WIKIPEDIA

Wavefront .obj file

OBJ (or .OBJ) is a geometry definition file format first developed by Wavefront Technologies for its Advanced Visualizer animation package. The file format is open and has been adopted by other 3D graphics application vendors.

The OBJ file format is a simple data-format that represents 3D geometry alone — namely, the position of each vertex, the UV position of each texture coordinate vertex, vertex normals, and the faces that make each polygon defined as a list of vertices, and texture vertices. Vertices are stored in a counter-clockwise order by default, making explicit declaration of face normals unnecessary. OBJ coordinates have no units, but OBJ files can contain scale information in a human readable comment line.

OBJ geometry format

Filename extension	`.obj`
Internet media type	`text/plain`
Developed by	Wavefront Technologies
Type of format	3D model format

Contents

File format

Material template library

File format

Lines beginning with a hash character (#) are comments.

```
# this is a comment
```

An OBJ file may contain vertex data, free-form curve/surface attributes, elements, free-form curve/surface body statements, connectivity between free-form surfaces, grouping and display/render attribute information. The most common elements are geometric vertices, texture coordinates, vertex normals and polygonal faces:

```
  # List of geometric vertices, with (x, y, z [,w]) coordinates, w is optional and defaults to 1.0.
  v 0.123 0.234 0.345 1.0
  v ...
  ...
  # List of texture coordinates, in (u, [,v ,w]) coordinates, these will vary between 0 and 1. v, w are optional
and default to 0.
  vt 0.500 1 [0]
  vt ...
  ...
  # List of vertex normals in (x,y,z) form; normals might not be unit vectors.
  vn 0.707 0.000 0.707
  vn ...
  ...
  # Parameter space vertices in ( u [,v] [,w] ) form; free form geometry statement ( see below )
  vp 0.310000 3.210000 2.100000
  vp ...
  ...
  # Polygonal face element (see below)
  f 1 2 3
  f 3/1 4/2 5/3
  f 6/4/1 3/5/3 7/6/5
  f 7//1 8//2 9//3
  f ...
  ...
  # Line element (see below)
  l 5 8 1 2 4 9
```

Geometric vertex

A vertex can be specified in a line starting with the letter `v`. That is followed by (x,y,z[,w]) coordinates.

W is optional and defaults to 1.0. Some applications support vertex colors, by putting red, green and blue values after x y and z. The color values range from 0 to 1.[1]

Parameter space vertices

A free-form geometry statement can be specified in a line starting with the string `vp`. Define points in parameter space of a curve or surface. `u` only is required for curve points, `u` and `v` for surface points and control points of non-rational trimming curves, and `u`, `v` and `w` (weight) for control points of rational trimming curves.

Face elements

Faces are defined using lists of vertex, texture and normal indices which start at 1. Polygons such as quadrilaterals can be defined by using more than three vertex/texture/normal indices.

OBJ files also support free-form geometry which use curves and surfaces to define objects, such as NURBS surfaces.

Vertex indices

A valid vertex index matches the corresponding vertex elements of a previously defined vertex list. If an index is positive then it refers to the offset in that vertex list, starting at 1. If an index is negative then it relatively refers to the end of the vertex list, -1 referring to the last element.

Each face can contain three or more vertices.

```
f v1 v2 v3 ....
```

Vertex texture coordinate indices

Optionally, texture coordinate indices can be used to specify texture coordinates when defining a face. To add a texture coordinate index to a vertex index when defining a face, one must put a slash immediately after the vertex index and then put the texture coordinate index. No spaces are permitted before or after the slash. A valid texture coordinate index starts from 1 and matches the corresponding element in the previously defined list of texture coordinates. Each face can contain three or more elements.

```
f v1/vt1 v2/vt2 v3/vt3 ...
```

Vertex normal indices

Optionally, normal indices can be used to specify normal vectors for vertices when defining a face.

To add a normal index to a vertex index when defining a face, one must put a second slash after the texture coordinate index and then put the normal index. A valid normal index starts from 1 and matches the corresponding element in the previously defined list of normals. Each face can contain three or more elements.

```
f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3 ...
```

Vertex normal indices without texture coordinate indices

As texture coordinates are optional, one can define geometry without them, but one must put two slashes after the vertex index before putting the normal index.

```
f v1//vn1 v2//vn2 v3//vn3 ...
```

Line elements

Records starting with the letter "l" specify the order of the vertices which build a polyline.

```
l v1 v2 v3 v4 v5 v6 ...
```

Other geometry formats

Obj files support higher-order surfaces using several different kinds of interpolation, such as Taylor and B-splines, although support for those features in third party file readers is far from universal. Obj files also do not support mesh hierarchies or any kind of animation or deformation, such as vertex skinning or mesh morphing.

