The UNOFFICIAL x Object Handbook for OpenBVE

Patrick Norqvist

The UNOFFICIAL x Object Handbook for OpenBVE

Patrick Norqvist

The Unofficial x Object Handbook for OpenBVE

Author: Patrick Norqvist
ISBN: 978-91-982993-1-1
Website: www.openbve.net

First edition

Preamble

This handbook is a first try to explain the x object file format for OpenBVE. It was written because of the lack of such information published on the Internet.

The material is mainly derived from dissection of x object files available in a few routes, information available on the Internet on the Direct-X Retained mode, and files made with the csv object file to x object file converter program that was made by the BVE author Mr. Takashi Kojima a number of years ago.

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 given to:

- Mr. Paul Bourke for his paper "Direct-X File Format", which has been a valuable source of information.
- Mr. Anthony Bowden, with his website `http://www.railsimroutes.net/` , for his x 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 4.

Table of Contents

1 Building a simple object 1

2 Using textures 15

3 Using colors 31

4 Transparency 41

5 Calculating the mesh normals 45

6 Statements in alphabetical order 51

- Header 52
- Material 53
- Mesh 55
- MeshMaterialList 58
- MeshNormals 61
- MeshTextureCoords 64
- TextureFilname 66
- xof 68
- / / 69

Chapter 1

Building a simple object

The x file format allows us to build objects by writing a number of statements in a text file. Most of the statements used in the x file format, has its equivalents in the csv and b3d object file formats. One difference compared with these other file formats is the requirement for a header. The file extension should be ".x".

The x file format is a subset of the Direct-X Retained Mode file format.

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

The first we write in the text file are two headers:

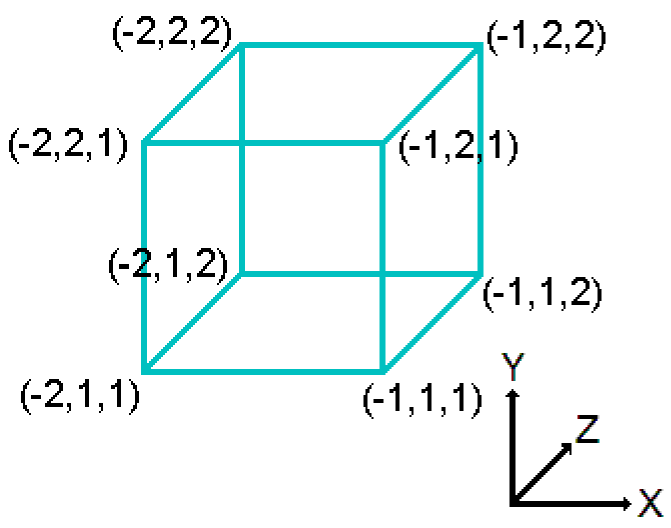
```
xof 0303txt 0032

Header
  {
    1;   // Major version
    0;   // Minor version
    1;   // Text file
  }
```

The first line contains what is known as the "Magic number" xof, followed by major version number 03 and minor version number 03 and txt to define the the rest of the file is a text file. The number 0032 defines the number of bits for float numbers (that is numbers with decimals and/or exponents).

The Header statement with parameters below also defines major and minor version number and file type.

We also make comments in the code. A comment starts with 2 slashes / /.

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



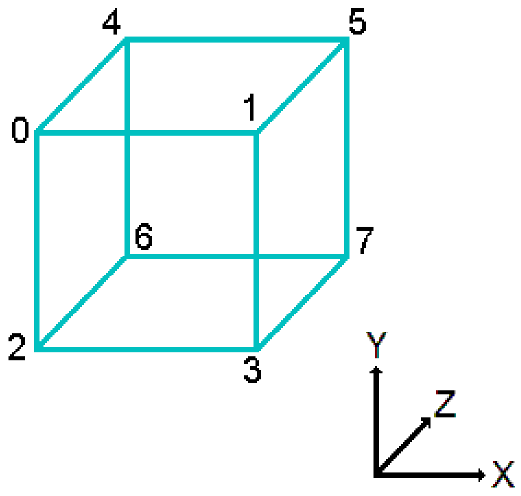
Each vertex has the coordinates (x,y,z) to define its location in the 3D world. The cube has a side of 1 meter, and is located with its right bottom front corner (-1,1,1) 1 meter to the left on the x-axis (which gives a negative x coordinate), one meter up on the y axis and 1 meter forward on the z axis.

We add the cube's 8 vertices to the x file using the Mesh statement:

```
xof 0303txt 0032

Header
  {
    1;  // Major version
    0;  // Minor version
    1;  // Text file
  }

Mesh
  {
    8; // 8 vertices
    -2.000000;2.000000;1.000000;, // Vertex 0
    -1.000000;2.000000;1.000000;, // Vertex 1
    -2.000000;1.000000;1.000000;, // Vertex 2
    -1.000000;1.000000;1.000000;, // Vertex 3
    -2.000000;2.000000;2.000000;, // Vertex 4
    -1.000000,2.000000;2.000000;, // Vertex 5
    -2.000000;1.000000;2.000000;, // Vertex 6
    -1.000000;1.000000;2.000000;; // Vertex 7
```

The first parameter states that the coordinates of 8 vertices will follow. Then follow the coordinates x; y; z; for each of the 8 vertices of the cube. The 8 vertices has numbers fro 0 to 7

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 to the mesh statement that we have already begun with. We define a face by first stating how many vertices that make the face, then listing the vertex numbers, in clockwise order seen from the outside of the object, of the face as parameters for each face:

```
xof 0303txt 0032

Header
  {
    1;   // Major version
    0;   // Minor version
    1;   // Text file
  }
```

```
Mesh
  {
    8; // 8 vertices
    -2.000000;2.000000;1.000000;, // Vertex 0
    -1.000000;2.000000;1.000000;, // Vertex 1
    -2.000000;1.000000;1.000000;, // Vertex 2
    -1.000000;1.000000;1.000000;, // Vertex 3
    -2.000000;2.000000;2.000000;, // Vertex 4
    -1.000000,2.000000;2.000000;, // Vertex 5
    -2.000000;1.000000;2.000000;, // Vertex 6
    -1.000000;1.000000;2.000000;; // Vertex 7
    6 // 6 faces
    4;0,1,3,2;, // 4 vertices - face 0
    4;1,5,7,3;, // 4 vertices - face 1
    4;5,4,6,7;, // 4 vertices - face 2
    4;4;0;2;6;, // 4 vertices - face 3
    4;4,5,1,0;, // 4 vertices - face 4
    4;2,3,7,6;; // 4 vertices - 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 first define materials for the faces, the apply them to each face. Here we want a green color on all 6 faces.

To define a material, we start with the statement MeshMaterialList. It takes 3 parameters: Number of materials, number of faces, and a list of face numbers:

```
xof 0303txt 0032

Header
  {
    1;   // Major version
    0;   // Minor version
    1;   // Text file
  }
```

```
Mesh
  {
    8; // 8 vertices
    -2.000000;2.000000;1.000000;, // Vertex 0
    -1.000000;2.000000;1.000000;, // Vertex 1
    -2.000000;1.000000;1.000000;, // Vertex 2
    -1.000000;1.000000;1.000000;, // Vertex 3
    -2.000000;2.000000;2.000000;, // Vertex 4
    -1.000000,2.000000;2.000000;, // Vertex 5
    -2.000000;1.000000;2.000000;, // Vertex 6
    -1.000000;1.000000;2.000000;; // Vertex 7
    6 // 6 faces
    4;0,1,3,2;, // 4 vertices - face 0
    4;1,5,7,3;, // 4 vertices - face 1
    4;5,4,6,7;, // 4 vertices - face 2
    4;4;0;2;6;, // 4 vertices - face 3
    4;4,5,1,0;, // 4 vertices - face 4
    4;2,3,7,6;; // 4 vertices - face 5

    MeshMaterialList
      {
        1; // We will define 1 material
        1; // Apply to 1 (all) faces
        0; // Material 0 to face 0 (all faces)
```

There is a special case when only one material is defined and should be applied to all faces. We do not need to specify the true number of faces and a complete face/material number list.

Now we use the Material statement, within the MeshMaterialList statement, to define material of 1 color: Green.

The Material 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 1 (a value of 0 will mean full transparency). The value for each of these 4 parameters can range from 0 to 1.

The specular color is the color that comes when the lights reflects in (a) certain point(s) of a shiny material. Its parameters are the power and the color. Usually those reflections appear white. The power means how intense those reflections are, and here we have to experiment with the value to find the result we want for an object.

```
xof 0303txt 0032

Header
  {
    1;  // Major version
    0;  // Minor version
    1;  // Text file
  }

Mesh
  {
    8; // 8 vertices
    -2.000000;2.000000;1.000000;, // Vertex 0
    -1.000000;2.000000;1.000000;, // Vertex 1
    -2.000000;1.000000;1.000000;, // Vertex 2
    -1.000000;1.000000;1.000000;, // Vertex 3
    -2.000000;2.000000;2.000000;, // Vertex 4
    -1.000000,2.000000;2.000000;, // Vertex 5
    -2.000000;1.000000;2.000000;, // Vertex 6
    -1.000000;1.000000;2.000000;; // Vertex 7
    6 // 6 faces
    4;0,1,3,2;, // 4 vertices - face 0
    4;1,5,7,3;, // 4 vertices - face 1
    4;5,4,6,7;, // 4 vertices - face 2
    4;4;0;2;6;, // 4 vertices - face 3
    4;4,5,1,0;, // 4 vertices - face 4
    4;2,3,7,6;; // 4 vertices - face 5

    MeshMaterialList
      {
        1; // We will define 1 material
        1; // Apply to 1 (all) faces
        0; // Material 0 to face 0 (all faces)
        Material
          {
            // Red Green Blue and Alpha channel
            0.000000;1.000000;0.000000;1.000000;;
            //Specular power
            0.000000;
            // Specular color Red Green Blue
            0.000000;0.000000;0.000000;
            // Emissive color Red Green Blue
            0.000000;0.000000;0.000000;
          }
      }
```

The next step is to add the normal vectors for each face's vertices. This is done using the statement MeshNormals. That calculation of the normal vectors is rather cumbersome, and we leave those values to 0 and let OpenBVE automatically calculate the values when loading the object.

If one wants to define the normals manually, the MeshNormals statement takes a number of parameters. The first is the number of normals, the the coordinates for each normal, then the number of faces in the object and finally for each face the number of vertices, and the number of the normal to use for each of the vertices of the face.

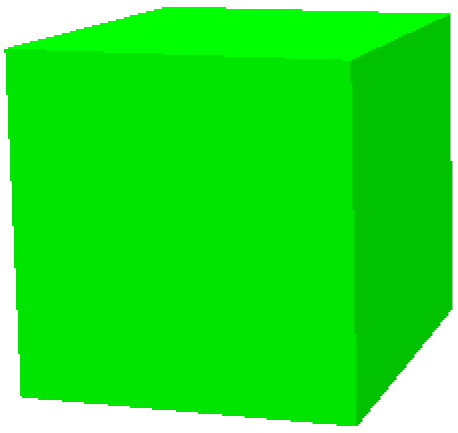
```
xof 0303txt 0032

