace, 0, 1, 3, 2,  ; Face 0
  AddFace, 1, 5, 7, 3,  ; Face 1
  AddFace, 5, 4, 6, 7,  ; Face 2
  AddFace  4, 0, 2, 6,  ; Face 3
  AddFace, 4, 5, 1, 0,  ; Face 4
  AddFace, 2, 3, 7, 6,  ; Face 5
```

Here face 0 is the front of the cube, face 1 the right side, face 2 the back side, face 3 the left side, face 4 the top and face 5 the bottom of the cube.

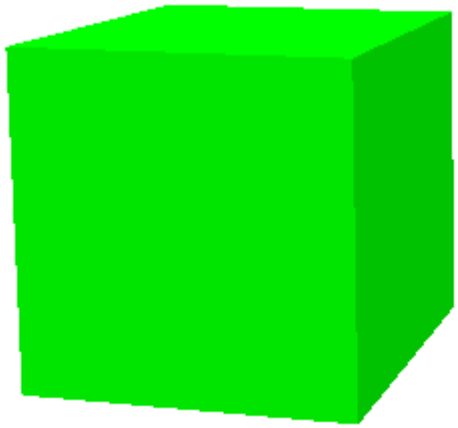
Then we want to add colors to the faces so that they can be seen. We use the SetColor statement that will apply a color to each face in the same CreateMeshBuilder section. Here we want a green color on all 6 faces.

```
CreateMeshBuilder

  AddVertex, -2.000000,2.000000,1.000000, ; Vertex 0
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex 1
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex 2
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex 3
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex 4
  AddVertex, -1.000000,2.000000,2.000000, ; Vertex 5
  AddVertex, -2.000000,1.000000,2.000000, ; Vertex 6
```

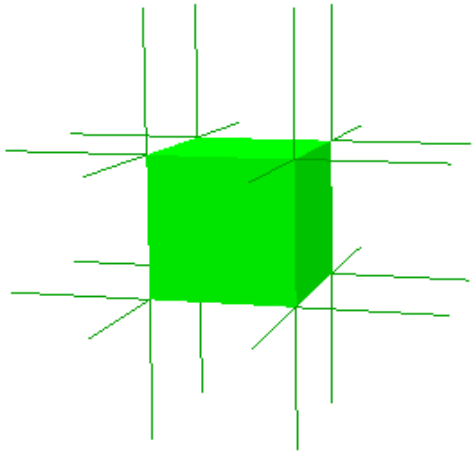
```
AddVertex, -1.000000,1.000000,2.000000, ; Vertex 7

AddFace, 0, 1, 3, 2,  ; Face 0
AddFace, 1, 5, 7, 3,  ; Face 1
AddFace, 5, 4, 6, 7,  ; Face 2
AddFace, 4, 0, 2, 6,  ; Face 3
AddFace, 4, 5, 1, 0,  ; Face 4
AddFace, 2, 3, 7, 6,  ; Face 5

SetColor, 0, 255, 0, 255  ; Red Green Blue Alpha
```

The SetColor statement takes a number of parameters. The first parameters are the colors that is separated into the format RGBA, which means the first parameter determines the amount of red, the second the amount of green, the third the amount of blue, and the fourth the alpha channel. All colors can be made up of a proper mix of red, green and blue. The alpha channel determines the amount of transparency of the color. As we want no transparency, we set the value to 255 (a value of 0 will mean full transparency). The value for each of these 4 parameters can range from 0 to 255.

Now we have the complete text for the csv object file of the green cube. If we put this text in a text file with the name such as “Cube_green.csv” we can watch the result in the OpenBVE Object Viewer:



If we in the OpenBVE Object Viewer toggles Normals to on, we will see the normals that were automatically created for the vertices of each surface:



If we for some reason need to be able to see a face from both sides, there is a special statement AddFace2. Its parameters are the same as for the AddFace statement, and the vertices' numbers are given in clockwise order for the front side face. The difference is that the face will also be visible from the other side. There are however a few drawbacks: The lighting of the back side might not be correct, but be the same as on the front side face. Only convex polygons are supported for use of the AddFace2 statement. For the green cube, using the AddFace2 statement gives us this code:

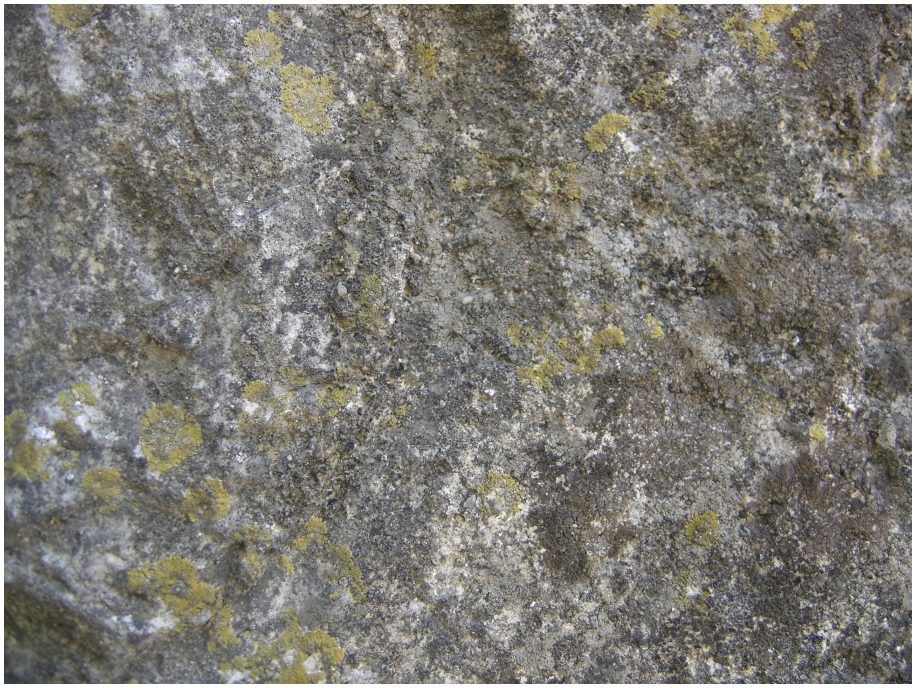
```
CreateMeshBuilder

  AddVertex, -2.000000,2.000000,1.000000, ; Vertex 0
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex 1
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex 2
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex 3
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex 4
  AddVertex, -1.000000,2.000000,2.000000, ; Vertex 5
  AddVertex, -2.000000,1.000000,2.000000, ; Vertex 6
  AddVertex, -1.000000,1.000000,2.000000, ; Vertex 7

  AddFace2, 0, 1, 3, 2,   ; Face 0
  AddFace2, 1, 5, 7, 3,   ; Face 1
  AddFace2, 5, 4, 6, 7,   ; Face 2
  AddFace2, 4, 0, 2, 6,   ; Face 3
  AddFace2, 4, 5, 1, 0,   ; Face 4
  AddFace2, 2, 3, 7, 6,   ; Face 5

  SetColor, 0, 255, 0, 255  ; Red Green Blue Alpha
```

Chapter 2

Using textures

Now we will use the cube created in chapter 1 and see how to apply a texture to the cube's faces. We will use a rock surface texture, to make it looks like the cube is made from stone.

Searching the Internet, we find a number of suitable textures. One is downloaded.



Using a simple image editing program, such as the freeware program IrfanView, we change the width and height of the picture to be a power of 2. If the original image width was 2592 pixels and the height 1944 pixels, we can change the with to 2^{11}=2048 pixels and the height to 2^{10}=1024 pixels. After adapting the image size, we save the image as a Portable Network Graphics (.png) file with the name "RockTexture.png". The Portable Network Graphics format compresses the file without distorting it in any way.

In the csv object file for the cube, we will replace the SetColor statement with the LoadTexture statement.

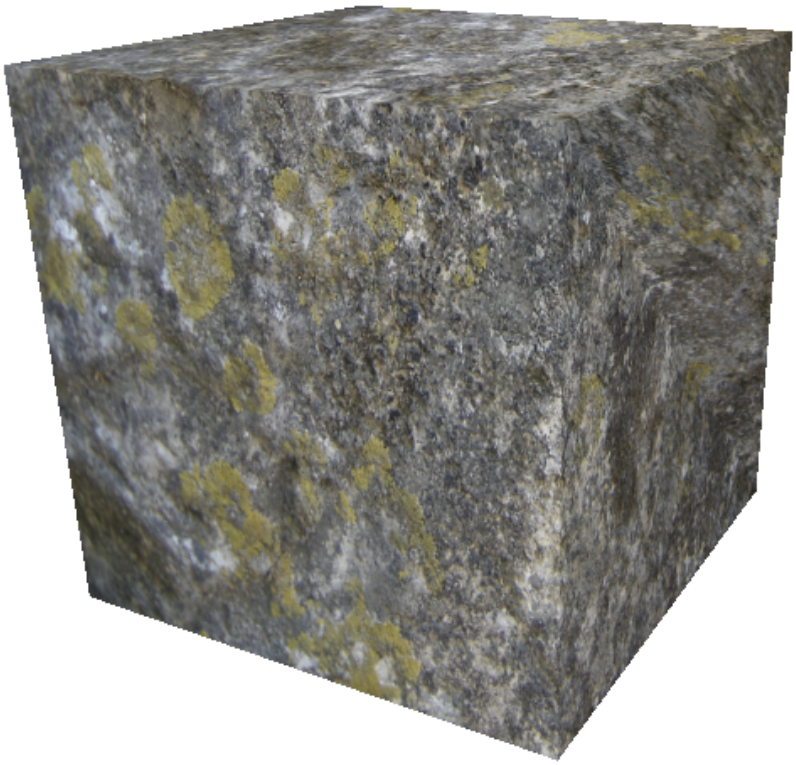
We will also add texture coordinates for all 8 vertices.

The whole csv object file will now be this:

```
CreateMeshBuilder

  AddVertex, -2.000000,2.000000,1.000000, ; Vertex 0
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex 1
```

```
AddVertex, -2.000000,1.000000,1.000000, ; Vertex 2
AddVertex, -1.000000,1.000000,1.000000, ; Vertex 3
AddVertex, -2.000000,2.000000,2.000000, ; Vertex 4
AddVertex, -1.000000,2.000000,2.000000, ; Vertex 5
AddVertex, -2.000000,1.000000,2.000000, ; Vertex 6
AddVertex, -1.000000,1.000000,2.000000, ; Vertex 7

AddFace, 0, 1, 3, 2,  ; Face 0
AddFace, 1, 5, 7, 3,  ; Face 1
AddFace, 5, 4, 6, 7,  ; Face 2
AddFace, 4, 0, 2, 6,  ; Face 3
AddFace, 4, 5, 1, 0,  ; Face 4
AddFace, 2, 3, 7, 6,  ; Face 5

