OPT Files

OPT Files General

Updated: 19/Jan/99

1. ---------
2. Updated textures

Introduction

While not listed each individual that has contributed to the opt specs from past or present research it is hoped this document will prove to be a summation of that knowledge for all to use.

Conventions

Values are Pascal Longints (Signed 32-bit ,4 bytes) unless noted otherwise.

Floats are (signed 32-bit, 4 byte) floating-point numbers.

Arrays start at 0 to whatever ,denoted as [0..] unless otherwise noted

---

OPT Files have an elaborate offset and header system. They contain the 3D information and textures palettes etc needed to render an XWvT ship. The following is the "general" structures there are many variations to both of these general types.

The General Structure for XWVT is:

- Mesh Directory (02 Header)

- Main Mesh Header
  - Mesh Descriptor (25)
  - # Block 22s (hardpoint block header)
  - 23 Block (rotation/scale etc)
  - Mesh Info Header

  - Mesh Info Header
    - Mesh Vertices (3)
    - Texture Vertices (13)
    - Vertex Normals (11)
    - Mesh Data Header (21)

    - Mesh Data Header (21)
      - Mesh Face Header

      - Mesh Face Header
        - # of Face Texture header (20) not always present
        - # of Face Texture header (07) not always present
        - # of Face Texture header (24) not always present
        - ## of Face Texture header (00) not always present ** appears to be old code
        - # of Face Data Header (1)

The General Structure for BOP is:

- Mesh Directory (02 Header)
  - Main Mesh Header
    - Mesh Descriptor (25)
    - # Block 22s (hardpoint block header)
    - Mesh Vertices (3)
    - Texture Vertices (13)
    - Vertex Normals (11)
    - 23 Block (insertion offset etc)
    - Mesh Info Header

    - Mesh Info Header
      - nulls
      - Mesh Data Header (21)

      - Mesh Data Header (21)
        - Mesh Face Header

        - Mesh Face Header
          - # of Face Texture header (20) not always present
          - # of Face Texture header (07) not always present
          - # of Face Texture header (24) not always present

of Face Texture header (00) not always present ** appears to be old code
of Face Data Header (1)

There are several types of what appears to be offsetts, we'll cover them as we get to them.

The first one is found in the Main Header. It may appear to be an arbitary number, This number in BOP files is a combination of what appaers to be a time stamp plus an offset . When you read the number you must subtract 8 from it's value before you can use it. The resultant number is used as a global offset number. When you encounter a jump offset value you subtract this global number from it to arrive at the offset in decimal from the beginning of file.

This global number can be set to " 8 " and at each place later in the file where an offsett is needed you use file offset(decimal) + 8 to create a valid offset to where you want the offsett to point in the file. This was tested by hand in bop. Bop then created a much bigger number as the apparent time stamp of file creation was added to ea offsett. All bop files created for a mission will have the same global offsett numbers. Later or earlier files will have different global offsetts. Thus you could have the same op1 file with different offsets depending on when the file was created. Op1 files are the same as opt files.

Main Header

```
Main Header
 ------------
 FFFF FFFF                  // FFFF
 Filesize                   // size of file - 8 bytes
 Global Offset Number       // to get the number to use subtract 8
                            // use this number to subtract from jump offsets
                               to arrive at offset from begining of file.
                               to write a file set to "8" and add 8 to all
                               other offsets + file offset from beginning of file
```

Main Jump Header

Next is the 02 Header. This contains the jump offsets to the Main Mesh Headers. One mesh header exists for each piece of the 3D object.

Note : The Jump offsets don`t always point to a Main Mesh Header, at times usualy only the first jump will point to the first entry of a (0 20) Texture Header. It is unkown why this was done. This texture is soetimes used later on sometimes not.as in one of the sattelite opt`s there is an upside down baby picture here :-)

```
02 Header
 ------------
 2                  // (byte) "FE" for bomb.opt
 0                  // (byte)
 NumOfOffsets       // (Longint) the number of jump offsets to Main Mesh Headers ** See Note above
```

```
Jump to Offsets   // Points to first offset
[1..NumOfOffsets] // (beginning at offset 22)
```

Main Mesh Header

XWVT:: Main Mesh Headers can have any number of blocks to read so a file reader should not rely on the (003) byte format to locate this block. Each has different information. The one caveat is the number of block 22 to read. If the Main Mesh header block starts with a 3 there are no block 22's(hardpoint info) to read. An opt file reader to be resonably acurate of what it is encountering should not rely on what will appear in each of these headers instead it should test to see what is at that offsett and take the appropriate action. This is especcially true in bop files where there may be an offsett to a null entry.

