A simple ASE file of a square polygon

NOTE : Since initially writing this page I have done a lot more work with the ASE file format. I have written a yacc/lex based parser that reads ASE files into a set of C++ objects, and have also made public the Unreal ASE Converter. You can find some details of the ASE parser here.

- solosnake, 2nd March 2005

Please note that this is not an autorised or definitive analysis of the ASE file format. Instead I have addressed only the contents absalutely neccessary for creating and rendering real time 3D scenes, with referance to the DXA file format. Below is a complete ascii scene exporter output for a simple square polygon: the data extracted for the DXA file I have highlighted. The format is very self explanatory.

```
*3DSMAX_ASCIIEXPORT 200
*COMMENT "AsciiExport Version 2.00 - Mon Feb 12 17:49:55 2001"
*SCENE {
    *SCENE_FILENAME "testplane.max"
    *SCENE_FIRSTFRAME 0
    *SCENE_LASTFRAME 100
    *SCENE_FRAMESPEED 30
    *SCENE_TICKSPERFRAME 160
    *SCENE_BACKGROUND_STATIC 0.0000 0.0000 0.0000
    *SCENE_AMBIENT_STATIC 0.0431 0.0431 0.0431
    }

*MATERIAL_LIST {

    *MATERIAL_COUNT 1
    *MATERIAL 0 {

        *MATERIAL_NAME "Material #1"
        *MATERIAL_CLASS "Standard"
        *MATERIAL_AMBIENT  0.1791 0.0654 0.0654
        *MATERIAL_DIFFUSE  0.5373 0.1961 0.1961
        *MATERIAL_SPECULAR 0.9000 0.9000 0.9000
        *MATERIAL_SHINE 0.2500
        *MATERIAL_SHINESTRENGTH 0.0500
        *MATERIAL_TRANSPARENCY 0.0000
        *MATERIAL_WIRESIZE 1.0000
        *MATERIAL_SHADING Blinn
        *MATERIAL_XP_FALLOFF 0.0000
        *MATERIAL_SELFILLUM 0.0000
        *MATERIAL_FALLOFF In
        *MATERIAL_XP_TYPE Filter
        *MAP_DIFFUSE {

            *MAP_NAME "Map #1"
            *MAP_CLASS "Bitmap"
            *MAP_SUBNO 1
            *MAP_AMOUNT 1.0000
            *BITMAP "C:\Textures\MyTexture.bmp"
            *MAP_TYPE Screen
            *UVW_U_OFFSET 0.9000
            *UVW_V_OFFSET 1.0300
            *UVW_U_TILING 1.0000
            *UVW_V_TILING 1.0000
            *UVW_ANGLE 0.0000
            *UVW_BLUR 1.0000
            *UVW_BLUR_OFFSET 0.0000
            *UVW_NOUSE_AMT 1.0000
            *UVW_NOISE_SIZE 1.0000
            *UVW_NOISE_LEVEL 1
            *UVW_NOISE_PHASE 0.0000
```

```
                *BITMAP_FILTER Pyramidal
                }

            }

        }

    *GEOMOBJECT {

        *NODE_NAME "Box01"
        *NODE_TM {

            *NODE_NAME "Box01"
            *INHERIT_POS 0 0 0
            *INHERIT_ROT 0 0 0
            *INHERIT_SCL 0 0 0
            *TM_ROW0 1.0000 0.0000 0.0000
            *TM_ROW1 0.0000 1.0000 0.0000
            *TM_ROW2 0.0000 0.0000 1.0000
            *TM_ROW3 0.0000 -35.1807 0.0000
            *TM_POS 0.0000 -35.1807 0.0000
            *TM_ROTAXIS 0.0000 0.0000 0.0000
            *TM_ROTANGLE 0.0000
            *TM_SCALE 1.0000 1.0000 1.0000
            *TM_SCALEAXIS 0.0000 0.0000 0.0000
            *TM_SCALEAXISANG 0.0000
            }

        *MESH {

            *TIMEVALUE 0
            *MESH_NUMVERTEX 4
            *MESH_NUMFACES 2
            *MESH_VERTEX_LIST {

                *MESH_VERTEX 0 -19.6285 -52.9010 -16.4621
                *MESH_VERTEX 1  15.8121 -52.9010 -16.4621
                *MESH_VERTEX 2 -19.6285 -52.9010  18.9785
                *MESH_VERTEX 3  15.8121 -52.9010  18.9785
                }

            *MESH_FACE_LIST {

                *MESH_FACE 0: A: 0 B: 1 C: 3 AB: 1 BC: 1 CA: 0 *MESH_SMOOTHING
                4 *MESH_MTLID 4
                *MESH_FACE 1: A: 3 B: 2 C: 0 AB: 1 BC: 1 CA: 0 *MESH_SMOOTHING
                4 *MESH_MTLID 4
                }

            *MESH_NUMTVERTEX 12
            *MESH_TVERTLIST {

                *MESH_TVERT  0 0.0000 0.0000 0.0000
                *MESH_TVERT  1 1.0000 0.0000 0.0000
                *MESH_TVERT  2 0.0000 1.0000 0.0000
                *MESH_TVERT  3 1.0000 1.0000 0.0000
                *MESH_TVERT  4 0.0000 0.0000 0.0000
                *MESH_TVERT  5 1.0000 0.0000 0.0000
                *MESH_TVERT  6 0.0000 1.0000 0.0000
                *MESH_TVERT  7 1.0000 1.0000 0.0000
                *MESH_TVERT  8 0.0000 0.0000 0.0000
                *MESH_TVERT  9 1.0000 0.0000 0.0000
                *MESH_TVERT 10 0.0000 1.0000 0.0000
                *MESH_TVERT 11 1.0000 1.0000 0.0000
                }

            *MESH_NUMTVFACES 2
            *MESH_TFACELIST {

                *MESH_TFACE 0 4 5 7
                *MESH_TFACE 1 7 6 4
                }

            *MESH_NORMALS {
            *MESH_FACENORMAL 0 0.0000 -1.0000 0.0000

                *MESH_VERTEXNORMAL 0 0.0000 -1.0000 0.0000
                *MESH_VERTEXNORMAL 1 0.0000 -1.0000 0.0000
                *MESH_VERTEXNORMAL 3 0.0000 -1.0000 0.0000
                *MESH_FACENORMAL 1 0.0000 -1.0000 0.0000
                *MESH_VERTEXNORMAL 3 0.0000 -1.0000 0.0000
                *MESH_VERTEXNORMAL 2 0.0000 -1.0000 0.0000
                *MESH_VERTEXNORMAL 0 0.0000 -1.0000 0.0000
                }

            }
```

```
*PROP_MOTIONBLUR 0
*PROP_CASTSHADOW 1
*PROP_RECVSHADOW 1
*MATERIAL_REF 0
}
```