LoadTexture, RockTexture.png,

; Texture coord @ vertex 0
SetTextureCoordinates, 0, 0.000000, 0.500000,
; Texture coord @ vertex 1
SetTextureCoordinates, 1, 0.500000, 0.500000,
; Texture coord @ vertex 2
SetTextureCoordinates, 2, 0.000000, 1.000000,
; Texture coord @ vertex 3
SetTextureCoordinates, 3, 0.500000, 1.000000,
; Texture coord @ vertex 4
SetTextureCoordinates, 4, 0.000000, 0.000000,
; Texture coord @ vertex 5
SetTextureCoordinates, 5, 0.500000, 0.000000,
; Texture coord @ vertex 6
SetTextureCoordinates, 6, 1.000000, 0.000000,
; Texture coord @ vertex 7
SetTextureCoordinates, 7, 1.000000, 1.000000,
```

There is a number of ways to arrange the texture coordinates. In this case, they are given in such a way that the 3 visible sides of of the cube are textured. The bottom and far side will however look silly. Normally, that does not matter as the object is viewed from almost only one direction, that is from the engineer's seat in a passing train.

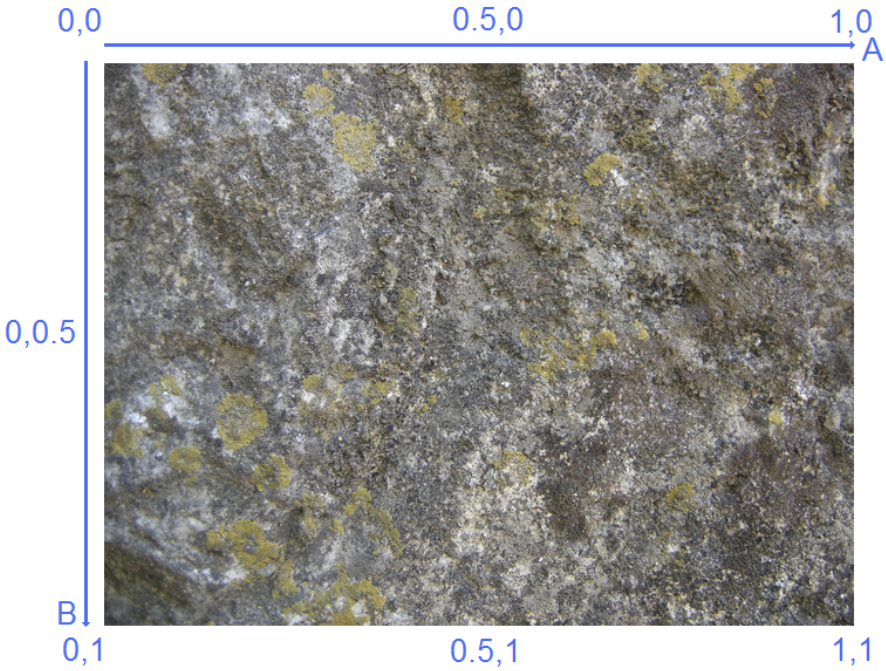


We take a closer look how the texture is applied to the cube above.

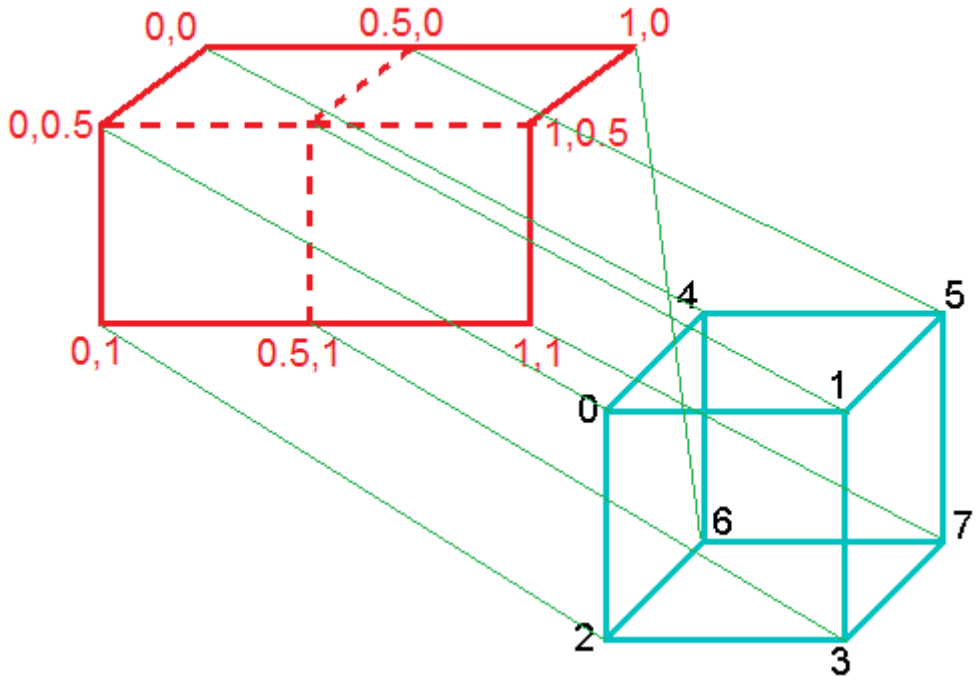
```
; Texture coord @ vertex 0
SetTextureCoordinates, 0, 0.000000, 0.500000,
; Texture coord @ vertex 1
SetTextureCoordinates, 1, 0.500000, 0.500000,
; Texture coord @ vertex 2
SetTextureCoordinates, 2, 0.000000, 1.000000,
; Texture coord @ vertex 3
SetTextureCoordinates, 3, 0.500000, 1.000000,
; Texture coord @ vertex 4
SetTextureCoordinates, 4, 0.000000, 0.000000,
; Texture coord @ vertex 5
SetTextureCoordinates, 5, 0.500000, 0.000000,
; Texture coord @ vertex 6
SetTextureCoordinates, 6, 1.000000, 0.000000,
; Texture coord @ vertex 7
SetTextureCoordinates, 7, 1.000000, 1.000000,
```

The cube itself has its 8 vertices numbered 0..7. For each of these vertices are given the coordinates of the point of the texture that should be “attached” to that vertex.

The coordinates of the texture is on the A axis from the left to the right, varying from 0..1, and on the B axis from the top to the bottom, varying from 0..1.



In the figure at the top of next side, we shown the texture (in red) folded, and how each vertex of the cube (in blue) is connected to a point of the texture. We see that the front and the top of the cube is easily connected to the texture. The right side however, is not connected to the texture in an easy way, as we have already used 3 of its 4 vertices (vertex # 3, 1 and 5) in specifying the texture for the front and the top of the cube. So we only can chose which point in the texture to connect with vertex 7. We chose the texture coordinate (1,1), which gives a reasonable result for the 3 faces that could be seen from the passing train. But this is only true if the texture is irregular as the pattern of the rock.



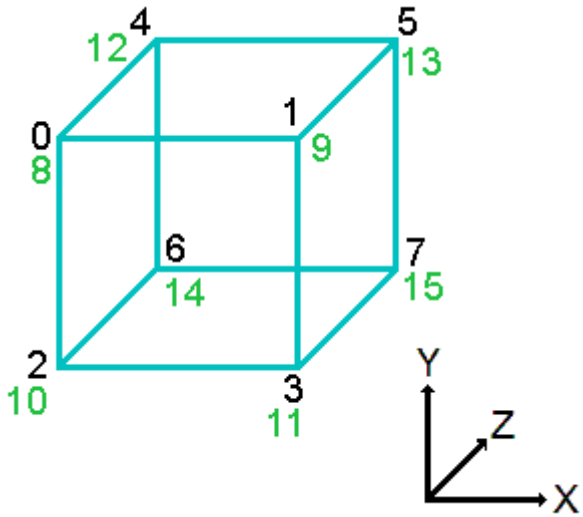
If these is a regular texture, such as a wooden box, we must resort to another method to get a reasonable result.

The problem is that for each vertex in the cube there can only be specified on texture coordinate. If we want to specify 2 different texture coordinates for one vertex in the cube, we seem to have to resort to making 2 cubes in the same place. That means we add another set of 8 vertices where the vertex coordinates are the same as for the 8 vertices we already have.

```
CreateMeshBuilder

  AddVertex, -2.000000,2.000000,1.000000, ; Vertex  0
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex  1
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex  2
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex  3
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex  4
  AddVertex, -1.000000,2.000000,2.000000, ; Vertex  5
  AddVertex, -2.000000,1.000000,2.000000, ; Vertex  6
  AddVertex, -1.000000,1.000000,2.000000, ; Vertex  7
  AddVertex, -2.000000,2.000000,1.000000, ; Vertex  8
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex  9
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex 10
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex 11
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex 12
```

```
AddVertex, -1.000000,2.000000,2.000000, ; Vertex 13
AddVertex, -2.000000,1.000000,2.000000, ; Vertex 14
AddVertex, -1.000000,1.000000,2.000000, ; Vertex 15

AddFace, 0, 1, 3, 2,       ; Face 0
AddFace, 1, 5, 7, 3,       ; Face 1
AddFace, 5, 4, 6, 7,       ; Face 2
AddFace, 4, 0, 2, 6,       ; Face 3
AddFace, 12, 13, 9, 8,     ; Face 4
AddFace, 10, 11, 15, 14,   ; Face 5
```

For the first 4 faces of the cube, that is the front, back, left and right faces, we use the same vertex numbers as before. But for the top and bottom face, we use the extra vertices 8..15 that we added. So now we have 2 vertices for each corner of the cube (the newly added in green numbers):



We still do specify 1 texture, which is the texture “RockTexture.png”:

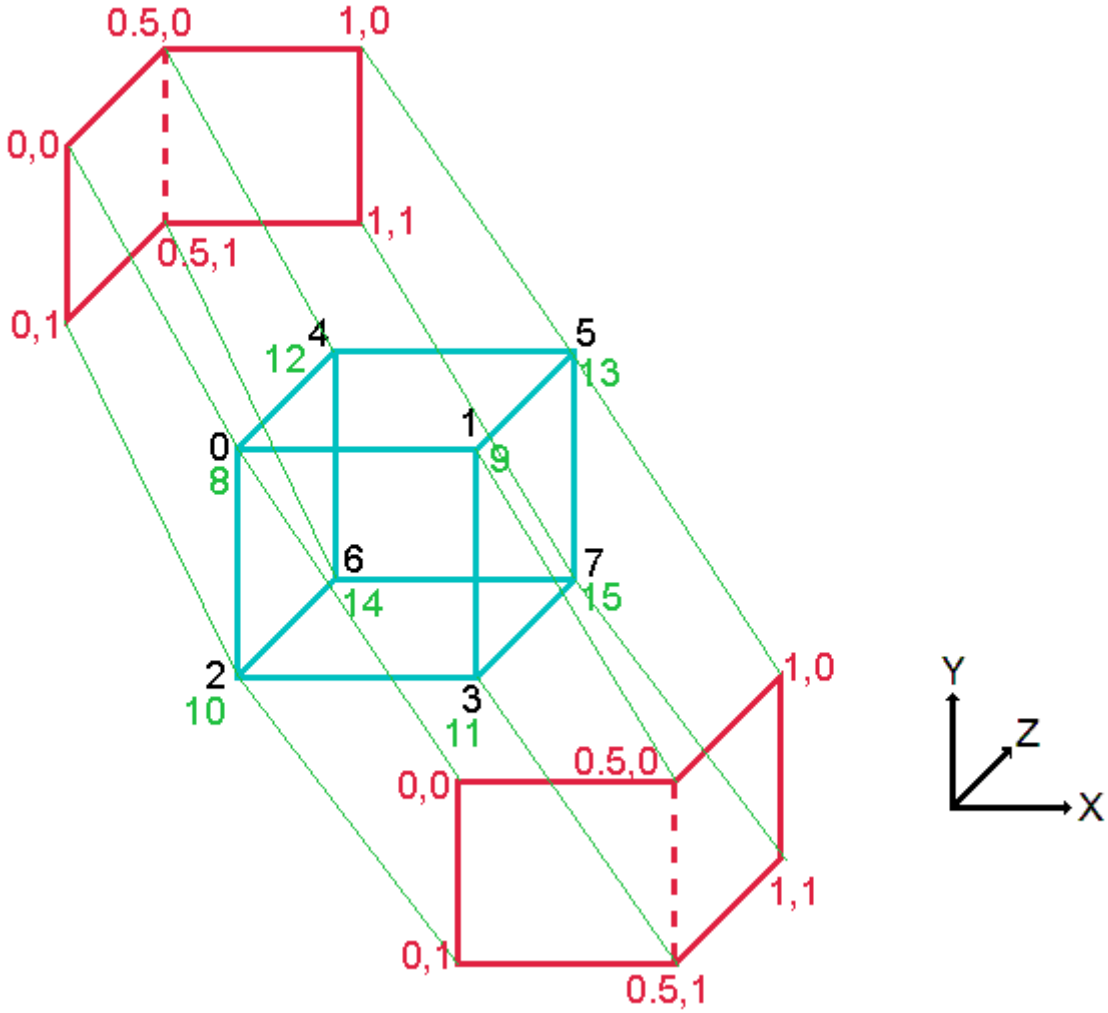
```
LoadTexture, RockTexture.png,
```

Then we come to the texture coordinates, which are to be set for all 16 vertices.

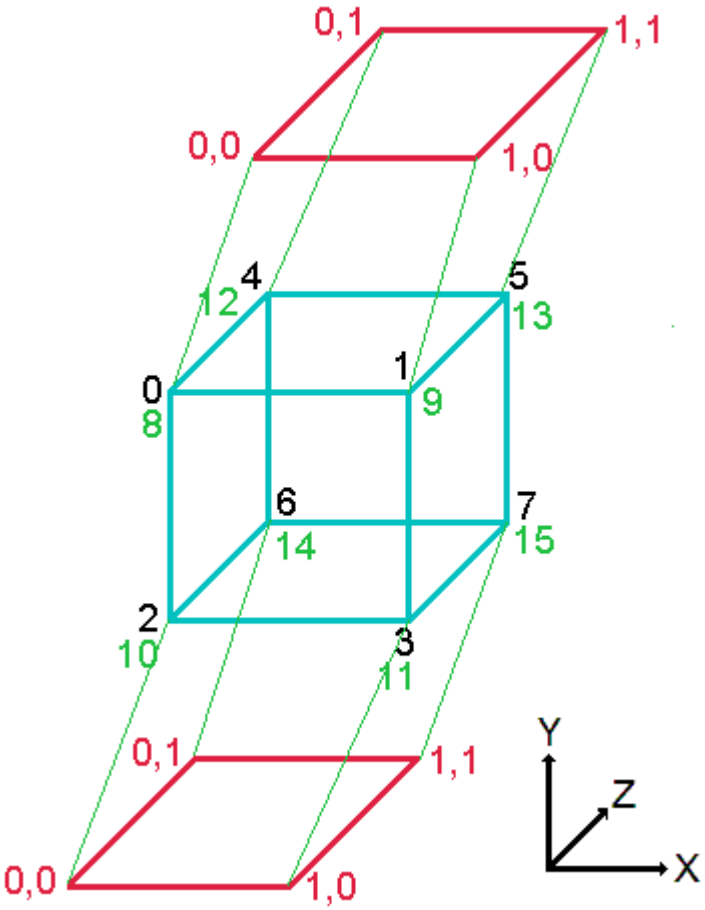
```
; Texture coord @ vertex  0
SetTextureCoordinates, 0,  0.000000, 0.000000,
; Texture coord @ vertex  1
SetTextureCoordinates, 1,  0.500000, 0.000000,
; Texture coord @ vertex  2
```