BOP:: Main Mesh Headers have even more blocks to read. see the Main Mesh Header BOP structure.

The **Mesh Master Reverse Offset (MMRV).**
This number is similar to the file global offset, however it is used to go back "x" number of bytes, usually if you need to go back to the MeshMaster header. To use the number simply store the meshmaster reverse offset, when you encounter a reverse offset in a Mesh Info header you subtract the MMRVfrom the Info headers RV number to get the number of bytes to go back.

The reverse offset can also be used to get to the next MMH block, Take the first MMH RV offset you encounter in the file, add any Main Jump offset (blk 02) to it. The result will be exactly 178 bytes more than the offset to that MMH blocks list of jumps.

Not really useful if you have "file seek position ability".

With all that above, If the reverse offsetts are set to "null" when writing a file , then they will not be changed when bop writes the opt file. So either they are redundant code ,or some sort of material flags settings. Either way the opt seems no different under 3d conditions with these set to null.

```
Main Mesh Header XWVT
-----------
 0
 0
 NumOfMeshSub headers        // num of block 22 to read = num sub headers-3
 Offset to Jump Offsets
 1
 mesh master reverse offset // Mesh Global reverse offset
 1                          // Not always present, use Jump Offset!
 offset to 25 block         // Collision Box
 num of block 22 offsets    // = above formula
 offset to 23 block         // Insertion offset, and scale or lighting values
 offset to 004 block        // Mesh Info Header
```

BOP Main Mesh Header::

Can contain more than the Xwvt offsets . they also may contain ,Vertice,texture normals,vertice normal blocks, there is a change to the Mesh Info header in this case.

```
Main Mesh Header BOP
------------
 0
 0
 NumOfMeshSub headers        // num of block 22 to read = num sub headers-3
 Offset to Jump Offsets
 1
 mesh master reverse offset // Mesh Global reverse offset
 1                          // Not always present, use Jump Offset!
 offset to 25 block         // Collision Box
 num of block 22 offsets    // = above formula
 offset to 03               // vertices
 offset to 11               // vertice normals
 offset to 13               // texture normals
 offset to 23 block         // Insertion offset, and scale or lighting values
 offset to 004 block        // Mesh Info Header
```

Mesh Descriptor 025 Block Header

```
Mesh Descriptor 025 Block Header
 -----------
 0
 25
 0
 0
 1
 Jump to mesh type
 Mesh type                // see list below
 uk longint
 array[0..11]of single; (float) //Hitzone info see below for note on calculating
 targeting group id          //  longint
 array[0..2] of single; (float) // Targeting box coord ***** see notes below
```

Here is a list of the Mesh Types

3. 0: Default
4. 1: MainHull

5. 2: Wing
6. 3: Fuselage
7. 4: GunTurret
8. 5: Smallgun
9. 6: Engine
10. 7: Bridge
11. 8: ShieldGen
12. 9: EnergyGen
13. 10: Launcher
14. 11: CommSys
15. 12: BeamSys
16. 13: CommandBeam
17. 14: DockingPlat
18. 15: LandingPlat
19. 16: Hangar
20. 17: CargoPod
21. 18: MiscHull
22. 19: Antenna
23. 20: RotWing
24. 21: RotGunTurret
25. 22: RotLauncher
26. 23: RotCommSys
27. 24: RotBeamSys
28. 25: RotCommandBeam
29. 26: Custom1
30. 27: Custom2
31. 28: Custom3
32. 29: Custom4
33. 30: Custom5
34. 31: Custom6

HitZone floats array[0..11] of single how to calculate values

The floats are in 4 sets representing X,Y,Z as follows.