Header
  {
    1;  // Major version
    0;  // Minor version
    1;  // Text file
  }

Mesh
  {
    8; // 8 vertices
    -2.000000;2.000000;1.000000;, // Vertex 0
    -1.000000;2.000000;1.000000;, // Vertex 1
    -2.000000;1.000000;1.000000;, // Vertex 2
    -1.000000;1.000000;1.000000;, // Vertex 3
    -2.000000;2.000000;2.000000;, // Vertex 4
    -1.000000,2.000000;2.000000;, // Vertex 5
    -2.000000;1.000000;2.000000;, // Vertex 6
    -1.000000;1.000000;2.000000;; // Vertex 7
    6 // 6 faces
    4;0,1,3,2;, // 4 vertices - face 0
    4;1,5,7,3;, // 4 vertices - face 1
    4;5,4,6,7;, // 4 vertices - face 2
    4;4;0;2;6;, // 4 vertices - face 3
    4;4,5,1,0;, // 4 vertices - face 4
    4;2,3,7,6;; // 4 vertices - face 5

    MeshMaterialList
      {
        1; // We will define 1 material
        1; // Apply to 1 (all) faces
        0; // Material 0 to face 0 (all faces)

        Material
          {
            // Red Green Blue and Alpha channel
            0.000000;1.000000;0.000000;1.000000;;
            //Specular power
            0.000000;
            // Specular color Red Green Blue
            0.000000;0.000000;0.000000;;
            // Emissive color Red Green Blue
            0.000000;0.000000;0.000000;;
          }
      }
    MeshNormals
      {
        6; // 6 normals
        0.000000;0.000000;0.000000;, // Normal 0
        0.000000;0.000000;0.000000;, // Normal 1
        0.000000;0.000000;0.000000;, // Normal 2
        0.000000;0.000000;0.000000;, // Normal 3
        0.000000;0.000000;0.000000;, // Normal 4
        0.000000;0.000000;0.000000;; // Normal 5
        6; // 6 faces
        4;0;0;0;0;, // Face 0, 4 vertices, normal 0
        4;1;1;1;1;, // Face 1, 4 vertices, normal 1
        4;2;2;2;2;, // Face 2, 4 vertices, normal 2
        4;3;3;3;3;, // Face 3, 4 vertices, normal 3
        4;4;4;4;4;, // Face 4, 4 vertices, normal 4
        4;5;5;5;5;, // Face 5, 4 vertices, normal 5
      }
```

Finally, we must use the statement MeshTextureCoords. It defines how a texture, if specified in connection with a material, should be applied to one or more faces.

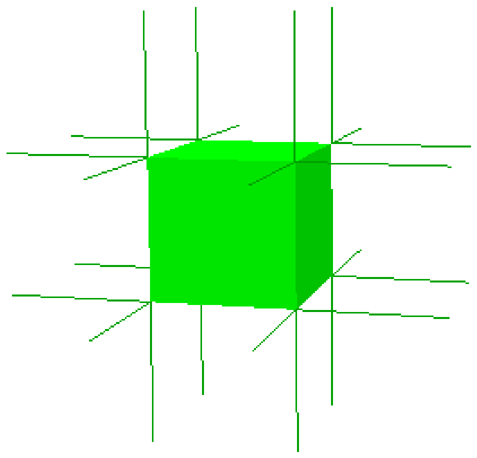
In this case, we have not defined a texture, so we leave all coordinates at 0:

```
xof 0303txt 0032

Header
  {
    1;  // Major version
    0;  // Minor version
    1;  // Text file
  }

Mesh
  {
    8; // 8 vertices
    -2.000000;2.000000;1.000000;, // Vertex 0
    -1.000000;2.000000;1.000000;, // Vertex 1
    -2.000000;1.000000;1.000000;, // Vertex 2
    -1.000000;1.000000;1.000000;, // Vertex 3
    -2.000000;2.000000;2.000000;, // Vertex 4
    -1.000000,2.000000;2.000000;, // Vertex 5
    -2.000000;1.000000;2.000000;, // Vertex 6
    -1.000000;1.000000;2.000000;; // Vertex 7
    6 // 6 faces
    4;0,1,3,2;, // 4 vertices - face 0
    4;1,5,7,3;, // 4 vertices - face 1
    4;5,4,6,7;, // 4 vertices - face 2
    4;4;0;2;6;, // 4 vertices - face 3
    4;4,5,1,0;, // 4 vertices - face 4
    4;2,3,7,6;; // 4 vertices - face 5
    MeshMaterialList
      {
        1; // We will define 1 material
        1; // Apply to 1 (all) faces
        0; // Material 0 to face 0 (all faces)

        Material // Material 0
          {
            // Red Green Blue and Alpha channel
            0.000000;1.000000;0.000000;1.000000;;
            //Specular power
            0.000000;
            // Specular color Red Green Blue
            0.000000;0.000000;0.000000;;
            // Emissive color Red Green Blue
            0.000000;0.000000;0.000000;;
          }
      }
    MeshNormals
      {
        6; // 6 normals
        0.000000;0.000000;0.000000;, // Normal 0
        0.000000;0.000000;0.000000;, // Normal 1
        0.000000;0.000000;0.000000;, // Normal 2
        0.000000;0.000000;0.000000;, // Normal 3
        0.000000;0.000000;0.000000;, // Normal 4
        0.000000;0.000000;0.000000;; // Normal 5
        6; // 6 faces
        4;0;0;0;0;, // Face 0, 4 vertices, normal 0
        4;1;1;1;1;, // Face 1, 4 vertices, normal 1
        4;2;2;2;2;, // Face 2, 4 vertices, normal 2
        4;3;3;3;3;, // Face 3, 4 vertices, normal 3
        4;4;4;4;4;, // Face 4, 4 vertices, normal 4
        4;5;5;5;5;, // Face 5, 4 vertices, normal 5
      }
    MeshTextureCoords
      {
        8;  // 8 vertices
        0.000000;0.000000;,  // Coord @ vertex 0
        0.000000;0.000000;,  // Coord @ vertex 1
        0.000000;0.000000;,  // Coord @ vertex 2
        0.000000;0.000000;,  // Coord @ vertex 3
        0.000000;0.000000;,  // Coord @ vertex 4
        0.000000;0.000000;,  // Coord @ vertex 5
        0.000000;0.000000;,  // Coord @ vertex 6
        0.000000;0.000000;;  // Coord @ vertex 7
      }
  }
```

Now we have the complete text for the x object file of the green cube. If we put this text in a text file with the name such as “Cube.x” 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:



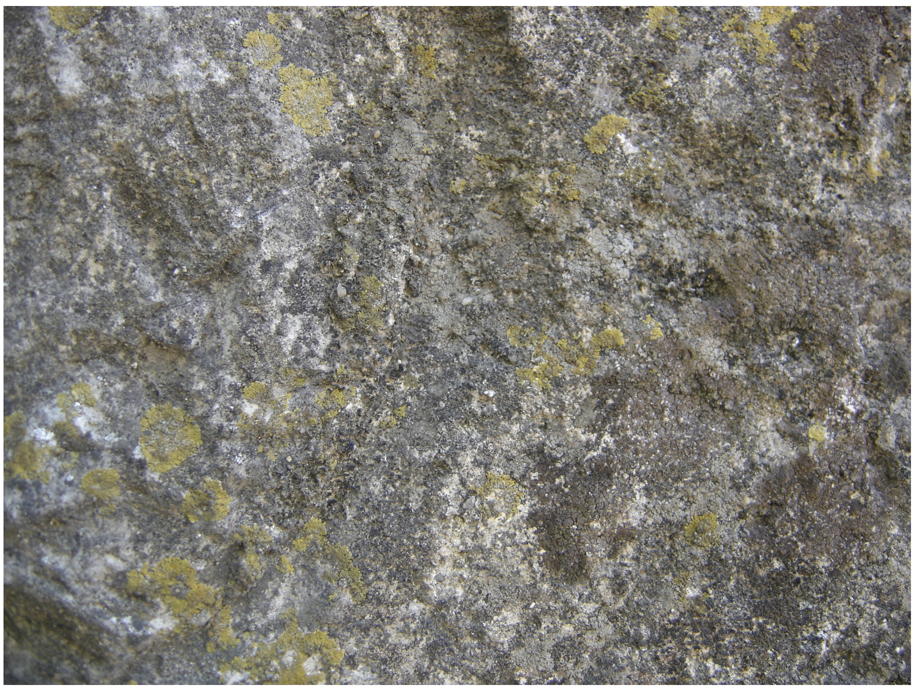
BLANK PAGE

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 Windows bitmap (.bmp) file with the name “RockTexture.bmp”.

In the x object file for the cube, we will add the TextureFilename statement within the Material statement:

```
            TextureFilename
              {
                "RockTexture.bmp";
              }
```

We will also change the basic material color to white, and add texture coordinates for all 8 vertices.

The whole x object file will now be this:

```
xof 0303txt 0032

Header
  {
    1;  // Major version
    0;  // Minor version
    1;  // Text file
  }

Mesh
  {
    8;  // 8 vertices
    -2.000000;2.000000;1.000000;,  // Vertex 0
    -1.000000;2.000000;1.000000;,  // Vertex 1
    -2.000000;1.000000;1.000000;,  // Vertex 2
    -1.000000;1.000000;1.000000;,  // Vertex 3
    -2.000000;2.000000;2.000000;,  // Vertex 4
    -1.000000;2.000000;2.000000;,  // Vertex 5
    -2.000000;1.000000;2.000000;,  // Vertex 6
    -1.000000;1.000000;2.000000;;  // Vertex 7
    6;  // 6 faces
    4;0,1,3,2;,  // 4 vertices, Face 0
    4;1,5,7,3;,  // 4 vertices, Face 1
    4;5,4,6,7;,  // 4 vertices, Face 2
    4;4,0,2,6;,  // 4 vertices, Face 3
    4;4,5,1,0;,  // 4 vertices, Face 4
    4;2,3,7,6;;  // 4 vertices, Face 5

    MeshMaterialList
      {
        1;  // We will define 1 material
        1;  // and apply to 1 (all) surfaces
        0;  // Material 0 to face 0 (all faces)

        Material
          {
            // Red Green Blue and Alpha channel
            1.000000;1.000000;1.000000;1.000000;;
            // Specular power
            0.000000;
            // Specular color Red Green Blue
            0.000000;0.000000;0.000000;;
            // Emissive color Red Green Blue
            0.000000;0.000000;0.000000;;

            TextureFilename
              {
                "RockTexture.bmp";
              }
          }
      }