```
SetTextureCoordinates, 2,  0.000000, 1.000000,
; Texture coord @ vertex  3
SetTextureCoordinates, 3,  0.500000, 1.000000,
; Texture coord @ vertex  4
SetTextureCoordinates, 4,  0.500000, 0.000000,
; Texture coord @ vertex  5
SetTextureCoordinates, 5,  1.000000, 0.000000,
; Texture coord @ vertex  6
SetTextureCoordinates, 6,  0.500000, 1.000000,
; Texture coord @ vertex  7
SetTextureCoordinates, 7,  1.000000, 1.000000,
; Texture coord @ vertex  8
SetTextureCoordinates, 8,  0.000000, 0.000000,
; Texture coord @ vertex  9
SetTextureCoordinates, 9,  1.000000, 0.000000,
; Texture coord @ vertex 10
SetTextureCoordinates, 10, 0.000000, 0.000000,
; Texture coord @ vertex 11
SetTextureCoordinates, 11, 1.000000, 0.000000,
; Texture coord @ vertex 12
SetTextureCoordinates, 12, 0.000000, 1.000000,
; Texture coord @ vertex 13
SetTextureCoordinates, 13, 1.000000, 1.000000,
; Texture coord @ vertex 14
SetTextureCoordinates, 14, 0.000000, 1.000000,
; Texture coord @ vertex 15
SetTextureCoordinates, 15, 1.000000, 1.000000,
```

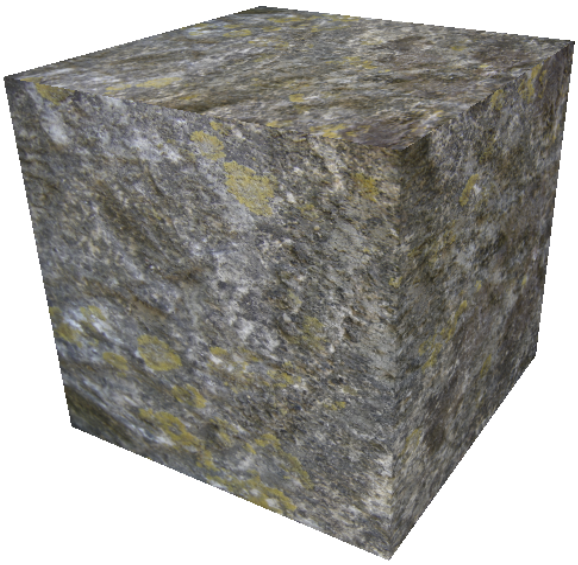
We wrap the texture around the cube's vertical sides (front, left back and right sides, attaching texture coordinates to the 8 vertices 0 to 7. The texture is attached to the front and right side, then repeated at the back and right sides:


0.5,0
1,0
0,0
1,1
0.5,1
0,1
4
12
5
13
1
9
0
8
6
14
7
15
2
10
3
11
1,0
0.5,0
0,0
1,1
0,1
0.5,1
Y
Z
X

To attach texture to the top and bottom face of the cube, we use the 8 new vertices 8..15



Now we have a cube with a proper texture at all 6 faces:



The csv object file for this cube is:

```
CreateMeshBuilder

  AddVertex, -2.000000,2.000000,1.000000, ; Vertex  0
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex  1
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex  2
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex  3
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex  4
  AddVertex, -1.000000,2.000000,2.000000, ; Vertex  5
  AddVertex, -2.000000,1.000000,2.000000, ; Vertex  6
  AddVertex, -1.000000,1.000000,2.000000, ; Vertex  7
  AddVertex, -2.000000,2.000000,1.000000, ; Vertex  8
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex  9
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex 10
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex 11
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex 12
  AddVertex, -1.000000,2.000000,2.000000, ; Vertex 13
  AddVertex, -2.000000,1.000000,2.000000, ; Vertex 14
  AddVertex, -1.000000,1.000000,2.000000, ; Vertex 15

  AddFace, 0, 1, 3, 2,       ; Face 0
  AddFace, 1, 5, 7, 3,       ; Face 1
  AddFace, 5, 4, 6, 7,       ; Face 2
  AddFace, 4, 0, 2, 6,       ; Face 3
  AddFace, 12, 13, 9, 8,     ; Face 4
  AddFace, 10, 11, 15, 14,   ; Face 5

  LoadTexture, RockTexture.png,

  ; Texture coord @ vertex  0
  SetTextureCoordinates, 0,  0.000000, 0.000000,
  ; Texture coord @ vertex  1
  SetTextureCoordinates, 1,  0.500000, 0.000000,
  ; Texture coord @ vertex  2
  SetTextureCoordinates, 2,  0.000000, 1.000000,
  ; Texture coord @ vertex  3
  SetTextureCoordinates, 3,  0.500000, 1.000000,
  ; Texture coord @ vertex  4
  SetTextureCoordinates, 4,  0.500000, 0.000000,
  ; Texture coord @ vertex  5
  SetTextureCoordinates, 5,  1.000000, 0.000000,
```

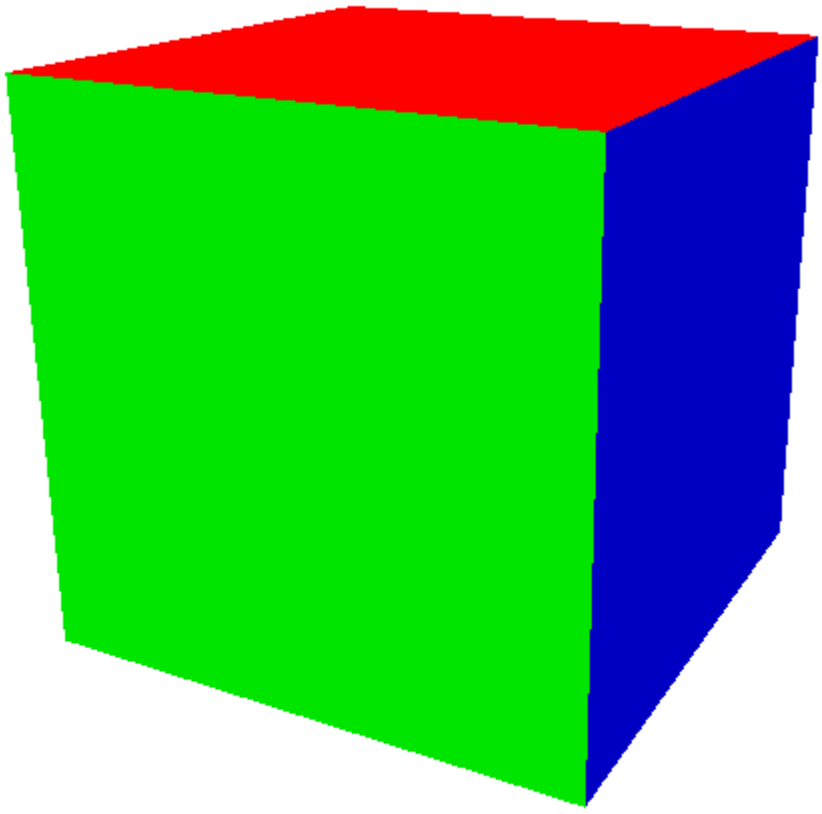
```
; Texture coord @ vertex  6
SetTextureCoordinates, 6,  0.500000, 1.000000,
; Texture coord @ vertex  7
SetTextureCoordinates, 7,  1.000000, 1.000000,
; Texture coord @ vertex  8
SetTextureCoordinates, 8,  0.000000, 0.000000,
; Texture coord @ vertex  9
SetTextureCoordinates, 9,  1.000000, 0.000000,
; Texture coord @ vertex 10
SetTextureCoordinates, 10, 0.000000, 0.000000,
; Texture coord @ vertex 11
SetTextureCoordinates, 11, 1.000000, 0.000000,
; Texture coord @ vertex 12
SetTextureCoordinates, 12, 0.000000, 1.000000,
; Texture coord @ vertex 13
SetTextureCoordinates, 13, 1.000000, 1.000000,
; Texture coord @ vertex 14
SetTextureCoordinates, 14, 0.000000, 1.000000,
; Texture coord @ vertex 15
SetTextureCoordinates, 15, 1.000000, 1.000000,
```

Chapter 3

Using colors

The first cube we made in chapter 1 had all sides green. For each CreateMeshBuilder statement there can only be one SetColor and/or LoadTexture statement, which will be applied to all faces declared in that CreateMeshBuilder section. We can use different colors on the faces of the cube, if we specify one CreateMeshBuilder section for each color..

We want to make a cube looking like this, with the front and back sides green, the right and left side blue and the top and bottom red:

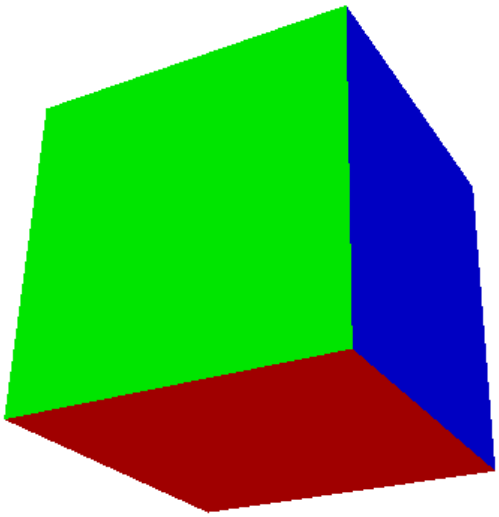


We use three CreateMeshBuilder sections, specify only the faces needed in each such a section, and apply a color to those faces. The csv object file will be like this:

```
CreateMeshBuilder  // Section for the green faces

  AddVertex, -2.000000,2.000000,1.000000, ; Vertex 0
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex 1
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex 2
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex 3
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex 4
  AddVertex, -1.000000,2.000000,2.000000, ; Vertex 5
```

```
  AddVertex, -2.000000,1.000000,2.000000, ; Vertex 6
  AddVertex, -1.000000,1.000000,2.000000, ; Vertex 7

  AddFace, 0, 1, 3, 2,  ; Face 0 - Front face
  AddFace, 5, 4, 6, 7,  ; Face 1 - Back face

  SetColor, 0, 255, 0, 255  ; Red Green Blue Alpha

CreateMeshBuilder  // Section for the blue faces

  AddVertex, -2.000000,2.000000,1.000000, ; Vertex 0
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex 1
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex 2
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex 3
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex 4
  AddVertex, -1.000000,2.000000,2.000000, ; Vertex 5
  AddVertex, -2.000000,1.000000,2.000000, ; Vertex 6
  AddVertex, -1.000000,1.000000,2.000000, ; Vertex 7

  AddFace, 1, 5, 7, 3,  ; Face 0 - Right side face
  AddFace, 4, 0, 2, 6,  ; Face 1 - Left side face