SCENE_AMBIENT_STATIC is the marker for the value of the scenes ambient lighting level, expressed as RGB floats. This is analogous to DirectX's D3DRENDERSTATE_AMBIENT setting. The ambient light is by how much every object in the scene is lit by default.

MATERIAL_COUNT is followed by an integer, the number of materials in the scene: note that if there are NO materials this marker is still present, but followed by a zero. In my example I have used only a single material. Each new material will be marked by *MATERIAL at its start.

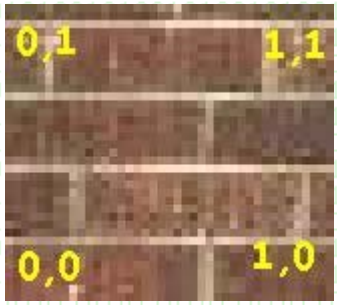
MATERIAL_AMBIENT and **MATERIAL_DIFFUSE** are float RGB values which represent what color the material is and how it interacts with lighting. The ambient and diffuse values are used in determining the materials apparant colour in a scene. The resulting colour is calculated using the vertex normal data (per vertex in DirectX) and the scenes ambient light against the materials ambient light, and any diffuse lights against the materials diffuse value. In general these values should be the same except for special effects.

MATERIAL_SPECULAR This is a float RGB colour value which represent what colour the material reflects or highlights. This value ties in with the dvPower member of a D3DMATERIAL7. When converting from ASE files which do not have a POWER value DXAtoASE sets the default dvPower to zero.

BITMAP This is the path and name of the bitmap used to texture the object in the ASE file. Note that DXA files do not copy the path data but instead only extract the actual texture filename to use. If a material does not contain a texture, then a DXA file mesh object will have a texture of NOTEXTURE.