Referencing materials

Materials that describe the visual aspects of the polygons are stored in external .mtl files. More than one external MTL material file may be referenced from within the OBJ file. The .mtl file may contain one or more named material definitions.

```
mtllib [external .mtl file name]
  ...
```

This tag specifies the material name for the element following it. The material name matches a named material definition in an external .mtl file.

```
usemtl [material name]
  ...
```

Named objects and polygon groups are specified via the following tags.

```
o [object name]
  ...
  g [group name]
  ...
```

Smooth shading across polygons is enabled by smoothing groups.

```
s 1
  ...
  # Smooth shading can be disabled as well.
  s off
  ...
```

Relative and absolute indices

OBJ files, due to their list structure, are able to reference vertices, normals, etc. either by their absolute position (1 represents the first defined vertex, N representing the Nth defined vertex), or by their relative position (-1 represents the latest defined vertex). However, not all software supports the latter approach, and conversely some software inherently writes only the latter form (due to the convenience of appending elements without needing to recalculate vertex offsets, etc.), leading to occasional incompatibilities.

Material template library

MTL material format

Filename extension	`.mtl`
Developed by	Wavefront Technologies
Type of format	3D texture format

The **Material Template Library** format (MTL) or .MTL File Format is a companion file format to .OBJ, also defined by Wavefront Technologies, that describes surface shading (material) properties of objects within one or more .OBJ files. A .OBJ file references one or more .MTL files (called "material libraries"), and from there, references one or more material descriptions by name. .MTL files are ASCII text that define the light reflecting properties of a surface for the purposes of computer rendering, and according to the Phong reflection model. The standard has widespread support among different computer software packages, making it a useful format for interchange of materials.

The MTL format, although still widely used, is outdated and does not fully support later technologies such as specular maps and parallax maps. However, due to the open and intuitive nature of the format, these can easily be added with a custom MTL file generator.

The MTL format defines a number of formats.[2][3]

Basic materials

A single .mtl file may define multiple materials. Materials are defined one after another in the file, each starting with the newmtl command:

```
# define a material named 'Colored'
   newmtl Colored
```

The ambient color of the material is declared using Ka. Color definitions are in RGB where each channel's value is between 0 and 1.

```
Ka 1.000 1.000 1.000     # white
```

Similarly, the diffuse color is declared using Kd.

```
Kd 1.000 1.000 1.000     # white
```

The specular color is declared using Ks, and weighted using the specular exponent Ns.

```
Ks 0.000 0.000 0.000     # black (off)
Ns 10.000                # ranges between 0 and 1000
```

Materials can be transparent. This is referred to as being *dissolved*. Unlike real transparency, the result does not depend upon the thickness of the object. A value of 1.0 for "d" is the default and means fully opaque, as does a value of 0.0 for "Tr".

```
d 0.9                    # some implementations use 'd'
Tr 0.1                   # others use 'Tr' (inverted: Tr = 1 - d)
```

Multiple illumination models are available, per material. These are enumerated as follows:

```
0. Color on and Ambient off
1. Color on and Ambient on
2. Highlight on
3. Reflection on and Ray trace on
4. Transparency: Glass on, Reflection: Ray trace on
5. Reflection: Fresnel on and Ray trace on
6. Transparency: Refraction on, Reflection: Fresnel off and Ray trace on
7. Transparency: Refraction on, Reflection: Fresnel on and Ray trace on
8. Reflection on and Ray trace off
9. Transparency: Glass on, Reflection: Ray trace off
10. Casts shadows onto invisible surfaces
```

```
   illum 2
```

Texture maps

Textured materials use the same properties as above, and additionally define texture maps. Below is an example of a common material file. See the full wavefront file format reference for more details.

```
newmtl Textured
Ka 1.000 1.000 1.000
Kd 1.000 1.000 1.000
Ks 0.000 0.000 0.000
d 1.0
illum 2
map_Ka lemur.tga           # the ambient texture map
map_Kd lemur.tga           # the diffuse texture map (most of the time, it will
                           # be the same as the ambient texture map)
map_Ks lemur.tga           # specular color texture map
map_Ns lemur_spec.tga      # specular highlight component
map_d lemur_alpha.tga      # the alpha texture map
map_bump lemur_bump.tga    # some implementations use 'map_bump' instead of 'bump' below
```

```
bump lemur_bump.tga        # bump map (which by default uses luminance channel of the image)
disp lemur_disp.tga        # displacement map
decal lemur_stencil.tga    # stencil decal texture (defaults to 'matte' channel of the image)
```

Texture map statements may also have option parameters (see full spec (http://paulbourke.net/dataformats/mtl/)).

```
map_Ka -o 1 1 1 ambient.tga            # texture origin (1,1,1)
refl -type sphere clouds.tga           # spherical reflection map
```