    MeshNormals
      {
        6;  // 6 normals
        0.000000;0.000000;0.000000;,  // Normal 0
        0.000000;0.000000;0.000000;,  // Normal 1
        0.000000;0.000000;0.000000;,  // Normal 2
        0.000000;0.000000;0.000000;,  // Normal 3
        0.000000;0.000000;0.000000;,  // Normal 4
        0.000000;0.000000;0.000000;;  // Normal 5
        6;  // 6 faces
        4;0,0,0,0;,  //Face0, 4 vertices, normal 0
        4;1,1,1,1;,  //Face1, 4 vertices, normal 1
        4;2,2,2,2;,  //Face2, 4 vertices, normal 2
        4;3,3,3,3;,  //Face3, 4 vertices, normal 3
        4;4,4,4,4;,  //Face4, 4 vertices, normal 4
        4;5,5,5,5;;  //Face5, 4 vertices, normal 5
      }

    MeshTextureCoords
      {
        8;  // 8 vertices
        0.000000;0.500000;,  // Texture coord @ vertex 0
        0.500000;0.500000;,  // Texture coord @ vertex 1
        0.000000;1.000000;,  // Texture coord @ vertex 2
        0.500000;1.000000;,  // Texture coord @ vertex 3
        0.000000;0.000000;,  // Texture coord @ vertex 4
        0.500000;0.000000;,  // Texture coord @ vertex 5
        1.000000;0.000000;,  // Texture coord @ vertex 6
        1.000000;1.000000;;  // Texture coord @ vertex 7
      }
  }
```

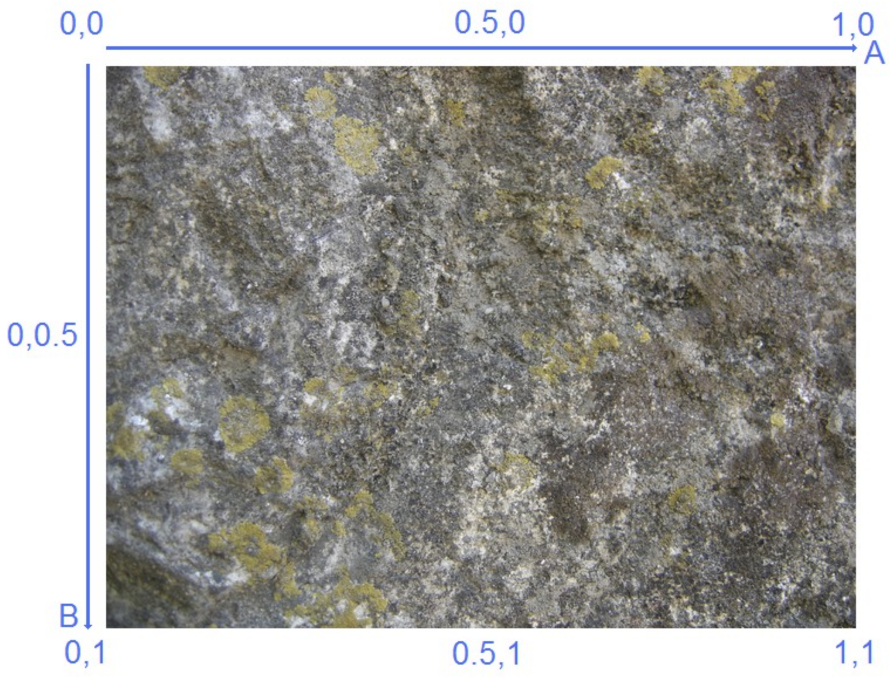
There is a number of ways to arrange the texture coordinates. In this case, they are given in such a way that the 3 visible sides of of the cube are textured. The bottom and far side will however look silly. Normally, that does not matter as the object is viewed from almost only one direction, that is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

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

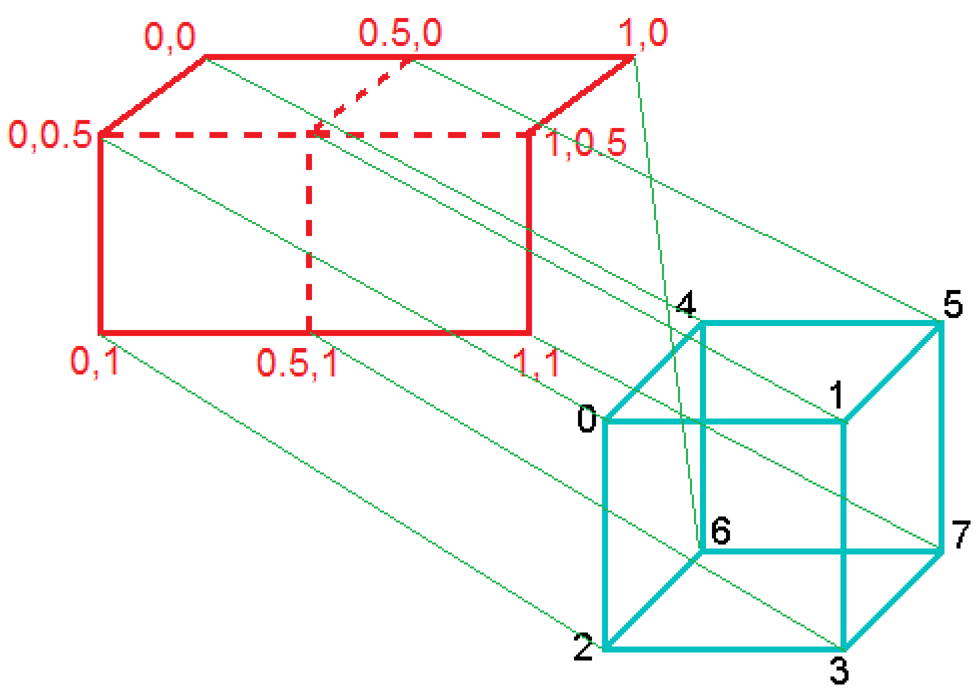
```
    MeshTextureCoords
      {
        8;  // 8 vertices
        0.000000;0.500000;,  // Texture coord @ vertex 0
        0.500000;0.500000;,  // Texture coord @ vertex 1
        0.000000;1.000000;,  // Texture coord @ vertex 2
        0.500000;1.000000;,  // Texture coord @ vertex 3
        0.000000;0.000000;,  // Texture coord @ vertex 4
        0.500000;0.000000;,  // Texture coord @ vertex 5
        1.000000;0.000000;,  // Texture coord @ vertex 6
        1.000000;1.000000;;  // Texture coord @ vertex 7
      }
```

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.

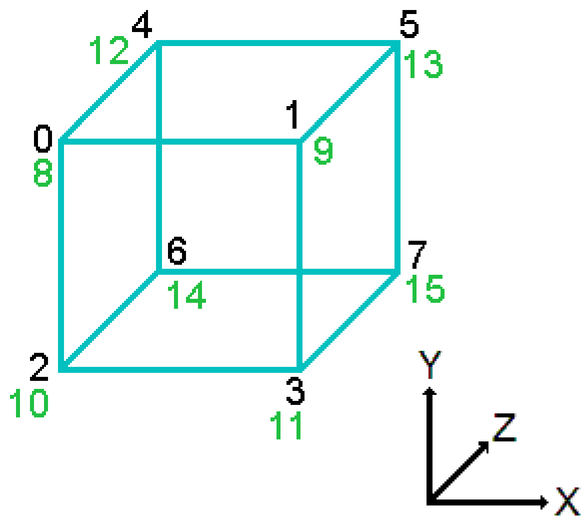


If these is a regular texture, such as a wooden box, we must resort to another method to get a reasonable result.

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.

```
xof 0303txt 0032

Header
  {
    1;   // Major version
    0;   // Minor version
    1;   // Text file
  }

Mesh
  {
    16; // 16 vertices
    -2.000000;2.000000;1.000000;,   // Vertex  0
    -1.000000;2.000000;1.000000;,   // Vertex  1
    -2.000000;1.000000;1.000000;,   // Vertex  2
    -1.000000;1.000000;1.000000;,   // Vertex  3
    -2.000000;2.000000;2.000000;,   // Vertex  4
    -1.000000;2.000000;2.000000;,   // Vertex  5
    -2.000000;1.000000;2.000000;,   // Vertex  6
    -1.000000;1.000000;2.000000;,   // Vertex  7
    -2.000000;2.000000;1.000000;,   // Vertex  8
    -1.000000;2.000000;1.000000;,   // Vertex  9
    -2.000000;1.000000;1.000000;,   // Vertex 10
    -1.000000;1.000000;1.000000;,   // Vertex 11
    -2.000000;2.000000;2.000000;,   // Vertex 12
    -1.000000;2.000000;2.000000;,   // Vertex 13
    -2.000000;1.000000;2.000000;,   // Vertex 14
    -1.000000;1.000000;2.000000;;   // Vertex 15
    6;  // 6 faces
    4;0,1,3,2;,      // 4 vertices, Face 0
    4;1,5,7,3;,      // 4 vertices, Face 1
    4;5,4,6,7;,      // 4 vertices, Face 2
    4;4,0,2,6;,      // 4 vertices, Face 3
    4;12,13,9,8;,    // 4 vertices, Face 4
    4;10,11,15,14;;  // 4 vertices, Face 5
```

For the first 4 faces of the cube, that is the front, back, left and right faces, we use see 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 now do specify 1 material, which is white with the texture “RockTexture.bmp” applied to it:

```
    MeshMaterialList
      {
        1;   // Number of materials
        6;   // Number of faces
        0,   // Material number for face 0
        0,   // Material number for face 1
        0,   // Material number for face 2
        0,   // Material number for face 3
        0,   // Material number for face 4
        0;   // Material number for face 5