  SetColor, 0, 0, 255, 255  ; Red Green Blue Alpha

CreateMeshBuilder  // Section for the red faces

  AddVertex, -2.000000,2.000000,1.000000, ; Vertex 0
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex 1
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex 2
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex 3
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex 4
  AddVertex, -1.000000,2.000000,2.000000, ; Vertex 5
  AddVertex, -2.000000,1.000000,2.000000, ; Vertex 6
  AddVertex, -1.000000,1.000000,2.000000, ; Vertex 7

  AddFace, 4, 5, 1, 0,  ; Face 0 - Top face
  AddFace, 2, 3, 7, 6,  ; Face 1 - Bottom face

  SetColor, 0, 255, 0, 255  ; Red Green Blue Alpha
```

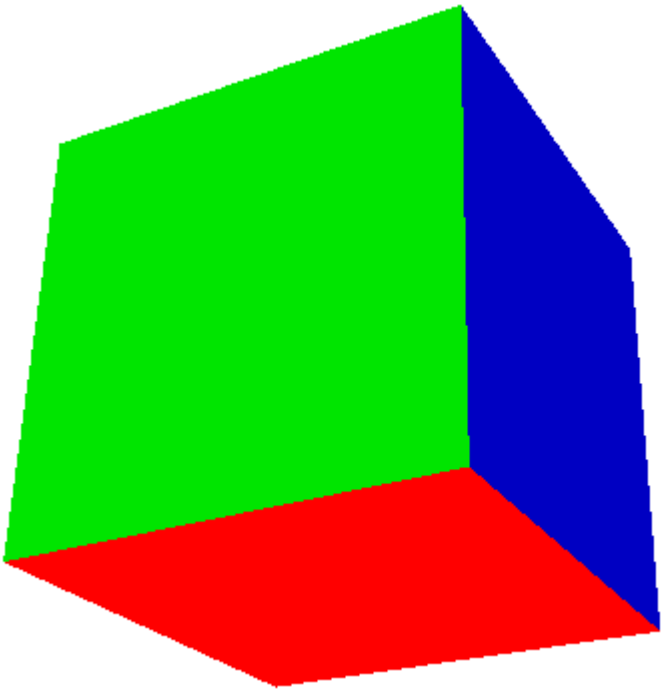
If we look at this cube from below, we see that its bottom is in shadow as the light source shines from above in the simulation.



If we however wants the bottom to be a light source itself, and as that not affected by shadows, we can use the SetEmissiveColor statement instead of the SetColor statement for the red color. We set red color to 255 in the SetEmissiveColor parameters in the CreateMeshBuilder section for the red faces:

```
  SetEmissiveColor, 255, 0, 0,  ;Red Green Blue
```

The rest of the contents of the csv object file for the cube remains the same. Now we will see that the red bottom of the cube is lit:



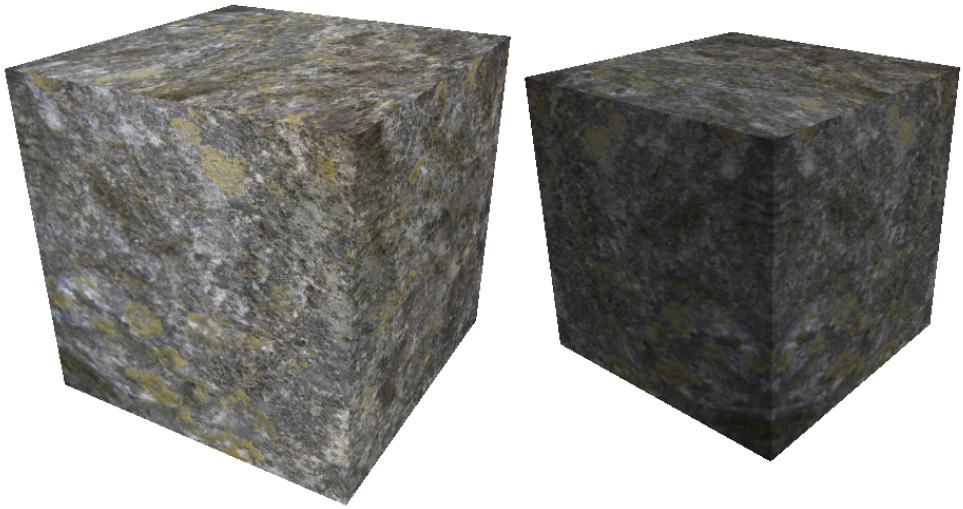
There is also a possibility to combine texture and material color. In the examples so far, we have used a colored materials on its own and textures on its own. This makes the texture colors to be shown as they are in the original texture file.

If we use a SetColor statement together with the LoadTexture statement, the color specified by SetColor will affect the texture. If we specify an all white color, the texture will not be affected, but any other color affects the texture.

If we want to make the texture darker, we change the material color to some shade of gray. Let's compare the cube with the rock texture with no color set and with a gray color set. We add the SetColor statement for the rock textured cube, and keeps the present LoadTexture statement.

```
SetColor, 155, 155, 155, 255,
```

This makes the cube shown to the right is a bit darker than the original one shown to the left:



BLANK PAGE

Chapter 4

Geometric shortcuts

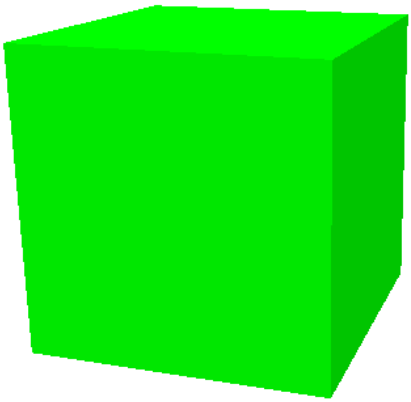
So far, we have created our objects by manually specifying the vertices och the mesh that builds the object. In the csv object file format, there are a few shortcuts. If one want to create a cuboid or a cylinder, there are easily used statements. There is also a statement to shear objects.

The Cube statement, that can be used not only for cubes but also for cuboids, takes parameters for Half Width, Half Height, and Half Depth. The cube or cuboid is made around the 3D coordinate system's origin, and has its 6 faces and 8 vertices.

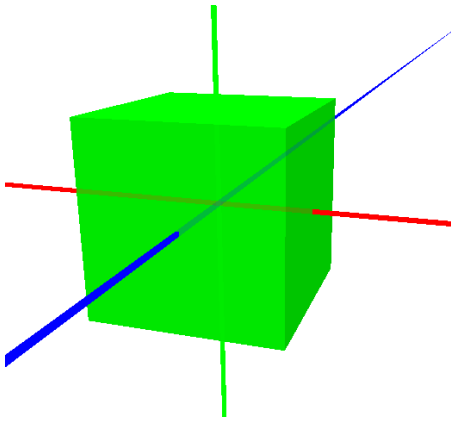
If we want to make a cube with the side 1 meter with all faces green, as we did manually in earlier examples in this book, we write:

```
CreateMeshBuilder

  Cube, 0.500000, 0.500000, 0.500000,  ; Cube w side 1m

  SetColor, 0, 255, 0, 255,  ; Green color to all faces
```

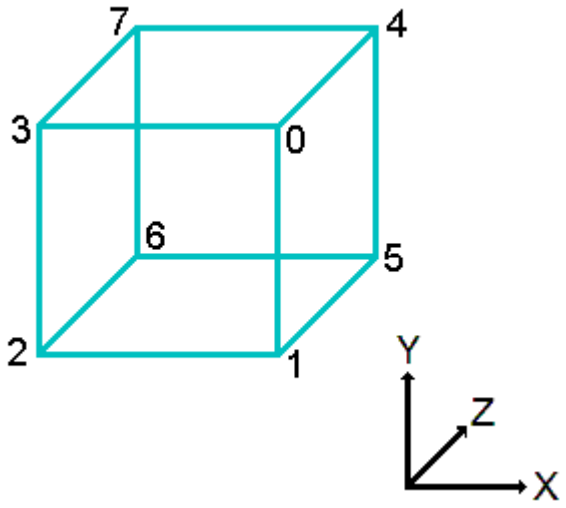
We achieve this result:



If we in the OpenBVE Object Viewer add the axes of the 3D coordinate system, we see that the center of the cube is at the origin of the coordinate system:



If we want the cube to have a texture surface, we need to add the LoadTexture statement and also to add texture coordinates using the SetTextureCoordinates statement. To properly set the texture coordinates, we need to know the numbers of the automatically created vertices so we can “attach” the texture properly to the mesh:



Making the multicolored cube we made in chapter 3 using the Cube statement is however not possible, because all faces are automatically generated for each cube mesh we define. To make the multicolored cube, we can only define 2 faces for each of the 3 cube meshes we need to create. In this case we have to stick to manually define vertices and faces.

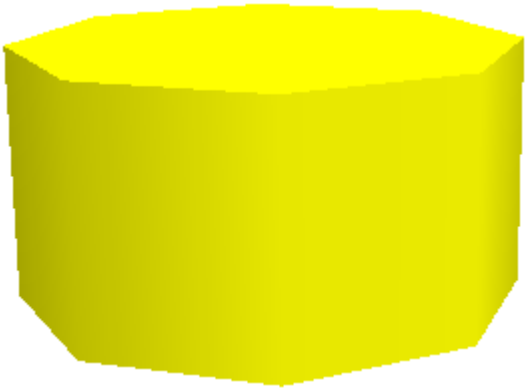
Another statement is the Cylinder statement. It is used for making cylinders, tapered cylinders and cones. It takes 4 parameters: The number of vertices to make the circular cross-section of the cylinder, Upper radius in meters, Lower radius in meters and the Height in meters. If the Lower radius is a positive number, the cylinder will be closed with a bottom cap, and the same condition applies to the top radius. If a negative number is used for any radius, the corresponding cap is omitted.

If either top or bottom radius is set to 0, we creates a cone. Just as with the Cube statement, the created geometry will be centered at the 3D coordinate system's origin.

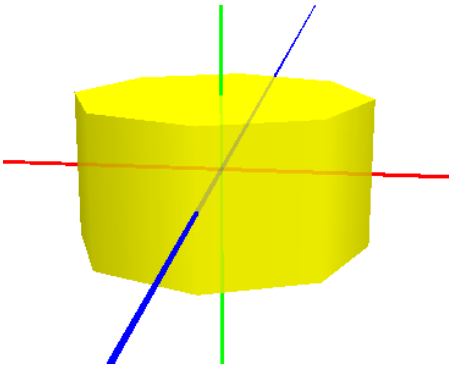
To make a cylinder with a height and a radius of 1 meter(diameter of 2 meters), with a top cap (but no bottom cap) and yellow color, we can write this code:

```
CreateMeshBuilder

  Cylinder, 8, 1, -1, 1,

  SetColor, 255, 255, 0, 255,
```

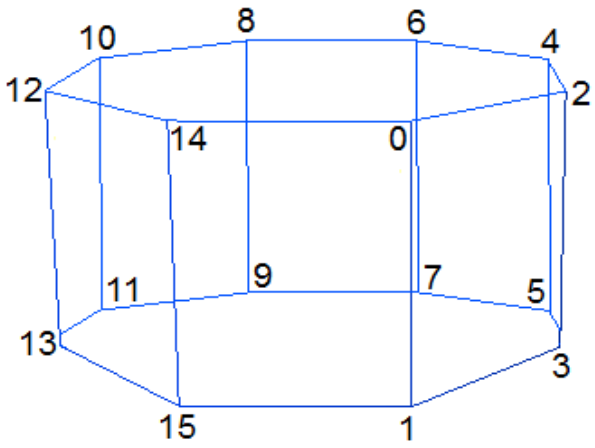
The result of this code is this octagonal “cylinder”:



If we want a more circular cross-section, we increase the number of vertices in the Cylinder statement's 1st parameter. Just as with the cube described earlier in this chapter, the cylinder is centered at the 3D coordinate system's origin.



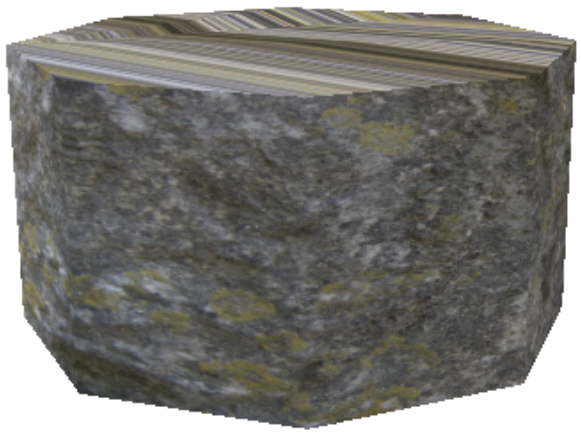
If we want the cylinder to have a texture surface, we need to add the LoadTexture statement and also to add texture coordinates using the SetTextureCoordinates statement. To properly set the texture coordinates, we need to know the numbers of the automatically created vertices so we can “attach” the texture properly to the mesh:



We can, as an example, wrap the stone texture around the octagonal “cylinder” by using this code:

```
CreateMeshBuilder

  Cylinder, 8, 1, -1, 1,

  LoadTexture, RockTexture.png

  SetTextureCoordinates,  0, 0.000000, 0.000000,
  SetTextureCoordinates,  1, 0.000000, 1.000000,
  SetTextureCoordinates,  2, 0.250000, 0.000000,
  SetTextureCoordinates,  3, 0.250000, 1.000000,
  SetTextureCoordinates,  4, 0.500000, 0.000000,
  SetTextureCoordinates,  5, 0.500000, 1.000000,
  SetTextureCoordinates,  6, 0.750000, 0.000000,
  SetTextureCoordinates,  7, 0.750000, 1.000000,
  SetTextureCoordinates,  8, 1.000000, 0.000000,
  SetTextureCoordinates,  9, 1.000000, 1.000000,
  SetTextureCoordinates, 10, 0.750000, 0.000000,
  SetTextureCoordinates, 11, 0.750000, 1.000000,
  SetTextureCoordinates, 12, 0.500000, 0.000000,
  SetTextureCoordinates, 13, 0.500000, 1.000000,
  SetTextureCoordinates, 14, 0.250000, 0.000000,
  SetTextureCoordinates, 15, 0.250000, 1.000000,
```

The result looks good at the sides, but the top looks really silly:



The problem is that we can not set the top face texture properly, as there already is a texture coordinate for every vertex the makes the top face. We need, as the example with to stone cube in chapter 2, make 2 identical cylinders at the same place, and define the side faces only for one of the cylinders, and the top face only for the other cylinder. This is not possible to do with the Cylinder statement, so we have in that case to resort to manually calculate the 3D coordinates for all 15 vertices.

Chapter 5

Manipulating objects

Using csv object files gives us a few statements that somehow manipulates already created objects. There are statements for moving the object's position, rotate or scale an object, and change an object's geometry by shearing it.

These statements all comes in two variants: One to manipulate just the (part of) the object that is define in a single CreateMeshBuilder section, one that affects all CreateMeshBuilder sections. The later statements all ends with “All”.

We start with the Translate (or TranslateAll) statement. It takes 3 parameters: Movement along the x-axis in meters, movement along the y-axis in meters and finally movement along the z-axis in meters.

To move the (part of an) object created in one CreateMeshBuilder section 5 meters away (on the z-axis) and move it down ½ meter (on the y-axis), we write in the end of the CreateMeshBuilder section:

```
Translate, 0, 5, 0.5,
```

To do the same movement for everything created in every CreateMeshBuilder section, we would write at the end of the csv object file:

```
TranslateAll, 0, 5, 0.5,
```

The next thing we can do is to rotate an object or all objects. The Rotate statement takes 4 parameters: x, y and z coordinates for the rotation axis, and the rotation angle in degrees counter-clockwise.

If we want to rotate the object 30 degrees clockwise around the z axis, we write:

```
Rotate, 0, 0, 1, -30,
```

To do the same rotation for everything created in every CreateMeshBuilder section, we would write at the end of the csv object file:

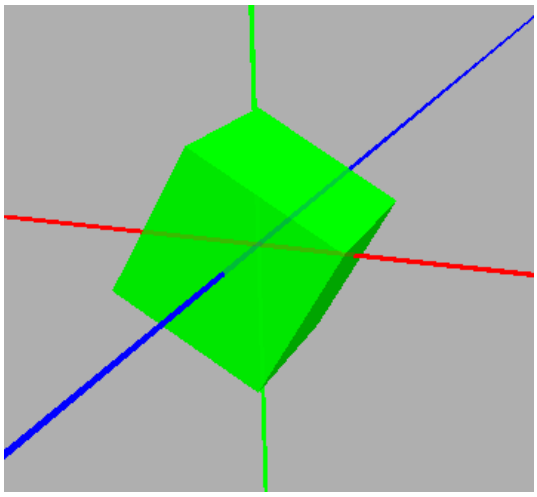
```
RotateAll, 0, 0, 1, -30,
```

We will do 2 examples of rotation, using the green cube from chapter 4.

We add the rotate statement to the code:

```
CreateMeshBuilder

  Cube, 0.500000, 0.500000, 0.500000,  ; Cube w side 1m

  SetColor, 0, 255, 0, 255,  ; Green color to all faces

  Rotate, 0, 0, 1, -30,
```

And the result is that the cube is rotated 30 degrees around the z axis:

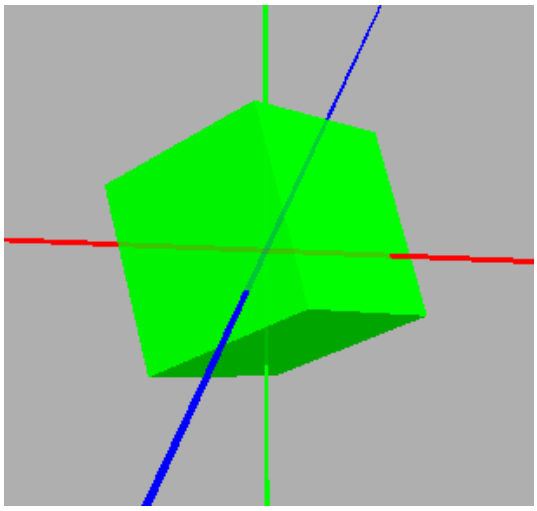


To make next example more complicated, we will define a turning axis that is not parallel to the axes of the 3D coordinate system. We will define a rotation axis that goes through the cube's bottom left front corner and through the upper right far corner. Then we will rotate the cube 45 degrees counter-clockwise.

The code for this rotation is:

```
CreateMeshBuilder

  Cube, 0.500000, 0.500000, 0.500000,  ; Cube w side 1m

  SetColor, 0, 255, 0, 255,  ; Green color to all faces

  Rotate, 1, 1, 1, 45,
```

Here a view of the result:



The next manipulation statements are Scale and ScaleAll. They scale an object (or part of it). The statement takes 3 parameters: Scaling along the x, y and z axes. The parameters are decimal numbers, there 50% of original size is 0.5, no scaling/100% is 1.0, double size/200% is 2.0 and so on.

To double the size of an object, we write the code:

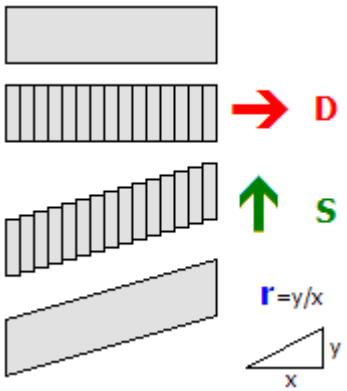
```
Scale, 2, 2, 2,
```

To do the same scaling for everything created in every CreateMeshBuilder section, we would write at the end of the csv object file:

```
ScaleAll, 2, 2, 2,
```

The last manipulation statements are Shear and ShearAll. This is doing what is called a shear mapping of the vertices for an object. The statement takes no less than 7 parameters: x, y and z coordinates of a vector $\vec{D}$, x, y and z coordinates of a vector $\vec{S}$ and a displacement ratio r.

This requires of course some explanation. To describe it in a 2D scenario, we can give a rectangular shape a slope along the vector $\vec{D}$ by moving the ends of it in the direction of the vector $\vec{S}$. The grade of the slope will be determined by the displacement ratio r.



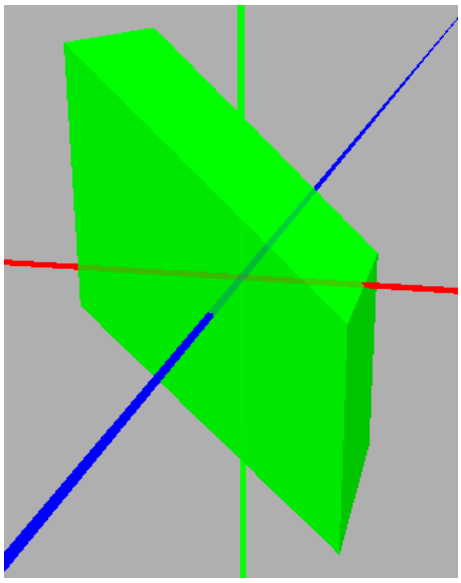
This illustration is a try to show the shear statement's function in a 2D case.

We make an 3D example by again using the green cube from chapter 4. We will put the vector $\vec{D}$ along the x axis but in the opposite direction (from right to left), and the vector $\vec{S}$ along the y axis (from bottom and up), and select a displacement ratio of 1:1, which gives us a 45 degrees slope.

The code for this is:

```
CreateMeshBuilder

  Cube, 0.500000, 0.500000, 0.500000,  ; Cube w side 1m

  SetColor, 0, 255, 0, 255,  ; Green color to all faces

  Shear, -1, 0, 0, 0, 1, 0, 1,
```

The result looks like this:



We can also notice from the picture that the displacement takes place around the objects center, which in this case is the 3D coordinate system's origin. The left part of the cube is lifted, the right part is lowered.

There is also, an mentioned, a ShearAll statement to affect all CreateMeshBuilder sections if put in the end of the csv object file. An example could be:

```
ShearAll, -.1, 0, 0, 0, 1, 0, 1,
```

Chapter 6

Transparency

There are 3 ways of achieving transparency, which are of different usability:

The first method is using the alpha channel setting for a material color. This allows for different levels of transparency by setting this parameter as a value in the range 0 (fully transparent) to 255 (opaque/no transparency). This affects the whole texture, every color becomes more or less transparent. Therefore, it is of no use when we want to make a complicated contour such as a tree. This method is anyway not recommended by the developers of the OpenBVE program, as it is very processing intensive and can slow down the frame-rate of the simulation seriously.

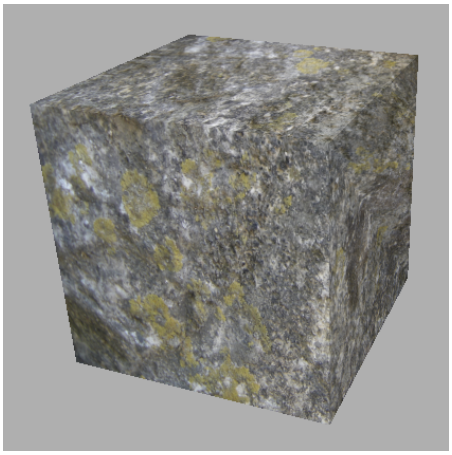
The second way is to use the SetDecalTransparentColor statement. By this full transparency is achieved by setting a specific color (RGB value) as transparent. Every part of the texture that we want transparent, we set to that color. This is much less processing intensive.

The third method is using the SetBlendMode statement. This makes an object fading out at a certain distance.

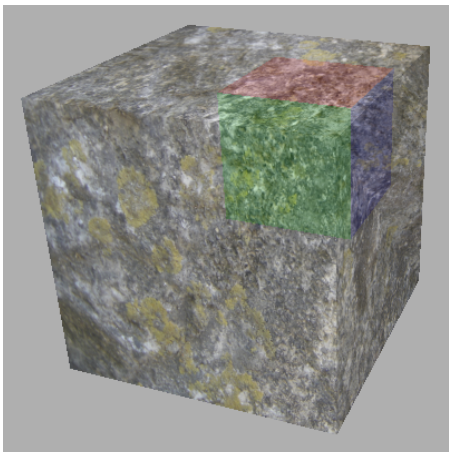
The alpha-channel method includes using the SetColor statement, in which one of the parameters is alpha channel. If we already have a SetColor statement we just change the alpha channel parameter to a desired value. If we have just a LoadTexture statement, we add a SetColor statement with white color (red, green and blue set to 255), which do not change the color of the texture, and use the alpha channel parameter to achieve some kind of partial transparency of the whole texture.

We try this with the cube that has a rock texture, and hide another multicolored cube behind it to demonstrate any transparency.

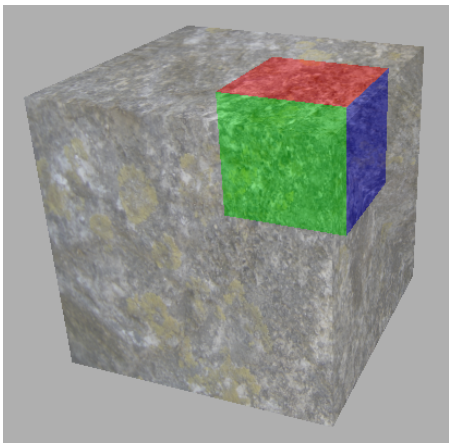
With the alpha channel value set to 255, full opacity, we see this:



Then we change the alpha channel value to 220, and now we suddenly see the other cube through the first one:



Lowering the alpha channel value even more, in this case to 150, we see even more of the multicolored cube hidden behind the first cube:



The csv object file code for the last example of the first cube is:

```
CreateMeshBuilder

  AddVertex, -2.000000,2.000000,1.000000, ; Vertex 0
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex 1
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex 2
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex 3
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex 4
  AddVertex, -1.000000,2.000000,2.000000, ; Vertex 5
  AddVertex, -2.000000,1.000000,2.000000, ; Vertex 6
  AddVertex, -1.000000,1.000000,2.000000, ; Vertex 7
```

```
  AddFace, 0, 1, 3, 2,  ; Face 0
  AddFace, 1, 5, 7, 3,  ; Face 1
  AddFace, 5, 4, 6, 7,  ; Face 2
  AddFace, 4, 0, 2, 6,  ; Face 3
  AddFace, 4, 5, 1, 0,  ; Face 4
  AddFace, 2, 3, 7, 6,  ; Face 5

  SetColor, 255, 255, 255, 150,

  LoadTexture, RockTexture.png,

  ; Texture coord @ vertex 0
  SetTextureCoordinates, 0, 0.000000, 0.500000,
  ; Texture coord @ vertex 1
  SetTextureCoordinates, 1, 0.500000, 0.500000,
  ; Texture coord @ vertex 2
  SetTextureCoordinates, 2, 0.000000, 1.000000,
  ; Texture coord @ vertex 3
  SetTextureCoordinates, 3, 0.500000, 1.000000,
  ; Texture coord @ vertex 4
  SetTextureCoordinates, 4, 0.000000, 0.000000,
  ; Texture coord @ vertex 5
  SetTextureCoordinates, 5, 0.500000, 0.000000,
  ; Texture coord @ vertex 6
  SetTextureCoordinates, 6, 1.000000, 0.000000,
  ; Texture coord @ vertex 7
  SetTextureCoordinates, 7, 1.000000, 1.000000,
```

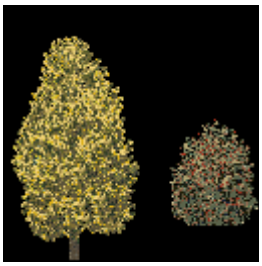
For the second cube, the multicolored, we use this code there we by using the TranslateAll statement has moved it behind the other one, when we load these 2 cubes at the same time in the OpenBVE Object Viewer:

```
CreateMeshBuilder  ; Section for the green faces

  AddVertex, -2.000000,2.000000,1.000000, ; Vertex 0
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex 1
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex 2
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex 3
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex 4
  AddVertex, -1.000000,2.000000,2.000000, ; Vertex 5
  AddVertex, -2.000000,1.000000,2.000000, ; Vertex 6
  AddVertex, -1.000000,1.000000,2.000000, ; Vertex 7
```

```
  AddFace, 0, 1, 3, 2,  ; Face 0 - Front face
  AddFace, 5, 4, 6, 7,  ; Face 1 - Back face

  SetColor, 0, 255, 0, 255  ; Red Green Blue Alpha

CreateMeshBuilder  ; Section for the blue faces

  AddVertex, -2.000000,2.000000,1.000000, ; Vertex 0
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex 1
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex 2
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex 3
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex 4
  AddVertex, -1.000000,2.000000,2.000000, ; Vertex 5
  AddVertex, -2.000000,1.000000,2.000000, ; Vertex 6
  AddVertex, -1.000000,1.000000,2.000000, ; Vertex 7

  AddFace, 1, 5, 7, 3,  ; Face 0 - Right side face
  AddFace, 4, 0, 2, 6,  ; Face 1 - Left side face

  SetColor, 0, 0, 255, 255  ; Red Green Blue Alpha

CreateMeshBuilder  ; Section for the red faces

  AddVertex, -2.000000,2.000000,1.000000, ; Vertex 0
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex 1
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex 2
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex 3
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex 4
  AddVertex, -1.000000,2.000000,2.000000, ; Vertex 5
  AddVertex, -2.000000,1.000000,2.000000, ; Vertex 6
  AddVertex, -1.000000,1.000000,2.000000, ; Vertex 7

  AddFace, 4, 5, 1, 0,  ; Face 0 - Top face
  AddFace, 2, 3, 7, 6,  ; Face 1 - Bottom face

  SetColor, 255, 0, 0, 255  ; Red Green Blue Alpha

TranslateAll, -1, -1, 3,  ; Move 1m left, 1m down 3m back
```

For the other method, using the SetDecalTransparentColor statement, we chose an example with two trees on a with black (RGB 0,0,0) background. The contours of the trees are complicated, and we cannot make them with a large number of vertices. Instead we put the rectangular picture on a rectangular “frame” with 4 vertices, and strip of anything black, the background, by setting black as transparent color.

We start with the texture that contains to trees och a black background. It is named “Tree_Pair.bmp” and has a height of 128 pixels and a width of 128 pixels. As 128 is a power of 2, that is 2^7, so the format is suitable for a texture file.



We use this csv object code by Mr. Anthony Bowden:

```
;Birmingham Cross-City BVE Route Object
;Cross-City South
;--------------------------------------
;By Anthony Bowden 2002-2008
;www.railsimroutes.co.uk
;
;Please acknowledge the original authors
;and keep this header intact
;
;See 'Readme.txt'

CreateMeshBuilder,

  AddVertex,-20,9,0,
  AddVertex,-20,0,0,
  AddVertex,0,-5,0,
  AddVertex,0,14,0,
  AddFace,3,2,1,0,

  GenerateNormals,
```

```
LoadTexture, Tree_Pair.bmp,

SetTextureCoordinates,0,00,0,
SetTextureCoordinates,1,00,1,
SetTextureCoordinates,2,01,1,
SetTextureCoordinates,3,01,0,

SetDecalTransparentColor,000,000,000,
```

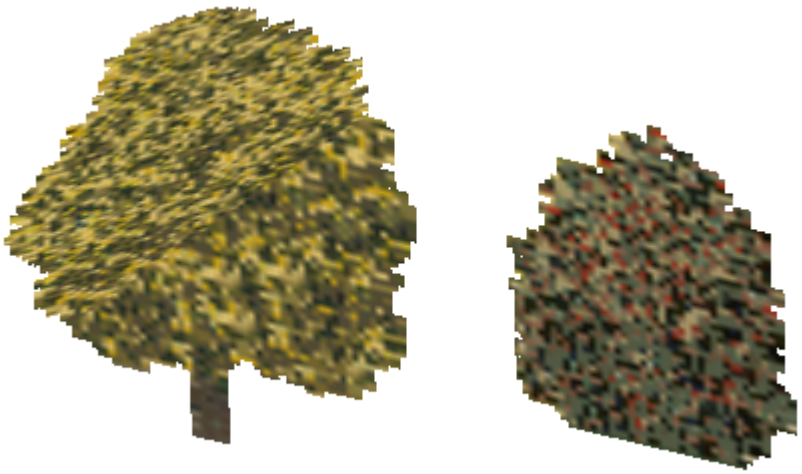
The code starts with Mr. Bowden's copyright notice (in Sweden, where this book is published, quoting of copyrighted material is allowed by the law).

The the 4 vertices of the “frame” for the trees are defined. The GenerateNormals statement is obsolete in OpenBVE, but is required if one wants BVE2 compatibility.

Then we use LoadTexture to add the picture of the trees, and SetTextureCoordinates to “attach” the picture to the “frame”.

The SetDecalTransparentColor statement sets everything black in the texture as transparent.

The final result is that the trees, with their complicated contours, have the black background removed as it is treated as transparent.



The third method of achieving some kind of transparency is the SetBlendMode statement. It takes 3 parameters: Blend Mode, Glow Half Distance and Glow Attenuation Mode.

These parameters determines how an object's color will be viewed. The default settings, that are used if this statement is not present, are equivalent with the setting:

```
SetBlendMode, Normal, 0,
```

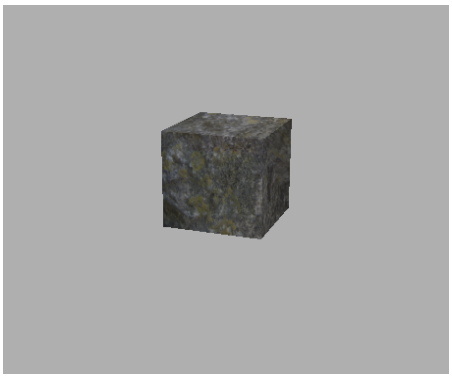
This means that the object's color/texture replaces the background's color/texture there the object is in front of the background seen from the viewer.

If we change the Glow Half Distance parameter to any other positive value, the object will fade away when coming closer to it. At the specified distance, the visibility of the object will be 50%. The distance can be from 1 to 4095 meters, and must be an integer.

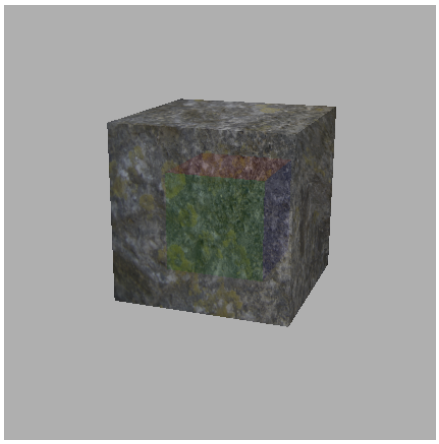
We set the Glow Half Distance parameter to 2 meters and see what happens:

```
SetBlendMode, Normal, 2,
```

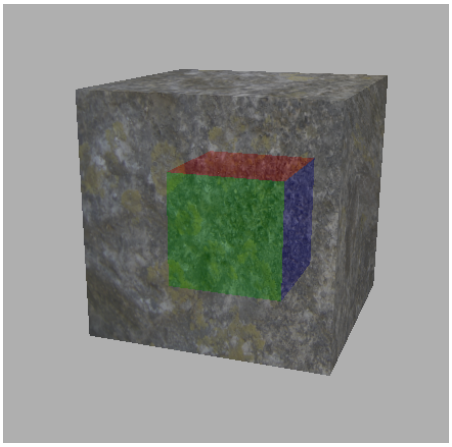
At a distance of 5.5 meters, the stone cube looks normal, we do not see the multicolored cube hidden behind it:



Coming closer, at a distance of 3 meters, we start to see the fade-away effect on the stone cube, and the multicolored cube behind it begins to become visible:



At a distance of 2 meters, the multicolored cube is seen even more:



Going even closer, the stone cube will fade away completely.

The complete csv object file code for the stone cube is now:

```
CreateMeshBuilder

  AddVertex, -2.000000,2.000000,1.000000, ; Vertex 0
  AddVertex, -1.000000,2.000000,1.000000, ; Vertex 1
  AddVertex, -2.000000,1.000000,1.000000, ; Vertex 2
  AddVertex, -1.000000,1.000000,1.000000, ; Vertex 3
  AddVertex, -2.000000,2.000000,2.000000, ; Vertex 4
  AddVertex, -1.000000,2.000000,2.000000, ; Vertex 5
  AddVertex, -2.000000,1.000000,2.000000, ; Vertex 6
  AddVertex, -1.000000,1.000000,2.000000, ; Vertex 7
```

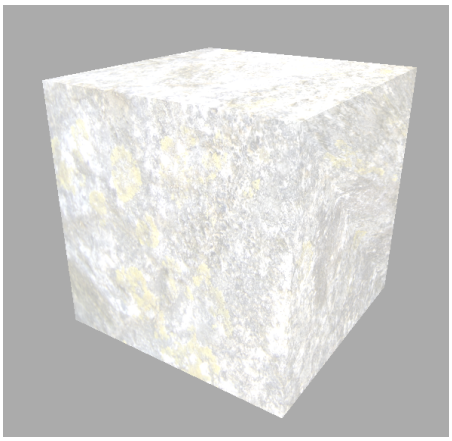
```
AddFace, 0, 1, 3, 2,  ; Face 0
AddFace, 1, 5, 7, 3,  ; Face 1
AddFace, 5, 4, 6, 7,  ; Face 2
AddFace, 4, 0, 2, 6,  ; Face 3
AddFace, 4, 5, 1, 0,  ; Face 4
AddFace, 2, 3, 7, 6,  ; Face 5