35. Floats [0..2] = The SPAN X,Y,Z
36. Floats [3..5] = Center point of HitZone (very close to block 23 center point)
37. Floats [6..8] = MIN X,Y,Z of the USED in face data vertices
38. Floats [9..11]= MAX X,Y,Z of the USED in face data vertices

How to calculate The SPAN X,Y,Z.

```
SpanX = MaxX subtract MinX
```

,repeat for Y,Z.

How to calculate the HitZone center point.

```
CenterX = SpanX divided by negative 2 then add MaxX
```

repeat for Y,Z.

Targeting Group ID

The targeting group id is used to describe which and where the small yellow targeting boxes show up in the Hud with the "," key. You can associate several meshes together by assigning them the same coordinates in the float array + the same group id number. Turrets/generators etc are set to coordinates "0,0,0" because they can be blown up.In that case the mesh center is used(Could be the Turrent block coordinate too but the mesh hit zone center makes more sense).

IE: Two seperate main hull meshes could be set to Id number 2 and both sets of floats are set to the same X,Y,Z coordinate of where you want the targeting box to show up at in the HUD.

conversly you could have each Mesh show up as as seperate target by giving ea a different id and set of coordinates.

The Ids start from 0 to ~.

Rotation Scale Block Header

This block describes various aspects about which way the mesh is oriented. These are not fully tested yet. the last three sets of floats are vectors. Most notably used in turrets to describe which is the plane of rotation and aiming.

```
Rotation/Scale Block Header
 -----------
 0
 23
 0
 0
 1
 jump to floats
 array[0..2] of float    //  mesh center
 array[3..5] of float    //  axis of rotation   (z)
 array[6..8] of float    //  aim direction      (x)
 array[9..11]of float    //  90 deg to aim direction
```

HardPoint Block Header

Block 22 describes the location of the hard points on ships ,ie the end of the gun barrels... Ships such as a platform have two sets sometimes of hardpoints in the same location, one set is empire weapons the other set is rebel weapons.

```
HardPoint Block Header
 ----------
 0
 22
 0
 0
 1
 jump to floats
 Hardpoint type         //  hardpoint type ie: Rebellaser
 3 Floats              // Hardpoint coordinate
```

Here is a list of the hardpoints.

39. 0: NONE
40. 1: REBELLASER
41. 2: TURBOREBELLASER
42. 3: EMPIRELASER
43. 4: TURBOEMPIRELASER
44. 5: IONCANNON
45. 6: TURBOIONCANNON
46. 7: TORPEDO
47. 8: MISSILE
48. 9: SUPERREBELLASER
49. 10: SUPEREMPIRELASER
50. 11: SUPERIONCANNON
51. 12: SUPERTORPEDO
52. 13: SUPERMISSILE
53. 14: DUMBBOMB
54. 15: FIREDBOMB
55. 16: MAGPULSE
56. 17: TURBOMAGPULSE
57. 18: SUPERMAGPULSE
58. 19: NEWWEAPON1
59. 20: NEWWEAPON2
60. 21: NEWWEAPON3
61. 22: NEWWEAPON4
62. 23: NEWWEAPON5
63. 24: NEWWEAPON6
64. 25: INSIDEHANGAR
65. 26: OUTSIDEHANGAR
66. 27: DOCKFROMBIG
67. 28: DOCKFROMSMALL
68. 29: DOCKTOBIG

69. 30: DOCKTOSMALL
70. 31: COCKPIT

Mesh Info Header

Mesh Info Headers contain

```
Main Info Header 004 block XWVT
---------
 0
 0
 4
 jump to jumps
 1
 Reverse Offset        // Subtract the mesh reverse num from block 3 to
                       // arrive at how many bytes to get back there
 Jump to 3 block       // Mesh Vertices
 Jump to 13 block      // Texture Vertices
 Jump to 11 block      // Vertex Normals
 Jump to 21 block      // Jump to Face Jump
```

Bop Mesh Info Header "A"

```
Main Info Header 004 block BOP "A"
---------
 0
 0
 4
 jump to jumps
 1
 Reverse Offset        // Subtract the mesh reverse num from block 3 to
                       // arrive at how many bytes to get back there
 0                     // Nulled vertex jump
 0                     // Null Texture vertex jump
 0                     // Null Vertex normal jump
 Jump to 21 block      // Jump to Face Jump
```