        Material  // Material number 0
          {
            // Red Green Blue and Alpha channel
            1.000000;1.000000;1.000000;1.000000;;
            // Specular power
            0.000000;
            // Specular color Red Green Blue
            0.000000;0.000000;0.000000;;
            // Emissive color Red Green Blue
            0.000000;0.000000;0.000000;;
            TextureFilename
              {
                "RockTexture.bmp";
              }
          }
      }
```

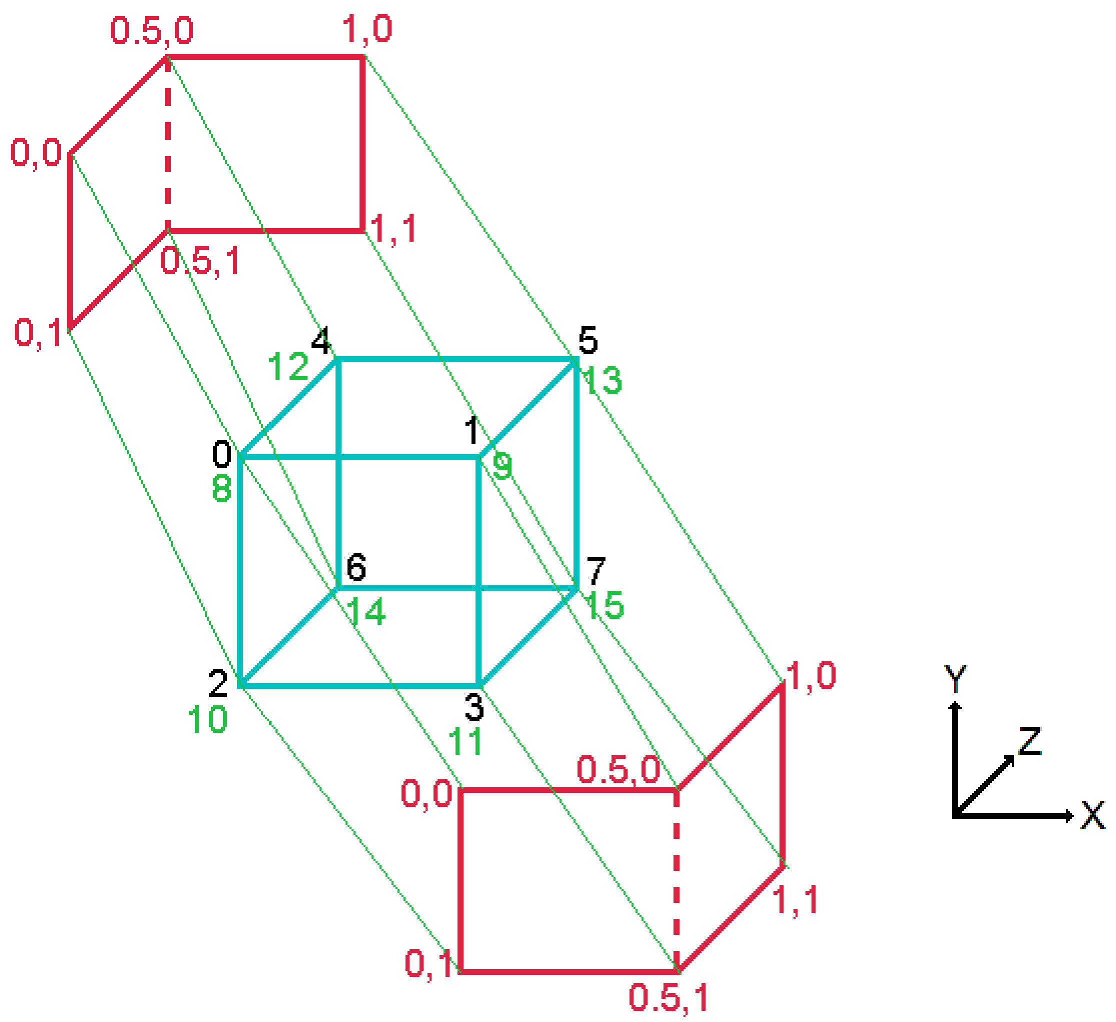
As before, we set the normal vectors in the MeshNormals statement to 0 to get them automatically calculated for all faces:

```
   MeshNormals
      {
        6;   // 6 normals
        0.000000;0.000000;0.000000;,  // Normal to face 0
        0.000000;0.000000;0.000000;,  // Normal to face 1
        0.000000;0.000000;0.000000;,  // Normal to face 2
        0.000000;0.000000;0.000000;,  // Normal to face 3
        0.000000;0.000000;0.000000;,  // Normal to face 4
        0.000000;0.000000;0.000000;;  // Normal to face 5
        6;  // 6 faces
        4;0,0,0,0;,  // 4 vertices, normal #0 for face 0
        4;1,1,1,1;,  // 4 vertices, normal #1 for face 1
        4;2,2,2,2;,  // 4 vertices, normal #2 for face 2
        4;3,3,3,3;,  // 4 vertices, normal #3 for face 3
        4;4,4,4,4;,  // 4 vertices, normal #4 for face 4
        4;5,5,5,5;;  // 4 vertices, normal #5 for face 5
      }
```

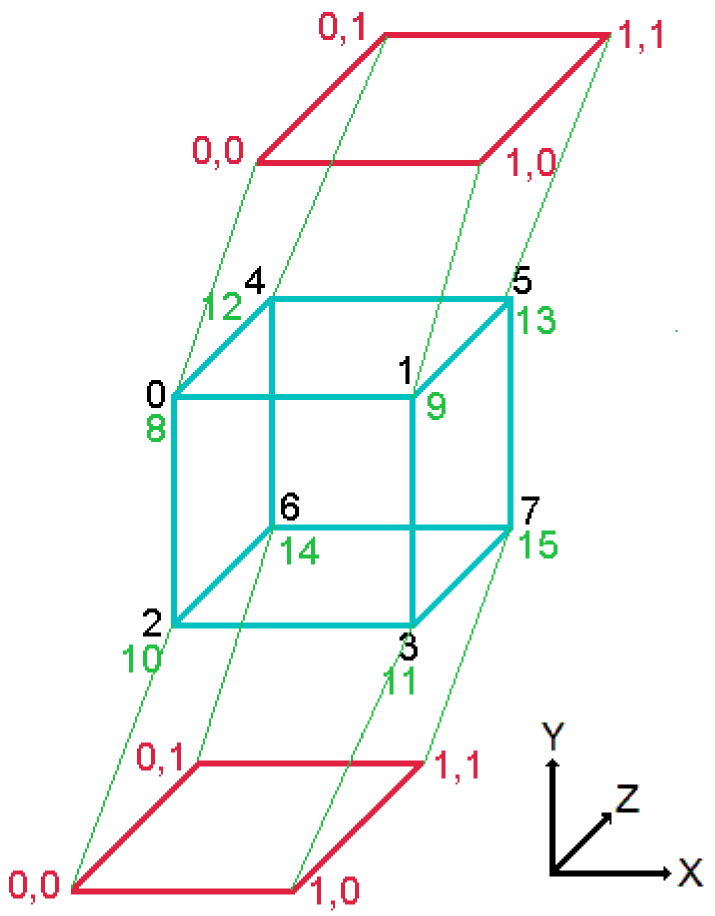
Then we come to the texture coordinates, which are to be set for all 16 vertices.

```
   MeshTextureCoords
      {
        16;
        0.000000;0.000000;,  // Vertex  0
        0.500000;0.000000;,  // Vertex  1
        0.000000;1.000000;,  // Vertex  2
        0.500000;1.000000;,  // Vertex  3
        0.500000;0.000000;,  // Vertex  4
        1.000000;0.000000;,  // Vertex  5
        0.500000;1.000000;,  // Vertex  6
        1.000000;1.000000;,  // Vertex  7
        0.000000;0.000000;,  // Vertex  8
        1.000000;0.000000;,  // Vertex  9
        0.000000;0.000000;,  // Vertex 10
        1.000000;0.000000;,  // Vertex 11
        0.000000;1.000000;,  // Vertex 12
        1.000000;1.000000;,  // Vertex 13
        0.000000;1.000000;,  // Vertex 14
        1.000000;1.000000;;  // Vertex 15
      }
```

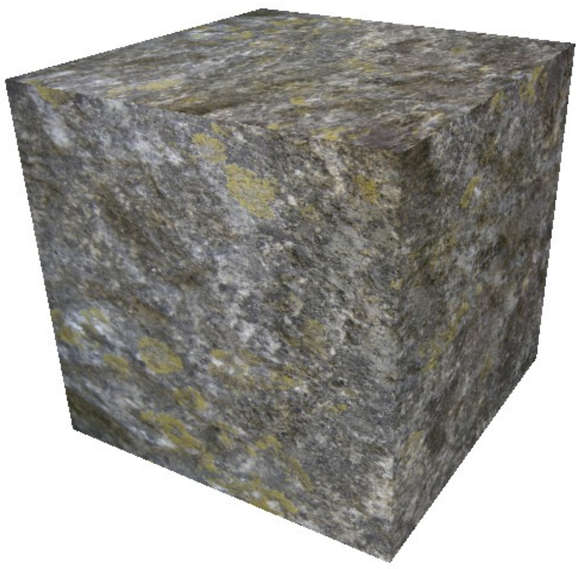
We wrap the texture around the cube's vertical sides (front, left back and right sides, attaching texture coordinates to the 8 vertices 0 to 7. The texture is attached to the front and right side, then repeated at the back and right sides:


0.5,0
1,0
0,0
1,1
0,5,1
0,1
4
12
5
13
1
0
9
8
6
7
14
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 x object file for this cube is:

```
xof 0303txt 0032

Header
  {
    1;  // Major version
    0;  // Minor version
    1;  // Text file
  }

Mesh
  {
    16;  // 16 vertices
    -2.000000;2.000000;1.000000;,  // Vertex  0
    -1.000000;2.000000;1.000000;,  // Vertex  1
    -2.000000;1.000000;1.000000;,  // Vertex  2
    -1.000000;1.000000;1.000000;,  // Vertex  3
    -2.000000;2.000000;2.000000;,  // Vertex  4
    -1.000000;2.000000;2.000000;,  // Vertex  5
    -2.000000;1.000000;2.000000;,  // Vertex  6
    -1.000000;1.000000;2.000000;,  // Vertex  7
    -2.000000;2.000000;1.000000;,  // Vertex  8
    -1.000000;2.000000;1.000000;,  // Vertex  9
    -2.000000;1.000000;1.000000;,  // Vertex 10
    -1.000000;1.000000;1.000000;,  // Vertex 11
    -2.000000;2.000000;2.000000;,  // Vertex 12
    -1.000000;2.000000;2.000000;,  // Vertex 13
    -2.000000;1.000000;2.000000;,  // Vertex 14
    -1.000000;1.000000;2.000000;;  // Vertex 15
    6;  // 6 faces
    4;0,1,3,2;,       // 4 vertices, Face 0
    4;1,5,7,3;,       // 4 vertices, Face 1
    4;5,4,6,7;,       // 4 vertices, Face 2
    4;4,0,2,6;,       // 4 vertices, Face 3
    4;12,13,9,8;,     // 4 vertices, Face 4
    4;10,11,15,14;;   // 4 vertices, Face 5

    MeshMaterialList
      {
        1;  // Number of materials
        6;  // Number of faces
        0,  // Material number for face 0
        0,  // Material number for face 1
        0,  // Material number for face 2
        0,  // Material number for face 3
        0,  // Material number for face 4
        0;  // Material number for face 5

        Material  // Material number 0
          {
            // Red Green Blue and Alpha channel
            1.000000;1.000000;1.000000;1.000000;;
            // Specular power
            0.000000;
            // Specular color Red Green Blue
            0.000000;0.000000;0.000000;;
            // Emissive color Red Green Blue
            0.000000;0.000000;0.000000;;

            TextureFilename
              {
                "RockTexture.bmp";
              }
          }
      }