SetColor, 150, 150, 150, 255,

LoadTexture, RockTexture.png,

SetBlendMode, Normal, 2,

; Texture coord @ vertex 0
SetTextureCoordinates, 0, 0.000000, 0.500000,
; Texture coord @ vertex 1
SetTextureCoordinates, 1, 0.500000, 0.500000,
; Texture coord @ vertex 2
SetTextureCoordinates, 2, 0.000000, 1.000000,
; Texture coord @ vertex 3
SetTextureCoordinates, 3, 0.500000, 1.000000,
; Texture coord @ vertex 4
SetTextureCoordinates, 4, 0.000000, 0.000000,
; Texture coord @ vertex 5
SetTextureCoordinates, 5, 0.500000, 0.000000,
; Texture coord @ vertex 6
SetTextureCoordinates, 6, 1.000000, 0.000000,
; Texture coord @ vertex 7
SetTextureCoordinates, 7, 1.000000, 1.000000,
```

The code of the multicolored cube is the same as in the earlier example at page 28-29.

If we however sets the blending mode to “Additive”, the light of the background's color/texture will be added to that of the object. That means that if we have a black background, nothing will be added and the object looks the way we have created it. If we have any another background, the object will be brighter than we originally created it. If the background is white, the whole object will be white. This mode is set by using the statement:

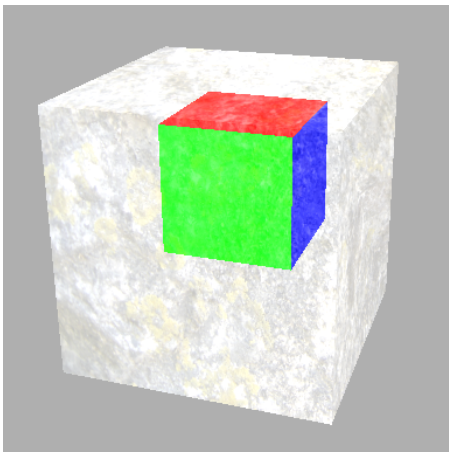
```
SetBlendMode, Additive, 0
```

We view the darker cube with a light gray background, the adding of the gray stone texture and the light gray background almost makes the cube white.



If we change the background to white, the cube will also be white and so not visible on the white background.

If we again put the multicolored cube behind the stone cube, we will see the multicolored cube through the stone cube at any distance. That is because the multicolored cube's clear colors will be added to those of the stone cube's texture:



If we use the Glow Half Distance parameter together with the Additive mode, the fade away effect on the stone cube will become visible as in the example with Normal blend mode.

```
SetBlendMode, Additive, 2
```

However, the fade away effect will be most visible where the gray background is, because the multicolored cube is already seen in bright colors.

There is a last parameter to discover related to the SetBlendMode statement. That is the parameter Glow Attenuation Mode. It can take 2 values: DivideExponent2 or DivideExponent4. If this parameter is not set, the default value is DivideExponent4.

```
SetBlendMode, Normal, 2, DivideExponent2
```

If we set the Glow Attenuation Mode parameter to DivideExponent2, the fade-away effect of the stone cube in earlier example will become more gradually, and will start at a greater distance, compared to if the parameter is DivideExponent4.

Finally, there is a warning from the OpenBVE developers that is version 2 of OpenBVE, only additive Blend Mode will be supported, and that the Glow Attenuation Mode parameter will be abolished. At the moment this text is written, the current version of OpenBVE is 1.4.2.0.

Chapter 8

Statements in Alphabetical Order

AddFace

```
AddFace, v1, v2, v3, … vn,
```

Variable	Type	Notice
Vertex numbers in clockwise order	Integer from 0 and up	At least 3 vertices are needed

The AddFace statement defines a face, which is a surface “stretched” between vertices in a mesh. Faces should be made up of polygons of 3 or more vertices, and the order of vertices should be clockwise. Face polygons should not be self-intersecting.

The parameters are the number of those vertices that should make the face. The vertex number is the order number in which it is defined within the CreateMeshBuilder section.

The face surface is not visible just because it is defined as a face, but the face must be colored by the SetColor (or SetEmissiveColor) statement, or given a texture by the LoadTexture statement. All faces defined in the same CreateMeshBuilder section will get the same color or texture (if any is defined).

Also, the face is only visible from one side, which is the side where the vertices number parameters are in clockwise order.

In those rare cases a face needs to be visible from both sides, the AddFace2 statement can be used.

Example 1:

```
AddFace, 2, 4, 3, 1,
```

This describes a face made of 4 vertices.

AddFace2

```
AddFace2, v1, v2, v3, … vn,
```

Variable	Type	Notice
Vertex numbers in clockwise order	Integer from 0 and up	At least 3 vertices are needed

The AddFace2 statement defines a face, which is a surface “stretched” between vertices in a mesh. Faces should be made up of polygons of 3 or more vertices, and the order of vertices should be clockwise. Face polygons should not be self-intersecting.

The parameters are the number of those vertices that should make the face. The vertex number is the order number in which it is defined within the CreateMeshBuilder section. The vertices must be stated in clockwise order when the face is seen from the side which should be the “front” side of the face..

The face surface is not visible just because it is defined as a face, but the face must be colored by the SetColor (or SetEmissiveColor) statement, or given a texture by the LoadTexture statement. All faces defined in the same CreateMeshBuilder section will get the same color or texture (if any is defined).

The face made with the SetFace2 statement will be visible from both sides. The lighting conditions will be properly shown at what is the face's front side, but may not be shown properly at the “back” side.

Use the AddFace2 statement only when the use of the AddFace statement is not usable.

Example 1:

```
AddFace2, 2, 4, 3, 1,
```

This describes a face made of 4 vertices, which will be visible from both sides.

AddVertex

```
AddVertex, X, Y ,Z, nX, nY, nZ,
```

Variable	Type	Notice
X coordinate	Decimal number	Positive right of the origin, negative left of the origin
Y coordinate	Decimal number	Positive up from the origin, negative down from the origin
Z coordinate	Decimal number	Positive behind the origin, negative in front of the origin
Normal X coordinate	Decimal number	Omit for automatic normal calculation
Normal Y coordinate	Decimal number	Omit for automatic normal calculation
Normal Z coordinate	Decimal number	Omit for automatic normal calculation

The AddVertex statement defines a vertex in the mesh that makes (a part) of an object. The 3 first parameters are the x, y and z coordinates for the vertex, the 3 last parameters, that are optional, are the x, y and z coordinates for the normal of that vertex. If the 3 last parameters are omitted, the normals are automatically calculated by OpenBVE.

The vertices are referred to by the order (from 0 and up) in which they are defined within a CreateMeshBuilder section.

Example 1:

```
AddVertex, 1, 0, 1,
```

CreateMeshBuilder

```
CreateMeshBuilder,
```

The CreateMeshBuilder statement starts a section in a csv object file, in which a mesh, faces and the face properties are defined.

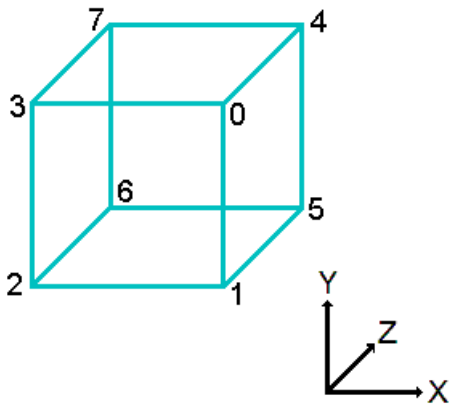
A csv object file must contain one but may contain more CreateMeshBuilder statements.

Cube

```
Cube, Half width, half height, half depth
```

Variable	Type	Notice
Half width	Decimal number	Half the width in meters (along the x axis)
Half height	Decimal number	Half the height in meters (along the y axis)
Half depth	Decimal number	Half the depth in meters (along the z axis)

The Cube statement defines a cube or cuboid with the 3D coordinate system's origin in its center. The statement automatically defines 8 vertices and 6 faces, which must be considered if referring to the order number of faces and vertices in the same CreateMeshBuilder section. The vertices are created in this order:



Example 1:

```
Cube, 1, 1, 1,
```

This describes a cube with the side 1 meter.

Example 2:

```
Cube, 2, 1, 3,
```

This describes a cuboid with the width 2 meters, the height 1 meter and the depth 3 meters.

Cylinder

```
Cylinder, vertices, U radius, L radius, Height,
```

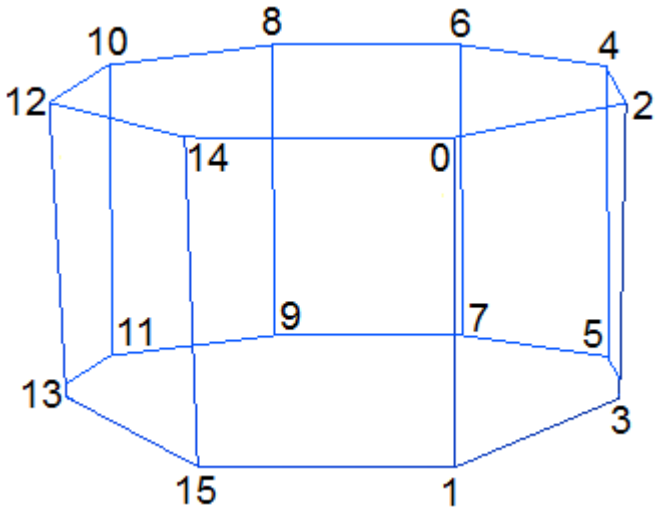
Variable	Type	Notice
Vertices	Positive integer	Number of vertices for the cylinder's base
Upper radius	Decimal number	Upper radius in meters. If negative, no top cap face will be created
Lower radius	Decimal number	Lower radius in meters. If negative, no bottom cap face will be created.
Height	Decimal number	The height in meters. If negative, the cylinder's faces will be visible from the inside, if positive, they will be visible from the outside.

The Cylinder statement defines a cylinder or other frustum. The cylinder is not perfectly round, but its base will be a polygon with the number of vertices described in the first parameter. If the upper and lower radius are equal, we get what can be approximated as a cylinder, else we get a frustum or, in case one of the radii is 0, a cone.

If the height is set to a positive value, the faces of the cylinder will be visible from the outside. If the height is a negative value, the faces will only be visible from within the cylinder.

The created object is centered around the 3D coordinate system's origin.

The statement automatically defines a number of vertices and faces, which must be considered if referring to the order number of faces and vertices in the same CreateMeshBuilder section. The vertices, if the parameter Vertices is set to 8, are created in this order:



Example 1:

```
Cylinder, 8, 1, -1, 1,
```

This describes creating a cylinder with 8 vertices in the base (which formally makes an octagonal cross-section), the upper radius 1 m, lower radius also 1 m (the negative radius value indicates that no bottom cap face is created), and a height of 1 m.

GenerateNormals

```
GenerateNormals,
```

The GenerateNormals statement is used in csv object files BVE 2 and BVE 4 to automatically generate normals for vertices defined with the AddVertex statement.

It has no function in OpenBVE; normals are generated automatically if they are not explicitly defined in the AddVertex statements.

Example 1:

```
CreateMeshBuilder,

  AddVertex,-0.4,3.7,0,
  AddVertex,-0.4,3.7,-13,
  AddVertex,-0.9,3.55,0,
  AddVertex,-0.9,3.55,-13,
  AddVertex,-1.1,3.35,0,
  AddVertex,-1.1,3.35,-13,
  AddVertex,-1.1,0,0,
  AddVertex,-1.1,0,-13,
  AddFace,0,1,3,2,
  AddFace,2,3,5,4,
  AddFace,4,5,7,6,

  GenerateNormals,
```

This describes the beginning of a csv object file for BVE2 or BVE 4 with 8 vertices, 3 faces, and the GenerateNormals statement after those other definitions.

LoadTexture