BOP may also have this section with no "Nulls" as below.
Bop Mesh Info Header "B"

```
Main Info Header 001 block BOP "B"
----------
 0
 0
 1
 jump to jumps
 1
 Reverse Offset        // Subtract the mesh reverse num from block 3 to
                       // arrive at how many bytes to get back there
  Jump to 21 block      // Jump to Face Jump
```

Mesh Vertex Header

Mesh Vertex Headers contain the float type data for the vertices. This header starts with "0, 3" do not confuse it with a Main Mesh Header which can be "0, 0, 3"

```
03 Vertex Header
------------------
 0
 3
 0
 0
 NumVertices        // (Longint)
 Jump to Floats
 Vertices          // NumVertices*3 Floats (x,y,z)
```

Texture Vertex Header

Texture Vertex Headers contain the float type data for alligning textures on faces. these are normalised u,v or s,t

```
13 Texture Vertex Header
 -------------------
 0
 13
 0
 0
 NumTexVertices       // (Longint)
 Jump to Floats
 TexVertices         // NumTexVertices*2 Floats (u,v)
```

Vertex Normal Header

Vertex Normal Headers contain the float type data describing the direction the vertex faces. This is found by averaging the normals of all the faces that use this vertex.

```
11 Vertex Normal Header
-----------------
0
11
0
0
NumVertNorms            // (Longint)
Jump to Floats
VertNorms               // NumVertNorms*3 Floats (i,j,k)
```

Mesh Data Header

This is a directory for master face groups. Follow the MFH Jumps to get to Face Headers (Face Headers then have their own groupings of faces). A small float is included.

```
Mesh Data Header
------------------
0
21
1                                   // Number of Face Data Header Jumps
Jump to the MFH Block Jump
1                                   // Number of jumps to face data headers
Jump to Float
Jump to Mesh Face Header            // High Res Face Data Header
Jump to Mesh Face Header            // Low Res Face Data Header if present see Notes**
Float[array]                        // One float for ea Face Data Header jump (Lo res distance numbers)
```

NOTES on Model Resolutions

If the Number of Face Data Headers is >1 then ,each sucsessive Face data header represents a lower deatil model.

Ie: an Xwing has 3 detail models, the highest would be FDH 0, next highest detail would be FDH 1 , and finaly the lowest detail model is FDH 2.

Mesh Face Header

This Block jumps to various areas related to the geometry and appearance of faces. Various "face groups" can be specified so different textures will be used.

```
Mesh Face Header
 ------------------
0
0
NumFaceGroups          // Num of 20/00/01/07/24 blocks to read
Offset to Jumps
1
Reverse Offset?      // Not sure because it's kinda big
Jump to 20 Block     // Texturing Block ** Can also be a texture "B" block
Jump to 00 block     // Unknown texturing block
Jump to 01 Block     // Face Data
Jump to 07 Block     // texture name redeclared for reuse on another face.
Jump to 24 block     // Texture Jump list, Likely for Flight Group Textures
```

*** Does not always have block 20/07/24 to read only 01 block if num is 1.

*** 20 block new texture.
*** 24 Block New Group of textures (always 4)? Flight Group colors
*** 07 Block Re use texture on this face
*** 01 Face Data

20 Texture Header

Includes Texture and MipMaps. MipMaps are a series of images that decrease in dimensions by a factor of 2. Three MipMaps are included along with the main image. ie. A Texture would include an image that is 64x64, 32x32, 16x16, and 8x8.

Texture data is stored like a bmp (upside down to convention) . Ie bottom right pixel of image is first byte and top right pixel last byte ,unlike a bmp there is no padding of the bytes.