   MeshNormals
      {
        6;  // 6 normals
        0.000000;0.000000;0.000000;,  // Normal to face 0
        0.000000;0.000000;0.000000;,  // Normal to face 1
        0.000000;0.000000;0.000000;,  // Normal to face 2
        0.000000;0.000000;0.000000;,  // Normal to face 3
        0.000000;0.000000;0.000000;,  // Normal to face 4
        0.000000;0.000000;0.000000;;  // Normal to face 5
        6;  // 6 faces
        4;0,0,0,0;,  // 4 vertices, normal #0 for face 0
        4;1,1,1,1;,  // 4 vertices, normal #1 for face 1
        4;2,2,2,2;,  // 4 vertices, normal #2 for face 2
        4;3,3,3,3;,  // 4 vertices, normal #3 for face 3
        4;4,4,4,4;,  // 4 vertices, normal #4 for face 4
        4;5,5,5,5;;  // 4 vertices, normal #5 for face 5
      }
```

```
  MeshTextureCoords
     {
       16;
       0.000000;0.000000;,  // Vertex  0
       0.500000;0.000000;,  // Vertex  1
       0.000000;1.000000;,  // Vertex  2
       0.500000;1.000000;,  // Vertex  3
       0.500000;0.000000;,  // Vertex  4
       1.000000;0.000000;,  // Vertex  5
       0.500000;1.000000;,  // Vertex  6
       1.000000;1.000000;,  // Vertex  7
       0.000000;0.000000;,  // Vertex  8
       1.000000;0.000000;,  // Vertex  9
       0.000000;0.000000;,  // Vertex 10
       1.000000;0.000000;,  // Vertex 11
       0.000000;1.000000;,  // Vertex 12
       1.000000;1.000000;,  // Vertex 13
       0.000000;1.000000;,  // Vertex 14
       1.000000;1.000000;;  // Vertex 15
     }
 }
```

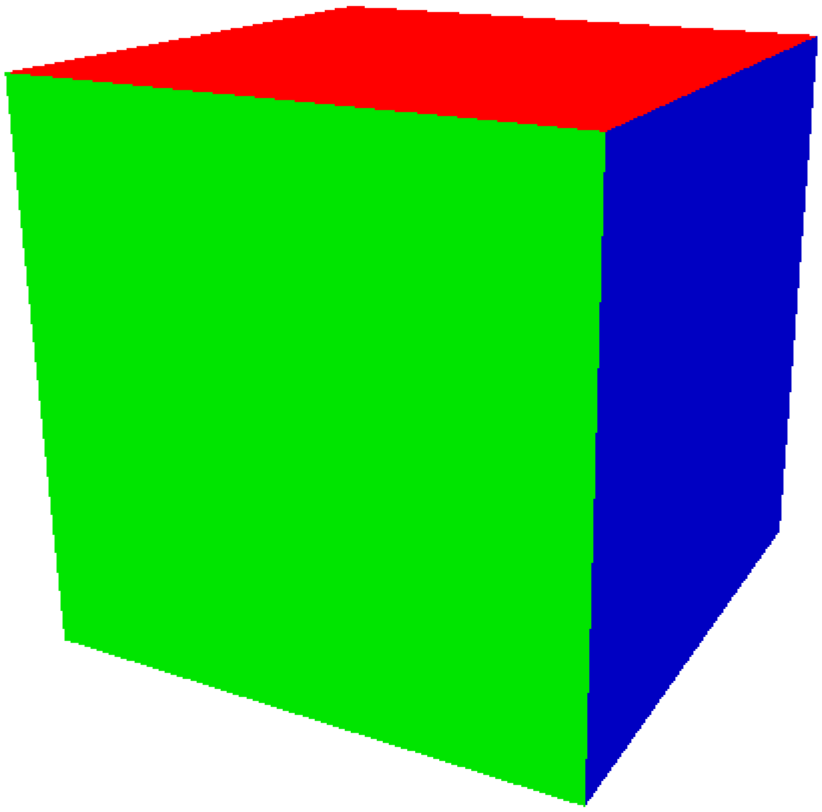
BLANK PAGE

Chapter 3

Using colors

The first cube we made in chapter 1 had all sides green. We can use different colors on the faces of the cube, if we specify more materials. One color means one material.

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 the MeshMaterialList statement do specify which material/color to apply on each face, then we specify each material, one green, one red and one blue.

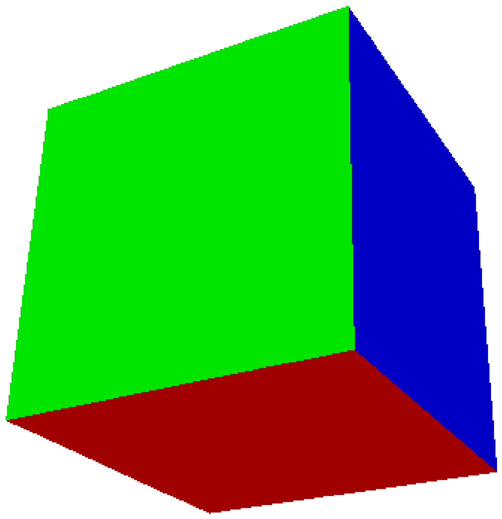
```
    MeshMaterialList
      {
        3;   // 3 materials
        6;   // 6 faces
        0,   // Material 0 on face 0
        2,   // Material 2 on face 1
        0,   // Material 0 on face 2
        2,   // Material 2 on face 3
        1,   // Material 1 on face 4
        1;   // Material 1 on face 5
        Material  // Material 0 (Green)
          {
            // Red Green Blue Alpha channel
            0.000000;1.000000;0.000000;1.000000;;
            // Specular power
            0.000000;
            // Specular color Red Green Blue
            0.000000;0.000000;0.000000;;
            // Emissive color Red Green Blue
            0.000000;0.000000;0.000000;;
          }

        Material  // Material 1 (Red)
          {
            // Red Green Blue Alpha channel
            1.000000;0.000000;0.000000;1.000000;;
            // Specular power
            0.000000;
            // Specular color Red Green Blue
            0.000000;0.000000;0.000000;;
            // Emissive color Red Green Blue
            0.000000;0.000000;0.000000;;
          }

        Material  // Material 2 (Blue)
          {
            // Red Green Blue Alpha channel
            0.000000;0.000000;1.000000;1.000000;;
            // Specular power
            0.000000;
            // Specular color Red Green Blue
            0.000000;0.000000;0.000000;;
            // Emissive color Red Green Blue
            0.000000;0.000000;0.000000;;
          }
      }
```

As we use no texture, we leave all texture coordinates at 0.

The x object file for this cube is:

```
xof 0303txt 0032

Header
  {
    1;  // Major version
    0;  // Minor version
    1;  // Text file
  }

Mesh
  {
     8;  // 8 vertices
    -2.000000;2.000000;1.000000;,  // Vertex 0
    -1.000000;2.000000;1.000000;,  // Vertex 1
    -2.000000;1.000000;1.000000;,  // Vertex 2
    -1.000000;1.000000;1.000000;,  // Vertex 3
    -2.000000;2.000000;2.000000;,  // Vertex 4
    -1.000000;2.000000;2.000000;,  // Vertex 5
    -2.000000;1.000000;2.000000;,  // Vertex 6
    -1.000000;1.000000;2.000000;;  // Vertex 7
    6;  // 6 faces
    4;0,1,3,2;,  // 4 vertices, face 0
    4;1,5,7,3;,  // 4 vertices, face 1
    4;5,4,6,7;,  // 4 vertices, face 2
    4;4,0,2,6;,  // 4 vertices, face 3
    4;4,5,1,0;,  // 4 vertices, face 4
    4;2,3,7,6;;  // 4 vertices, face 5

    MeshMaterialList
      {
        3;  // 3 materials
        6;  // 6 faces
        0,  // Material 0 on face 0
        2,  // Material 2 on face 1
        0,  // Material 0 on face 2
        2,  // Material 2 on face 3
        1,  // Material 1 on face 4
        1;  // Material 1 on face 5
        Material  // Material 0 (Green)
          {
            // Red Green Blue Alpha channel
            0.000000;1.000000;0.000000;1.000000;;
            // Specular power
            0.000000;
            // Specular color Red Green Blue
            0.000000;0.000000;0.000000;;
            // Emissive color Red Green Blue
            0.000000;0.000000;0.000000;;
          }

        Material  // Material 1 (Red)
          {
            // Red Green Blue Alpha channel
            1.000000;0.000000;0.000000;1.000000;;
            // Specular power
            0.000000;
            // Specular color Red Green Blue
            0.000000;0.000000;0.000000;;
            // Emissive color Red Green Blue
            0.000000;0.000000;0.000000;;
          }

        Material  // Material 2 (Blue)
          {
            // Red Green Blue Alpha channel
            0.000000;0.000000;1.000000;1.000000;;
            // Specular power
            0.000000;
            // Specular color Red Green Blue
            0.000000;0.000000;0.000000;;
            // Emissive color Red Green Blue
            0.000000;0.000000;0.000000;;
          }
      }

    MeshNormals
      {
        6;  // 6 normals
        0.000000;0.000000;0.000000;,  // Normal face 0
        0.000000;0.000000;0.000000;,  // Normal face 1
        0.000000;0.000000;0.000000;,   // Normal face 2
        0.000000;0.000000;0.000000;,   // Normal face 3
        0.000000;0.000000;0.000000;,   // Normal face 4
        0.000000;0.000000;0.000000;;   // Normal face 5
        6;  // 6 faces
        4;0,0,0,0;,  // 4 vertices, normal #0 for face 0
        4;1,1,1,1;,  // 4 vertices, normal #1 for face 1
        4;2,2,2,2;,  // 4 vertices, normal #2 for face 2
        4;3,3,3,3;,  // 4 vertices, normal #3 for face 3
        4;4,4,4,4;,  // 4 vertices, normal #4 for face 4
        4;5,5,5,5;;  // 4 vertices, normal #5 for face 5
      }

    MeshTextureCoords
      {
        8;  // 8 Vertices
        0.000000;0.000000;,  // No texture, we leave
        0.000000;0.000000;,  // these coordinates at 0
        0.000000;0.000000;,
        0.000000;0.000000;,
        0.000000;0.000000;,
        0.000000;0.000000;,
        0.000000;0.000000;,
        0.000000;0.000000;;
      }
  }
```

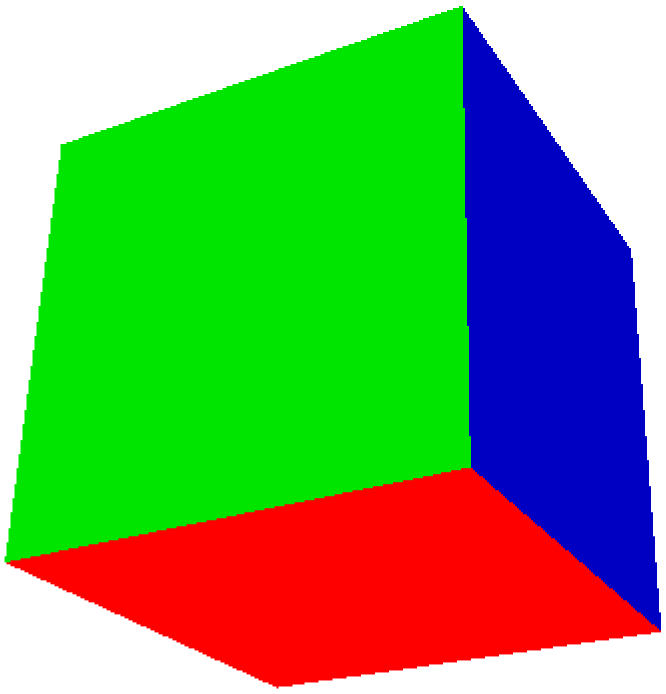
If we look at this cube from below, we see that its bottom is in shadow as the light source shines from above in the simulation.



If we however wants the bottom to be a light source itself, and as that not affected by shadows, we can use the Emissive color parameter in the Material definition. We set red color to 1 in the Emissive color parameter for material 1:

```
        Material  // Material 1 (Red)
          {
            // Red Green Blue Alpha channel
            1.000000;0.000000;0.000000;1.000000;;
            // Specular power
            0.000000;
            // Specular color Red Green Blue
            0.000000;0.000000;0.000000;;
            // Emissive color Red Green Blue
            1.000000;0.000000;0.000000;;
          }
```

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

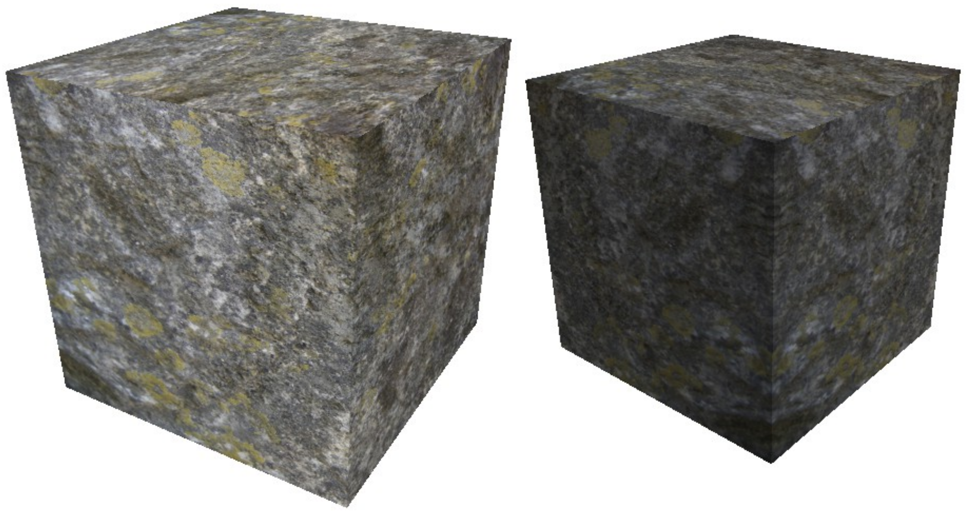


There is also a possibility to combine texture and material color. In the examples so far, we have used a white material on which we have applied the texture. This makes the texture colors to be shown as they are in the original texture file.

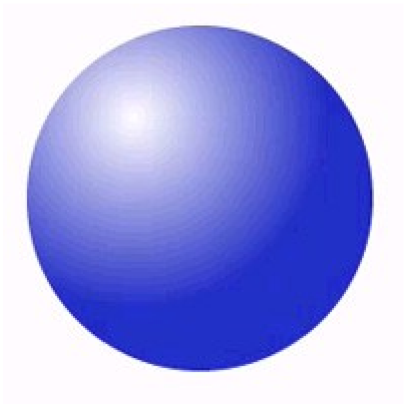
If we want to make the texture darker, we change the material color to some shade of gray. Let's compare the cube with the rock texture with white background and a gray background. We change the material color for the rock cube to:

```
        Material  // Material number 0
          {
            // Red Green Blue and Alpha channel
            0.600000;0.600000;0.600000;1.000000;;
            // Specular power
            0.000000;
            // Specular color Red Green Blue
            0.000000;0.000000;0.000000;;
            // Emissive color Red Green Blue
            0.000000;0.000000;0.000000;;

            TextureFilename
              {
                "RockTexture.bmp";
              }
          }
```

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



We have not mentioned the parameters for specular lighting so far. Specular lighting is about the reflection highlights that can appear especially on curved shiny surfaces, such as a shiny sphere:



The color of this reflection is usually set to white, that means that the specular color RGB is 1, 1, 1.

The specular power parameter takes values from 0 to infinity, but usually ranges between 1..128. The higher value, the sharper (but smaller) and brighter will the reflection be.

Further information cannot be given here, as it has not been possible to view any effect of the settings of these parameters while in OpenBVE Object Viewer viewing built objects.

BLANK PAGE

Chapter 4

Transparency

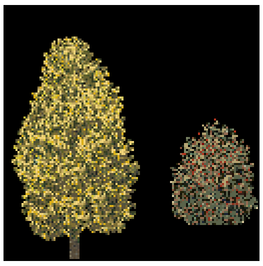
There is no special statement for setting transparency for parts of textures in the x object file format. There is 2 ways of achieving transparency, which are of different usability:

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 decimal value in the range 0 (fully transparent) to 1 (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 other way is to use the default transparent color black (RGB 0,0,0). By this full transparency is achieved by setting the color of the parts of a texture that we want transparent to that color. If we want visible black, we can use a close-to-black value (such as RGB 1,1,1). This is much less processing intensive.

Further information on the alpha-channel method cannot be given here, as it has not been possible to view any effect of the settings of this parameter while in OpenBVE Object Viewer viewing built objects.

For the other method with black (RGB 0,0,0) as transparent color, we start with a 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.

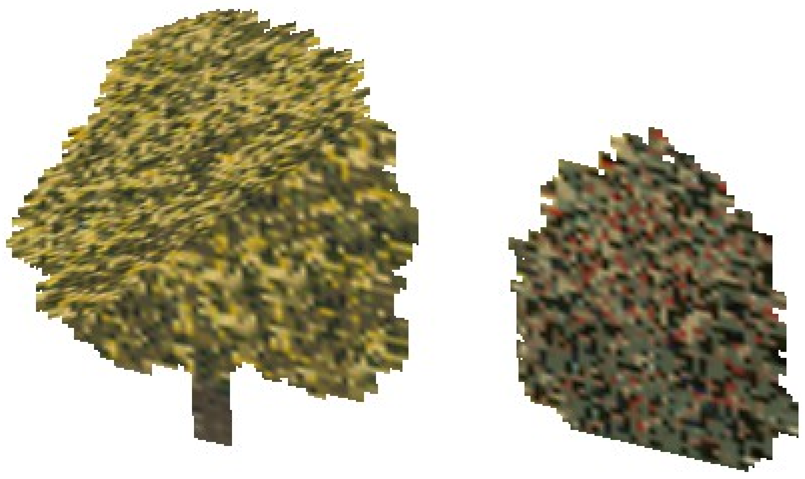


Then we start with the Mesh statement to build a plane vertical surface with one face to place this texture on, then with the Material statement creates a white material with the texture mentioned above as texture using the TextureFilename statement. We leave the normals as 0 in the MeshNormals statement to get them automatically calculated, and finally attaches the texture to the vertices with the MeshTextureCoords statement.

The x format object file will look like this:

```
xof 0303txt 0032

Header
  {
    1;  // Major version
    0;  // Minor version
    1;  // Text file
  }

Mesh
  {
    4;  // 4 vertices
    -20.000000;9.000000;0.000000;,  // Vertex 0
    -20.500000;0.000000;0.000000;,  // Vertex 1
     0.000000;-5.000000;0.000000;,  // Vertex 2
     0.000000;14.000000;0.000000;;  // Vertex 3
    1;  // 1 face
    4;3,2,1,0;;  // 4 vertices, face 0

    MeshMaterialList
      {
        1;  // 1 material
        1;  // 1 face
        0;  // Material 0 to face 0

        Material  // Material 0
          {
            // Red Green Blue Alpha channel
            1.000000;1.000000;1.000000;1.000000;;
            // Specular power
            0.000000;
            // Specular color Red Green Blue
            0.000000;0.000000;0.000000;;
            // Emissive color Red Green Blue
            0.000000;0.000000;0.000000;;
            TextureFilename
              {
                "Tree_Pair.bmp";
              }
          }
      }

    MeshNormals
      {
         1;   // 1 normal
         0.000000;0.000000;0.000000;;   // Normal 0
         1;   // 1 face
         4;0,0,0,0;;   // 4 vertices, normal 0 to face 0
      }