```
LoadTexture, Daytime texture, Nighttime texture
```

Variable	Type	Notice
Daytime texture	Character string	File type could be Windows bitmap (.bmp) or Portable Network Graphics (.png)
Nighttime texture	Character string	File type could be Windows bitmap (.bmp) or Portable Network Graphics (.png)

The LoadTexture statement takes 2 parameters, which both are file names of a texture. It may include a relative path, should the texture image file not be in the same folder as the csv object file.

Normally, only the Daytime texture is stated. Nighttime textures are only used when making train interior and exterior definitions, but normally never in a route.

The file type could be a Windows bitmap (.bmp) or a Portable Network Graphics (.png) file.

The height and width in pixels of the texture file should be a power of 2. Such numbers from 2^0 to 2^{10} are: 1, 2, 4, 8, 16, 32, 64, 128, 256, 516 and 1024.

Example 1:

```
LoadTexture, WoodenWall.png,
```

This loads a texture file WoodenWall.png, that can be applied to one or more faces, within a CreateMeshBuilder section, by a set of SetTextureCoordinates statements.

Rotate

```
Rotate, Ax, Ay, Az, Angle,
```

Variable	Type	Notice
Ax	Decimal number	The x component of the rotation axis $\vec{A}$
Ay	Decimal number	The y component of the rotation axis $\vec{A}$
Az	Decimal number	The z component of the rotation axis $\vec{A}$
Angle	Decimal number	The rotation in degrees counter-clockwise

The Rotate statement rotates all vertices created so far in a CreateMeshBuilder section around a rotation axis. The rotation axis is defined by a vector $\vec{A}$. The direction of the rotational axis vector is set by its x, y, z components.

The Rotate statement takes 4 parameters: The x, y, z component of the vector $\vec{A}$ which defines the rotational axis, and the rotation in degrees around the mentioned rotation axis.

The rotation is counter-clockwise if the Angle parameter is positive, and clockwise if that parameter has a negative value.

The Rotate statement affects all vertices defined so far in the CreateMeshBuilder section where it is used. More than one Rotate statement can be used in a single CreateMeshBuilder section.

If a rotation around one of the axes of the 3D coordinate system is wanted, set the corresponding rotation axis parameter to 1 and the other 2 rotation axis parameters to 0.

Example 1:

```
Rotate, 0, 0, 1, -45,
```

This describes a 45 degrees clockwise rotation around the 3D coordinate system's z axis.

RotateAll

```
Rotate, Ax, Ay, Az, Angle
```

Variable	Type	Notice
Ax	Decimal number	The x component of the rotation axis $\vec{A}$
Ay	Decimal number	The y component of the rotation axis $\vec{A}$
Az	Decimal number	The z component of the rotation axis $\vec{A}$
Angle	Decimal number	The rotation in degrees counter-clockwise

The RotateAll statement do the same thing as the Rotate statement, **but** affects all CreateMeshBuilder sections written before the RotateAll statement. The 4 parameters are also the same as for the Rotate statement.

Example 1:

```
Rotate, 0, 1, 0, -45,
```

This describes a 45 degrees clockwise rotation around the 3D coordinate system's y axis.

Scale

```
Scale, x, y, z,
```

Variable	Type	Notice
x	Decimal number	Scale factor along the x axis
y	Decimal number	Scale factor along the y axis
z	Decimal number	Scala factor along the z axis

The Scale statement scales an object (or part of it). It takes 3 parameters: x that is the scale factor along the x axis, y that is the scale factor along the y axis, and z that is the scale factor along the z axis.

The scale factor tells how many times the original size the scaled object should be. A scale factor of 1 means the original size, a scale factor of 0.5 means half the original size, a scale factor of 2 means double size, and so on.

The Translate statement affects all vertices defined so far in the CreateMeshBuilder section where it is used. More than one Translate statement can be used in a single CreateMeshBuilder section.

Example 1:

```
Scale, 2, 2, 2,
```

This describes a scaling of all distances between vertices created so far in the CreateMeshBuilder to double the original size.

ScaleAll

```
ScaleAll, x, y, z,
```

Variable	Type	Notice
x	Decimal number	Scale factor along the x axis
y	Decimal number	Scale factor along the y axis
z	Decimal number	Scala factor along the z axis

The ScaleAll statement do the same thing as the Scale statement, **but** affects all CreateMeshBuilder sections written before the ScaleAll statement. The 3 parameters are also the same as for the Scale statement.

Example 1:

```
ScaleAll, 0.5, 0.5, 0.5,
```

This describes a scaling of all distances between vertices in all CreateMeshBuilder sections before the ScaleAll statement to half the original size.

SetBlendMode

```
SetBlendMode, Blend mode, Glow half distance,
    Glow attenuation mode,
```

Variable	Type	Notice
Blend mode	Text string	Can take 2 values: “Normal” or “Additive”. Default value is “normal”.
Glow half distance	Integer	To disable glow attenuation, set to 0. Else the value can be 1 to 4095. Default value is 0.
Glow attenuation mode	Text string	Can take 2 values: “DivideExponent2” or “DivideExponent4”. Default value is “DivideExponent4”

The SetBlendMode sets the blend mode for all faces in the CreateMeshBuilder section it is used. It takes 3 parameters: Blend Mode, Glow Half Distance and Glow Attenuation Mode.

For a further description on how the SetBlendMode statement works, please refer to pages 44-48.

SetColor

```
SetColor, Red, Green, Blue, Alpha channel,
```

Variable	Type	Notice
Red	Integer 0 to 255	
Green	Integer 0 to 255	
Blue	Integer 0 to 255	
Alpha channel	Integer 0 to 255	

The SetColor parameter sets the color of all faces created in a CreateMeshBuilder Section.

The SetColor statement takes 4 parameters: The first parameter determines the amount of red, the second the amount of green, the third the amount of blue, and the fourth the alpha channel.

All colors can be made up of a proper mix of red, green and blue. The alpha channel determines the amount of transparency of the color. As we want no transparency, we set the value to 255 (a value of 0 will mean full transparency). The value for each of these 4 parameters can range from 0 to 255.

Example 1:

```
SetColor, 255, 255, 255, 255,
```

This describes white color and no transparency.

SetDecalTransparentColor

SetDecalTransparentColor, Red, Green, Blue,

Variable	Type	Notice
Red	Integer 0 to 255	
Green	Integer 0 to 255	
Blue	Integer 0 to 255	

The SetDecalTransparentColor statement makes one specified color in a texture bitmap transparent.

Set SetDecalTransparentColor statement takes 3 parameters which specifies the color that should be rendered as transparent: The first parameter determines the amount of red, the second the amount of green, the third the amount of blue.

Example 1:

```
SetDecalTransparentColor, 0, 0, 255,
```

This describes that all fully blue pixels in the texture image should be considered transparent.

Please notice that colors which may look the same as the “transparent” color, but have different color values, will be rendered. In this example, the RGB color 0,0,254 will look fully blue too, but will not be considered transparent.

SetEmissiveColor

```
SetEmissiveColor, Red, Green, Blue,
```

Variable	Type	Notice
Red	Integer 0 to 255	
Green	Integer 0 to 255	
Blue	Integer 0 to 255	

The SetEmissiveColor parameter sets the emissive color of all faces created in a CreateMeshBuilder Section.

The SetColor statement takes 3 parameters: The first parameter determines the amount of red, the second the amount of green, and the third the amount of blue.

All colors can be made up of a proper mix of red, green and blue. The value for each of these 3 parameters can range from 0 to 255.

The difference between the SetColor and SetEmissiveColor statements are that colors set by SetColor are affected by the light conditions. The color set by SetColor may be bright if in the light or it may be in the shadow. The SetEmissiveColor statement gives a color that is always bright independent of the lighting conditions. That is usable for objects the emits light such as signals, lighting armature etc.

Example 1:

```
SetEmissiveColor, 255, 0, 0,
```

This describes a red emissive color set for all faces in a CreateMeshBuilder section.

SetTextureCoordinates

```
SetTextureCoordinates, Vertex number, A, B,
```

Variable	Type	Notice
Vertex	Integer	
A coordinate	Decimal number	
B coordinate	Decimal number	

The SetTextureCoordinates statement is used to “attach” a texture to one or more faces in a CreateMeshBuilder section.

The SetTextureCoordinates statement takes 3 parameters: The first is the vertex number, the second is the A coordinate of the texture to “attach” to that vertex, and the third is the B coordinate of the texture to attach to that vertex.

For a further description on how the SetTextureCoordinates statement works, please refer to pages 10-18.

Shear

```
Shear, Dx, Dy, Dz, Sx, Sy, Sz, r
```

Variable	Type	Notice
Dx	Decimal number	x coordinate for vector $\vec{D}$
Dy	Decimal number	y coordinate for vector $\vec{D}$
Dz	Decimal number	z coordinate for vector $\vec{D}$
Sx	Decimal number	x coordinate for vector $\vec{S}$
Sy	Decimal number	y coordinate for vector $\vec{S}$
Sz	Decimal number	z coordinate for vector $\vec{S}$
r	Decimal number	Displacement ratio

The Shear statement shears an object in the direction of a vector $\vec{S}$ along a vector $\vec{D}$. The displacement ratio gives the amount of sharing in the direction of $\vec{S}$. The Shear statement affects all vertices created so far in a CreateMeshBuilder section.

The Shear statement takes no less than 7 parameters: The first 3 parameters are the x, y, and z coordinates of the vector $\vec{D}$, the next 3 parameters are the x, y, and z coordinates of the vector $\vec{S}$, and the seventh parameter is the displacement ratio.

For a further description on how the Shear statement works, please refer to pages 34-36.

ShearAll

```
ShearAll, Dx, Dy, Dz, Sx, Sy, Sz, r
```

Variable	Type	Notice
Dx	Decimal number	x coordinate for vector $\vec{D}$
Dy	Decimal number	y coordinate for vector $\vec{D}$
Dz	Decimal number	z coordinate for vector $\vec{D}$
Sx	Decimal number	x coordinate for vector $\vec{S}$
Sy	Decimal number	y coordinate for vector $\vec{S}$
Sz	Decimal number	z coordinate for vector $\vec{S}$
r	Decimal number	Displacement ratio

The ShearAll statement do the same thing as the Shear statement, **but** affects all CreateMeshBuilder sections written before the ShearAll statement. The 7 parameters are also the same as for the Shear statement.

Translate

```
Translate, x, y, z,
```

Variable	Type	Notice
x	Decimal number	Translation along the x axis in meters.
y	Decimal number	Translation along the y axis in meters.
z	Decimal number	Translation along the z axis in meters.

The Translate statement moves an object in the 3D space. The statement takes 3 parameters: x that is the movement along the x axis in meters where a positive number moves the object to the right and a negative number to the left; y that is the movement along the y axis in meters where a positive number mover the object upwards and a negative number downwards; z that is the movement along the z axis where a positive number is away from the viewer and a negative number is towards the viewer.

The Translate statement affects all vertices defined so far in the CreateMeshBuilder section where it is used. More than one Translate statement can be used in a single CreateMeshBuilder section.

Example 1:

```
Translate, -3, -1, 5.5,
```

This describes a movement of all vertices defined so far in the CreateMeshBuilder section 3 meters to the left, 1 meter downwards and 5.5 meters away from the viewer.

TranslateAll

```
TranslateAll, x,y,z,
```

Variable	Type	Notice
x	Decimal number	Translation along the x axis in meters.
y	Decimal number	Translation along the y axis in meters.
z	Decimal number	Translation along the z axis in meters.

The TranslateAll statement do the same thing as the Translate statement, **but** affects all CreateMeshBuilder sections written before the TranslateAll statement. The 3 parameters are also the same as for the Translate statement.

Example 1:

```
TranslateAll, -3, -1, 5.5,
```

This describes a movement of all vertices in all CreateMeshBuilder sections before the TranslateAll statement 3 meters to the left, 1 meter downwards and 5.5 meters away from the viewer.

;

```
; Comment text
```

Comment text in an csv object file begins with **;**. The comment can be on a line of its own, or after some other statement on a line.

Example 1:

```
; This is a comment
```

This is a comment on a line of its own

Example2:

```
AddFace, 0, 1, 3, 2,  ; Face 0
AddFace, 1, 5, 7, 3,  ; Face 1
AddFace, 5, 4, 6, 7,  ; Face 2
AddFace  4, 0, 2, 6,  ; Face 3
AddFace, 4, 5, 1, 0,  ; Face 4
AddFace, 2, 3, 7, 6,  ; Face 5
```

```
These are comments on lines with statements.
```