UVW_U_OFFSET and **UVW_V_OFFSET** The values represent the offset of the texture on the on the triangle to be textured. ASE files and DirectX agree on the meaning of the U value, using it to offset the texture in the conventional 'x' direction, to the right as looked at, however they disagree on the V value. In DirectX displacement by V moves a texture DOWN whereas in ASE files V represents displacemetn up. DXAtoASE compensates for this and maps correctly. Below is illustrated the diference in both sets:

DirectX texture co-ordinates



ASE texture co-ordinates

UVW_U_TILING and **UVW_V_TILING** These values represent how many times the triangle face is to be textured using the texture. Think of the texture as a tile. A UVW_U_TILING value of 1.0 simply means you want one tile to cover the whole polygon face (think of a rectangular polygon and a rectanguler texture). For values greater then 1.0 the texture is applied that many times, so a value of 2.5 means that your texture will appear on the polygon face 2.5 times. The values can also be less then 1.0, eg 0.5 which will stretch half the texture across the entire face of the rectangle. The U and V specify tiling in the directions indicated in the graphics above. DirectX and 3ds max and so ASE files differ in their understanding of U and V tiling. In 3ds max, textures are always centred and tiled from the ends away from the centre, so a U value of 2.0 results in a texture appearing once centred and having one half on either side. in DirectX, a U tiling of 2.0 will result in the texture being simply repeated twice across the face. Both effects can be simulated in the other environment (or corrected) using the U and V offsets. Again, DXA will correct the discrepancy between ASE and DirectX. Note also that although both DirectX and 3ds max allow specification of other forms of tiling, eg mirroring, this does not seem to be output to the ASE file, and so there is no way to tell in an ASE file what type of texture wrapping to use.

GEOMOBJECT This marks the beginning of a new geometric object. Although my example has only one, there may be (usually is) more of these.

MESH_NUMVERTEX This is followed by an integer indicating how many vertices this GEOMOBJECT has. Essentially this is an array of vertices which the MESH_FACE_LIST will index into later.

MESH_NUMFACES This is followed by the integer number of faces in the object. In my example, a simple plane square, there is two faces. The number of faces

equals the number of triangles the object is composed of. It is a good measure of the objects 'size'.

MESH_VERTEX is followed by the integer index number of the vertex, then by that vertex's x, y and z co-ords. Note that ASE files do not have a left hand co-ordinates sytem, but use the conventional 'real world' notation of z being 'up', x increasing rightwards, and y increasing away from you. DXA files use DirectX's left hand system, and convert ASE accordingly.

MESH_FACE I only understanding fully the first half of this row, but thats all thats needed for most 3d applications: a row is broken down as follows:

face number: A: X1 B:X2 C:X3

where X1, X2, X3 represent indices into the above vertex arrray. Thus face 0 in my example above has as its first point vertex number 0 which is x = -19.6285 y = -52.9010 z = -16.4621, its second point is vertex number 1, and its third is vertex 3.

Visible Edges I was contacted by Marc Reilly recently, who suggested that the values AB, BC and CA may be booleans, and that they indicate whether an egde is visible or not. I think he is probably right, as they are related to the edges, and they seem to be only range from 0 to 1. If anyone knows more, or better, please feel free to contact me. Thanks Marc!

MESH_NUMTVERTEX As might be guessed, this is the number of texture co-ordinates in this file. This does not have to equal the number of vertex co-ordinates.

MESH_TVERT A texture vertex appears to be the same as a regular vertex, but in fact most applications will only need the first two values, which are the U and V values for that point.

MESH_NUMTVFACES is followed by the integer number of texture faces. This should ALWAYS equal the MESH_NUMFACES value, as objects cannot be partially textured.

MESH_TFACE A MESH_TFACE is exactly the same in format as a MESH_FACE except it indexes into the MESH_TVERTLIST array.

MESH_NORMALS Following this is the breakdown of the objects normal data. For some reason the ASE here changes its format and instead of having a array of normals which are indexed into it actually just produces a verbatim list of the details.

MESH_FACENORMAL This is the x,y,z vector at right angles to the face it is associated with. Note that this can be calculated from the vertex data anyway, plus the fact that DirectX uses vertex normals to calculate lighting, and we can ignore this data.

MESH_VERTEXNORMAL The vertex normals are specified in the same order that the faces appear in the face list. The integer number refers to the vertex to which it is a normal. Note that it does not neccessarily have to be a true orthonormal vector to its vertex. Objects are made to appear smoooth by altering their vertex normals so that when lit the joining faces all meet at same colours, so masking the ridges. There is usually an option to 'smooth' an object in your ASE file generator.

MATERIAL_REF This refers back to the *MATERIAL marker, and it is the number of the material this GEOMOBJECT uses.

Contents
Previous