    MeshTextureCoords
      {
        4;   // 4 vertices
        0.000000;0.000000;,   // Vertex 0
        0.000000;1.000000;,   // Vertex 1
        1.000000;1.000000;,   // Vertex 2
        1.000000;0.000000;;   // Vertex 3
      }
  }
```

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



Chapter 5

Calculating the mesh normals

The mesh normals, which are vectors, needs to be calculated for each face and be applied to each vertex that is part of the face. For a vertex shared with more than one face, there will be one normal in that vertex for each face. The normals are perpendicular to the surface of the face. The length of the normal is usually 1.

The normals are of great importance for the optical behavior of the object when it comes to light and shadows. The experienced developer may play with the normal vectors to change the optical properties of the object.

To manually calculate normals is a bit tricky, but as with the earlier csv or b3d object formats, the normals can automatically calculated. They could be manually entered if one want to tweak the normals.

To calculate the normal for a face, we go through these steps:

1. Select 3 vertices from the list of vertices for the face which you want to calculate the normal of.
2. Calculate 2 vectors to represent 2 sides of the triangle formed by the 3 vertices selected in step 1.
3. Calculate the normal vector to these 2 vectors from step 2.
4. Normalize the normal vector from step 3 so its length becomes 1.

Step 1:

We select 3 vertices which are represented by the points **A**, **B** and **C**.

Step 2:

The orientation of each polygon face surface, even if the face is made with >3 vertices, can be defined by 3 of its vertices, if these vertices are not on a straight line. Therefore we pick the coordinates for 3 vertices from the face for which we want to calculate the normal. The vertices should be picked I clockwise order,

Then for a triangle with the vertices **A**-**B**-**C** we can calculate two vectors:

$$\vec{U} = \boldsymbol{B} - \boldsymbol{A}$$
$$\vec{V} = \boldsymbol{C} - \boldsymbol{A}$$

Step 3:

Then we calculate the not yet normalized normal vector $\vec{R}$ as the cross product of the two vectors $\vec{U}$ and $\vec{V}$:

$$\vec{R}=\vec{U}\times\vec{V}=\begin{bmatrix} U_y\cdot V_z-U_z\cdot V_y \\ U_z\cdot V_x-U_x\cdot V_z \\ U_x\cdot V_y-U_y\cdot V_x \end{bmatrix}$$

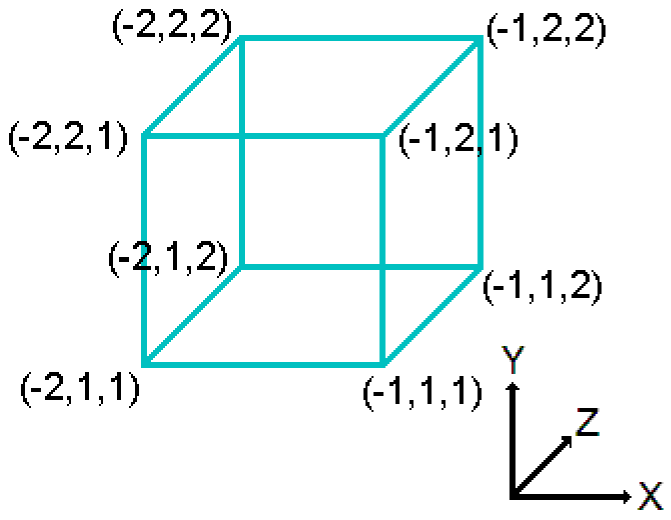
Step 4:

The length of the normal vector $\vec{R}$ depends on the length of and angle between the vectors $\vec{U}$ and $\vec{V}$. Before using the normal vector, we should normalize the normal vector to have the length of 1.

$$\vec{N}=\left[\frac{R_x}{|\vec{R}|},\frac{R_y}{|\vec{R}|},\frac{R_z}{|\vec{R}|}\right] \text{ there } |\vec{R}|=\sqrt{R_x^2+R_y^2+R_z^2}$$

Example 1:

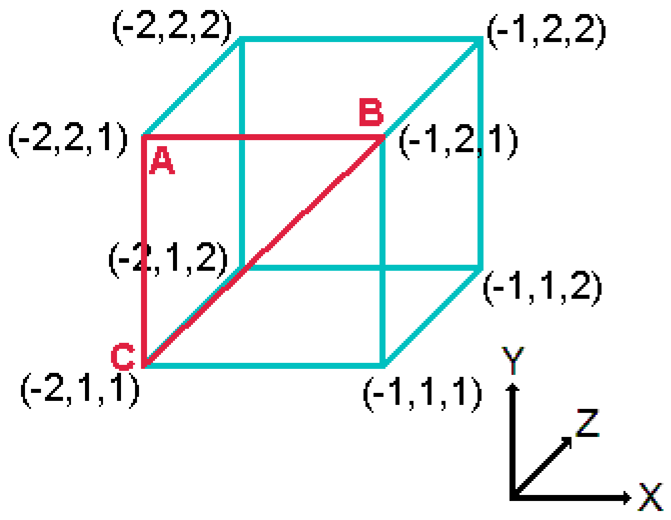
This example describes how to calculate one of the normals on a cubical object.



The cube is made up with a mesh of 8 vertices. Every side of the cube is a face. We chose to calculate the normal for the front face.

Step 1:

We pick 3 of the vertices to be the corners **A**, **B** and **C** of a triangle: **A** = (-2,2,1) **B**=(-1,2,1) **C**=(-2,1,1). The vertices should be picked in clockwise order.



Step 2:

We calculate the vectors $\vec{U}$ and $\vec{V}$.

$$\vec{U} = \boldsymbol{B} - \boldsymbol{A} = \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} - \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} = \begin{bmatrix} B_x - A_x \\ B_y - A_y \\ B_z - A_z \end{bmatrix} = \begin{bmatrix} (-1)-(-2) \\ 2-2 \\ 1-1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\vec{V} = \boldsymbol{C} - \boldsymbol{A} = \begin{bmatrix} C_x \\ C_y \\ C_z \end{bmatrix} - \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} = \begin{bmatrix} C_x - A_x \\ C_y - A_y \\ C_z - A_z \end{bmatrix} = \begin{bmatrix} (-2)-(-2) \\ 1-2 \\ 1-1 \end{bmatrix} = \begin{bmatrix} 0 \\ (-1) \\ 0 \end{bmatrix}$$

Step 3:

Now we can calculate the un-normalized normal vector $\vec{R}$ to the front face by calculation the cross product of the 2 vectors $\vec{U}$ and $\vec{V}$:

$$\vec{R} = \vec{U} \times \vec{V} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ (-1) \\ 0 \end{bmatrix} = \begin{bmatrix} U_y V_z - U_z V_y \\ U_z V_x - U_x V_z \\ U_x V_y - U_y V_x \end{bmatrix} = \begin{bmatrix} 0 \cdot 0 - 0 \cdot (-1) \\ 0 \cdot 0 - 1 \cdot 0 \\ 1 \cdot (-1) - 0 \cdot 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ (-1) \end{bmatrix}$$

Step 4:

We may see that this vector has the length of 1, but for the completeness of the calculations, we do the normalizing calculation for this vector $\vec{R}$ which gives us the normalized normal vector $\vec{N}$:

$$|\vec{R}|=\sqrt{R_x^2+R_y^2+R_z^2}=\sqrt{0^2+0^2+(-1)^2}=\sqrt{1}=1$$

$$\vec{N}=\left[\frac{R_x}{|\vec{R}|},\frac{R_y}{|\vec{R}|},\frac{R_z}{|\vec{R}|}\right]=\left[\frac{0}{1},\frac{0}{1},\frac{(-1)}{1}\right]=\begin{bmatrix}0\\0\\(-1)\end{bmatrix}$$

Now we are done, (0, 0, -1) is the normal for the front face, and will be applied to the 4 vertices that make this face.

The calculations needs to be repeated to find the normal vectors for the remaining 5 faces.

Probably needless to say, this is a task that is very time consuming for complicated objects with many vertices. But the steps necessary to perform the calculations is well suited for automation in almost any high level programming language.

BLANK PAGE

Chapter 6

Statements in Alphabetical Order

Header

```
Header
  {
    Major version;
    Minor version;
    File type flag
  }
```

Variable	**Type**	**Notice**
Major version	1 digit number	Should be 1
Minor version	1 digit number	Should be 0
File type flag	1 digit number	0=Binary file 1=Text file Should be 1

The header statement defines version and file type for the Direct3D Retained Mode of the DirectX file format. The statement should read according to example 1 (The comments are not necessary):

Example 1:

```
Header
  {
    1;   // Major version
    0;   // Minor version
    1;   // Text file
  }
```

This describes an x object file of Direct3D Retained Mode version 1.0 in text format.

Material

```
Material
  {
    Face color R;G;B;A;;
    Power;
    Specular color R;G;B;;
    Emissive color R;G;B;;

    TextureFilename
      {
        // Se TextureFilename
      }
  }
```

Variable	**Type**	**Notice**
Face color R G B A	Decimal numbers from 0..1	4 positive decimal numbers separated by ; that indicates red, green, blue and alpha component.
Power	Decimal number	A decimal number indicating the specular exponent of the material.
Specular color R G B	Decimal numbers from 0..1	3 positive decimal numbers separated by ; that indicates red, green and blue color.
Emissive color R G B	Decimal numbers from 0..1	3 positive decimal numbers separated by ; that indicates red, green and blue color.

The Material statement defines some of the optical properties of the colors to be applied on one or more faces. If the Material statement also contains a TextureFilename statement, a texture for the material will be defined and visible on the faces that uses this material.

The face color defines the basic color of the face. The specular exponent and color of the face defines the optical properties for such reflections that may occur from shiny surfaces such as metal, glass or glossy painted surfaces.

The emissive color defines which color of the object that emits light itself. This color is not affected by shades, and may be used on lighting armature etc.

Example 1:

```
Mesh
  {
    //Se Mesh

    MeshMaterialList
      {
        // Se MeshMaterialList

        Material
          {
            1.000000;1.000000;1.000000;1.000000;;
            0.000000;
            0.000000;0.000000;0.000000;;
            0.000000;0.000000;0.000000;;

            Texture Filename
              {
                // Se TextureFilename
              }
          }
      }
    MeshNormals
      {
        // Se MeshNormals
      }

    MeshTextureCoords
      {
        // Se MeshTextureCoords
      }
  }
```

This describes a pure white material (R=1.0 G=1.0 B=1.0 A=1.0), with no reflective properties (specular power =0), and no emissive color.

Mesh

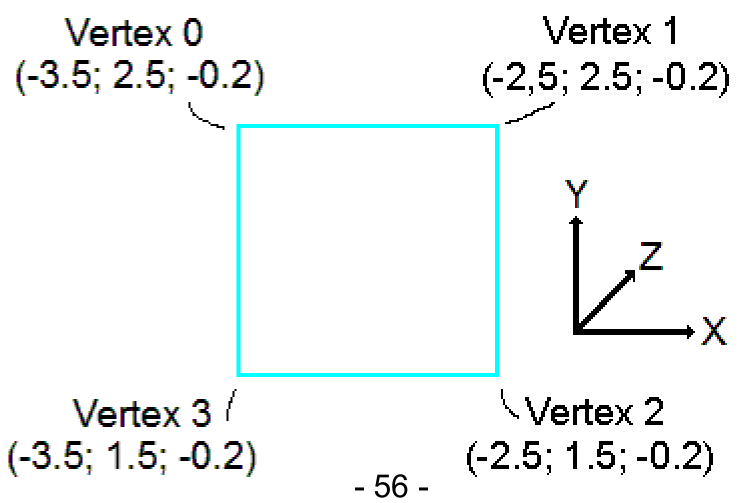
```
Mesh
  {
    Number of vertex;
    Vertex0 X; Vertex0 Y; Vertex0 Z;,
    ..
    Vertexn X; Vertexn Y; Vertexn Z;,
    Number of faces;
    Number of vertices for face0; Vertex, .. vertex;;
    ..
    Number of vertices for facen; Vertex, .. vertex;;

    MeshMaterialList
      {
        // Se Mesh Material list

        Material
          {
            // Se Material

            TextureFilename
              {
                // Se TextureFilename
              }
          }
      }

    MeshNormals
      {
        // Se MeshNormals
      }
    MeshTextureCoods
      {
        // Se MeshTextureCoords
      }
  }
```

Variable	**Type**	**Notice**
Number of vertex	Digit(s)	Positive integer
Vertex X coordinate	Decimal	May be positive or negative, decimal mark is dot **.**
Vertex Y coordinate	Decimal	May be positive or negative, decimal mark is dot **.**
Vertex Z coordinate	Decimal	May be positive or negative, decimal mark is dot **.**
Number of faces	Digit(s)	Positive integer
Number of vertices for a face	Digit(s)	Positive integer, specifies the number of vertices for each face
Vertex	Digit(s)	Positive integer (notice that the first vertex specified above has number 0)

The Mesh statement defines a mesh of 3 or more vertices to make up an object, the faces of the mesh built and may also contain other statements as MeshMaterialList, Material, TextureFilename, MeshNormals and MeshTextureCoords.

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.

Example 1:



```
Mesh // We will define a simple square with the side 1 m
  {
    4;  // 4 vertex
    -3.500000;2.500000;-0.200000;,  // Vertex 0
    -2.500000;2.500000;-0.200000;,  // Vertex 1
    -2.500000;1.500000;-0.200000;,  // Vertex 2
    -3.500000;1.500000;-0.200000;;  // Vertex 3
    1;    // One face
    4;0,1,2,3;  // Face with 4 vertex, clockwise
                // order vertex 0,1,2,3

    MeshMaterialList
      {
        // Se MeshMaterialList

        Material
          {
            // Se Material

            Texture Filename
              {
                // Se TextureFilename
              }
          }
      }

    MeshNormals
      {
        // Se MeshNormals
      }

    MeshTextureCoords
      {
        // Se MeshTextureCoords
      }
  }
```

This describes the mesh vertices and face for a simple square with the side 1 m. The position of the square is the the left of the coordinate's systems origin (0;0;0), hence negative X coordinates, above the origin, hence positive Y coordinates and slightly in front of origin, hence the negative Z coordinates.

The one and only face is towards the viewer, and is defined by the clockwise order of vertex 0,1,2,3. The first defined face has number 0, which is of importance in the other statements that is contained within the Mesh statement.

MeshMaterialList

```
MeshMaterialList
  {
    Number of materials;
    Number of faces;
    List of material numbers for each face;

    Material
      {
        // Se Material
        TextureFilename
          {
            // Se TextureFilename
          }
      }
  }
```

Variable	**Type**	**Notice**
Number of materials	Digit(s)	Positive integer
Number of faces	Digit(s)	Positive integer
List of material numbers for each face	Digit(s)	Comma separated list of integers from 0 and positive

The statement MeshMaterialList is contained within the Mesh statement. MeshMaterialList contains a short list of 3 variables and other statements that defines the material(s) and, if required, texture(s) of one or more faces.

Example 1:

```
Mesh
  {
    // Se Mesh

    MeshMaterialList
      {
        1;   // Number of materials;
        2;   // Number of faces;
        0,   // Material number for face 0
        0;   // Material number for face 1

        Material  // Material 0
          {
            // Se Material

            Texture Filename
              {
                // Se TextureFilename
              }
          }
      }
    MeshNormals
      {
        // Se MeshNormals
      }

    MeshTextureCoords
      {
        // Se MeshTextureCoords
      }
  }
```

This describes a MeshMaterialList statement containing 1 material to be used on 2 faces. The faces are earlier defined in the Mesh statement that contains the MeshMaterialList statement.

Example 2:

A special case is when the same material should be added to all faces. Then it is enough to specify 1 material 1 face and material 0 for face 0:

```
Mesh
  {
    // Se Mesh

    MeshMaterialList
      {
        1;   // Number of materials;
        1;   // Number of faces;
        0;   // Material number for face 0

        Material  // Material 0
          {
            // Se Material

            Texture Filename
              {
                // Se TextureFilename
              }
          }
      }
    MeshNormals
      {
        // Se MeshNormals
      }

    MeshTextureCoords
      {
        // Se MeshTextureCoords
      }
  }
```

MeshNormals

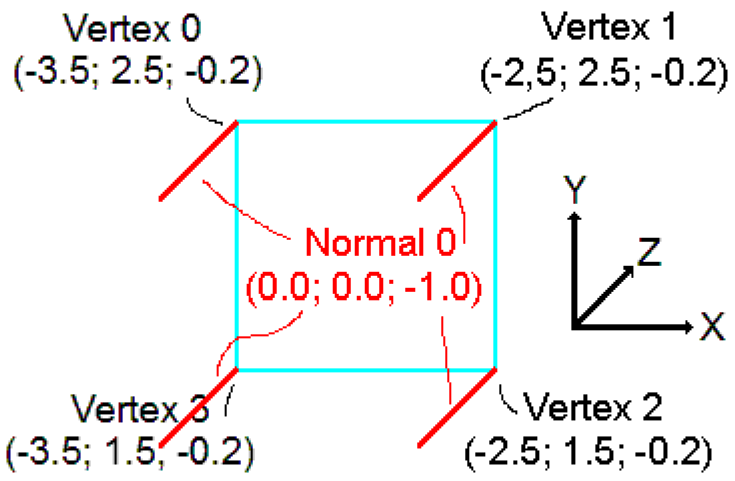
```
MeshNormals
  {
    Number of normals
    Coordinates X; Y; Z;
    Number of face normals
    Number of vertices for a face0
    Normal number for each vertex in face0
    ..
    Number of vertex for a facen;
    Normal number for each vertex in facen
  }
```

Variable	**Type**	**Notice**
Number of Normals	Digit(s)	Positive integer
Coordinates X; Y; Z;;	Decimal numbers	Decimal numbers from 0..1 defining the coordinates of a normal vector of the length 1 from it's origin (0;0;0)
Number of face normals	Digit(s)	Positive integer, should correspond to the number of faces defined in Mesh
Number of vertices for a face;	Digit(s)	The number of vertex for a face
Normal number for each vertex in face.	Digits	Comma separated list

The MeshNormals statement defines normals for each vertex which are a part of a face. The normal is perpendicular to the surface of the face. For a vertex located in corners between 2 or more faces, the normal of each adjacent face will be applied to that vertex. The length of the normal is usually 1.

The normals are of great importance for the optical behavior of the object when it comes to light and shadows. The experienced developer may play with the normal vectors to change the optical properties of the object.

Example 1:



```
Mesh
  {
    // Se Mesh

    MeshMaterialList
      {
        // Se MeshMaterial List

        Material
          {
            // Se Material

            Texture Filename
              {
                // Se TextureFilename
              }
          }
      }

    MeshNormals
      {
        1;
        0.000000;0.000000;-1.000000;;
        1;
        4;0,0,0,0;;
      }
    MeshTextureCoords
       {
         // Se MeshTextureCoords
       }
  }
```

This describes adding normals to the simple square with one face that was in example 1 in the description of the Mesh statement. The face has 4 vertices and to each vertex is added normal$_0$ that is perpendicular to the face. The normal is pointing towards the viewer along the negative part of the Z axis.

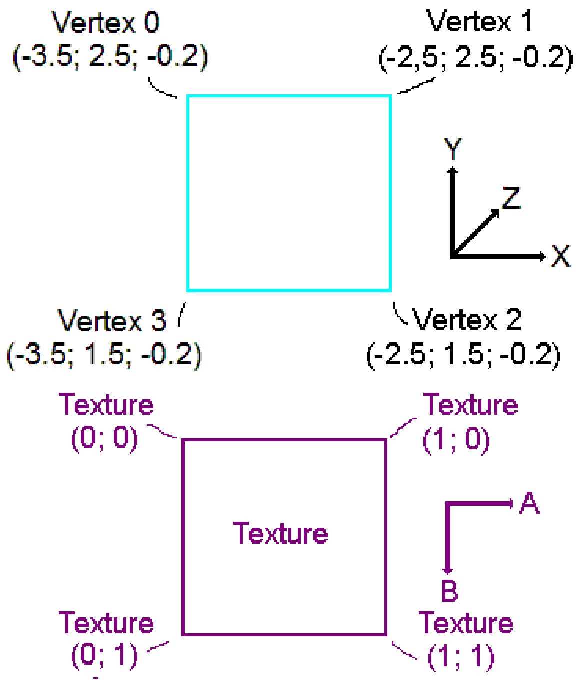
MeshTextureCoords

```
MeshTextureCoords
  {
    Number of coordinates;
    2D Texture coordinate₀ A; B;,
    ..
    2D Texture coordinateₙ A; B;,
  }
```

Variable	**Type**	**Notice**
Number of coordinates	Digit(s)	Positive integer
2D Texture coordinates	Decimal numbers	Coordinates A and B of texture to attach to a vertex that is a part of a face

The MeshTextureCoords statement ”attaches” a texture to a face. For each of the face's vertex, a coordinate of the **texture picture** is attached, and the texture is ”stretched” over the face.

Example 1:



```
Mesh
  {
    // Se Mesh

    MeshMaterialList
      {
        // Se MeshMaterial List

        Material
          {
            // Se Material

            Texture Filename
              {
                // Se TextureFilename
              }
          }
      }

    MeshNormals
      {
        // Se MeshNormals
      }

    MeshTextureCoords
      {
        4;
        0.000000;0.000000;,
        1.000000;0.000000;,
        1.000000;1.000000;,
        0.000000;1.000000;;
      }
  }
```

This describes adding av texture image to a simple square object. The object has 4 vertices and to each of them are the corresponding texture coordinates A and B are written. Notice that the order of vertices are clockwise 0 to 3.

TextureFilename

```
TextureFilename
     {
          "Filename.bmp";
     }
```

<u>**Variable**</u>	<u>**Type**</u>	<u>**Notice**</u>
Filename.bmp	String of text	The filename for the file containing the texture.

The TextureFilename statement is contained within a Material statement and defines the filename for the file that contains the texture that should be applied to the material.

The texture should be a bitmap file, the width and height in pixels of which should be a power of 2. That is, the number should be 2^n where n is a positive integer. Such numbers for 2^1 to 2^{10} are 2,4,8,16,32,64,128,256, 512, 1024.

If the texture image is located in another folder than the x object file, a relative path may precede the file-name, such as “..\..\textures\Filename.bmp” or, for a sub-folder, “textures\Filename.bmp”.

Example 1:

```
Mesh
  {
    // Se Mesh

    MeshMaterialList
      {
        // Se MeshMaterialList

        Material
          {
            // Se Material

            TextureFilename
              {
                ”PoorAdhesion.bmp”
              }
          }
      }
    MeshNormals
      {
        // Se MeshNormals
      }

    MeshTextureCoords
      {
        // Se MeshTextureCoords
      }
  }
```

This describes a file with the name PoorAdhesion.bmp that should be applied as texture.

xof

`xof` *`major version number+minor version number+format type float size`*

Variable	**Type**	**Notice**
Major version number	2 digit number	Should be 03
Minor version number	2 digit number	Should be 03
Format type	3 characters	Should be txt
Float size	4 digit number	Should be 0032

The beginning of an x object file for OpenBVE should contain the ”Magic number” `xof` followed by the major + minor version number + file format and finally the size in bits of float numbers. The string should read according to example one:

Example 1:

```
xof 0303txt 0032
```

This describes an x object file version 03.03 in text format and with 32-bit float numbers.

//

```
// Comment text
```

Comment text in an xobject 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:

```
Mesh
     {
          4; //Number of vertices
          -3.490000;2.500000;-0.200000  // Vertex 0
          -2.490000;2.500000;-0.200000  // Vertex 1
          -2.490000;1.500000;-0.200000  // Vertex 2
          -3.490000;1.500000;-0.200000  // Vertex 3
          1; // Number of faces
          4;0,1,2,3;; // Number of vertices, face vertex order
```

These are comments on lines with statements.