Texture headers have been found in several variations.

```
Texture Header "A"
 -------------------
JumpToTex              // Points the 'T' in "Tex00000"
20
0
```

```
0
1
Jump To "Palette offset " jump
Byte[0..8]              // "Tex00000" + '\0' (null term string (9-bytes))
Jump to "Palettes"      //  beginning of CDCD CDCD Block (Palettes)
0
TexSize                 // Size without MipMaps (Width*Height)
DataSize                // Size including MipMap Sizes
Widthbs                 // Width of Image
Height                  // Height of Image
Byte[0..DataSize-1]     // Image Data + MipMap Data
```

Texture Header "AA"

```
0               // BOMB.OPT ONLY ,not decoded yet..
20
0
0
```

Texture Header "B"

```
0
7
0
0
1
JumpToTex               // Points the 'T' in "Tex00000"
Byte[0..8]              // "Tex00000" + '\0' (null term string (9-bytes))
```

Texture Header "C"

```
0
24
NumOfJumps              // Number of (20) offsets to read
JumpListOffset          // Offset to Jump List
1
Reverse Offset
Array Of Offsets        // these offsets point to the first entry in a Texture Header"A" Block
```

Texture Header "D"

```
----------------
0
0
0
0
1
JumpToTex              // offsett into some unrelated texture data ,appears to be unused old code
```

Palettes

Palettes are found by using the Jump to palletes offset in the 20 texture header block. So Far the pallete data is seperated by a block of 4096 bytes of $CD(hex). Then the palette data starts. 16 pals ,ea uses 256 X 2 Bytes for a total of 512 bytes for ea palette/colormap. Grand total 8192 bytes.

Textures share a set of 16 palettes , if the "Jump to "Palettes" entry is the same in the Texture header "A" then these textures share that set of 16 palettes. The Pals go from dark to lighter shades with the middle pal being about right for viewing most of the time. It should be possible to make a set of color maps from the number 8 palette. upon quick analyisis it looks like the colormap values are always offset by a given amount from the main palette values.

Looking at the code with Texxtract thought fully provided by it`s author.It seems the current format people were using was a 5-4-3 blue,green,red bit pattern. After trying this process out ,it seemed to me the colors were off somewhat still. I have found a bit pattern of 5-6-5 b,g,r works much better. This is one of the bit patterns a win 95 16 bit bitmap can use, which makes a bit more sense IMO. The number 8 palette seems to be the most compatable.(assuming the first palette/colormap is number 1)

Unkown if there is a limit on the number of pals you can add, though file size would have to be a coinsideration if no pals were shared among textures.

```
Palette General
------------------
$CD                    // 4096 bytes
256 X 2 bytes          // ea palette X 16 = 8192 bytes
                       // pals use 5-6-5  blue,green,red bit pattern
```

Face Data Header

This block contains information for the faces in the mesh that called it.

```
Face Data Header
------------------
 0
 1
 0
```

```
0
NumFaces                    // Number of Faces being declared
Offset to Longint           // Face data immediately follows the longint
Edges                       // unique edge count
FaceVerts[0..NumFaces-1]    // See below for structure
FaceNorms[0..NumFaces-1]    // Face Normals, see below
FaceText[0..NumFaces-1]     // Face Texturing Info, see below
```

Vertices: Each face declares four groups of four vertex references (longints). The first group is the references to the geometry vertices (3 Block).

Edges The second group is edges numbers.Ea edge has a unique number in this group of faces.Ea pair of vertices that share an edge will have the same number.

TX Coordinates The third group is references to the texture vertice list.

Vertex Normals The fourth group is referenced to the vertex normal list . All references are Longints. If a reference is FFFFFFFF (-1), then it should be ignored, and that face should be assumed to have only 3 vertices. Each face will take 64 bytes. Faces are either a triangle or a quad.

Normals: Each face has three floats (i,j,k) representing its normalized vector. Each face then has 12 Bytes.

Texturing: Used by the software rendering engine. Six floats for each face (24 bytes). These are 2 vectors.Direction and magnitude are significant.
1. the cross product of the two must be the face normal (not unit length, but same direction as the face normal) or the texture exhibits weird "out-of-plane" effects.
2. The vectors are set to the "across-top" dx,dy,dz and the "down side" dx,dy,dz. In the case of a rectangular face the vector magnitudes (lengths) will equal the side lengths. Which side the face is viewed from in calculating this doesn't seem to matter.
3. The two vectors position the texture in 3-D space and also set the scale of the texture - the texture is basically stretched in space to fit the vectors.