Texture options

```
 -blendu on | off                       # set horizontal texture blending (default on)
 -blendv on | off                       # set vertical texture blending (default on)
 -boost float_value                     # boost mip-map sharpness
 -mm base_value gain_value              # modify texture map values (default 0 1)
                                        #     base_value = brightness, gain_value = contrast
 -o u [v [w]]                           # Origin offset             (default 0 0 0)
 -s u [v [w]]                           # Scale                     (default 1 1 1)
 -t u [v [w]]                           # Turbulence                (default 0 0 0)
 -texres resolution                     # texture resolution to create
 -clamp on | off                        # only render texels in the clamped 0-1 range (default off)
                                        #   When unclamped, textures are repeated across a surface,
                                        #   when clamped, only texels which fall within the 0-1
                                        #   range are rendered.
 -bm mult_value                         # bump multiplier (for bump maps only)

 -imfchan r | g | b | m | l | z         # specifies which channel of the file is used to
                                        # create a scalar or bump texture. r:red, g:green,
                                        # b:blue, m:matte, l:luminance, z:z-depth..
                                        # (the default for bump is 'l' and for decal is 'm')
```

```
    bump -imfchan r bumpmap.tga            # says to use the red channel of bumpmap.tga as the bumpmap
```

For reflection maps...

```
    -type sphere                           # specifies a sphere for a "refl" reflection map
    -type cube_top    | cube_bottom |      # when using a cube map, the texture file for each
          cube_front  | cube_back   |      # side of the cube is specified separately
          cube_left   | cube_right
```

Vendor specific alterations

Because of the ease in parsing the files, and the unofficial spreading of the file format, files may contain vendor specific alterations.

According to the spec, options are supposed to precede the texture filename. However, at least one vendor generates files with options at the end.

```
bump texbump.tga -bm 0.2               # bump multiplier of 0.2
```

Physically-based Rendering

The creators of the online 3D editing and modeling tool, Clara.io, proposed extending the MTL format to contain the following parameters to represent the physically-based rendering parameters:[4]

```
Pr/map_Pr     # roughness
Pm/map_Pm     # metallic
Ps/map_Ps     # sheen
Pc            # clearcoat thickness
Pcr           # clearcoat roughness
Ke/map_Ke     # emissive
aniso         # anisotropy
anisor        # anisotropy rotation
norm          # normal map, same format as "bump" parameter
```

Further proposed extensions come from the DirectXMesh toolkit for Microsoft's DirectX engine, allowing the ability to define a model's pre-compiled RMA material.[5]

```
map_RMA       # RMA material (roughness, metalness, ambient occlusion)
map_ORM       # alternate definition of map_RMA
```

See also

- OFF (file format)

- STL (file format)
- PLY (file format) is an alternative file format offering more flexibility than most stereolithography applications.

References

1. "How can I include vertex color information in .OBJ files?" (https://gamedev.stackexchange.com/questions/21303/how-can-i-include-vertex-color-information-in-obj-files/66270#66270). *Game Development Stack Exchange*. Retrieved 2014-10-08.
2. "MTL Files - Material Definitions for OBJ Files" (http://people.sc.fsu.edu/~burkardt/data/mtl/mtl.html). People.sc.fsu.edu. 2004-06-14. Retrieved 2010-11-26.
3. Author. "Wavefront .mtl file format info - GRIPES and GRUMBLES - Wings - Wings3D - Official Development Forum - Message Board" (http://nendowingsmirai.yuku.com/forum/viewtopic/id/1723) . Nendowingsmirai.yuku.com. Retrieved 2010-11-26.
4. "Exocortex | Extending Wavefront MTL for Physically-Based Rendering" (http://exocortex.com/blog/extending_wavefront_mtl_to_support_pbr). *exocortex.com*.
5. "Ability to define RMA texture in OBJ's MTL. by MattFiler · Pull Request #39 · microsoft/DirectXMesh" (https://github.com/microsoft/DirectXMesh/pull/39). *GitHub*.

External links

- Appendix B1. Object Files (.obj), Advanced Visualizer Manual (http://www.cs.utah.edu/~boulos/cs3505/obj_spec.pdf)
- Obj Specification as used by Wavefront (http://www.martinreddy.net/gfx/3d/OBJ.spec)
- Mtl Specification (http://paulbourke.net/dataformats/mtl/)
- Tools, libraries and example files (http://people.sc.fsu.edu/~jburkardt/data/obj/obj.html)
- FileFormat.Info: Wavefront OBJ File Format Summary (https://www.fileformat.info/format/wavefrontobj/egff.htm)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Wavefront_.obj_file&oldid=923366717"

This page was last edited on 28 October 2019, at 02:38 (UTC).