

NAME

obj2mesh - create a compiled RADIANCE mesh file from Wavefront .OBJ input

SYNOPSIS

obj2mesh [**-a** *matfile*] [**-l** *matlib*] [**-n** *objlim*] [**-r** *maxres*] [**-w**] [**-v**] [*input.obj* [*output.rtm*]]

DESCRIPTION

Obj2mesh reads a Wavefront .OBJ file from *input.obj* (or the standard input) and compiles it into a RADIANCE triangle mesh, which is sent to *output.rtm* (or standard output). Any RADIANCE material descriptions included via one or more **-a** options will be compiled and stored in the mesh as well. If the **-l** option is used to specify a material file, the RADIANCE library locations are searched. This mesh may be included in a RADIANCE scene description via the *mesh* primitive, thus:

```
mod mesh id
  1+ output.rtm [xform args]
  0
  0
```

The syntax and semantics are identical to the RADIANCE *instance* primitive. If *mod* is "void", then the stored mesh materials will be applied during rendering. Otherwise, the given material will be substituted on all the mesh surfaces.

The **-n** option specifies the maximum surface set size for each voxel. Larger numbers result in quicker mesh generation, but potentially slower rendering. Values below 6 are not recommended, since this is the median valence for a mesh vertex (the number of adjacent faces), and smaller values will result in pointless octree subdivision. The default setting is 9.

The **-r** option specifies the maximum octree resolution. This should be greater than or equal to the ratio of the mesh bounding box to the smallest triangle. The default is 16384.

The **-w** option suppresses warnings. The **-v** option prints out final mesh statistics.

Although the mesh file format is binary, it is meant to be portable between machines. The only limitation is that machines with radically different integer sizes will not work together.

DETAILS

The following Wavefront statements are understood and compiled by *obj2mesh*.

v *x y z* A vertex location, given by its Cartesian coordinates. The final mesh position may of course be modified by the transform arguments given to the *mesh* primitive in the Radiance scene description.

vn *dx dy dz* A vertex normal vector, given by its three direction components, which will be normalized by *obj2mesh*. Normals will be interpolated over the mesh during rendering to produce a smooth surface. If no vertex normals are present, the mesh will appear tessellated. A zero length normal (i.e., 0 0 0) will generate a syntax error.

vt *u v* A local vertex texture coordinate. These coordinates will be interpolated and passed to the "Lu" and "Lv" variables during rendering. Local coordinates can extend over any desired range of values.

usemtl *mname* A material name. The following faces will use the named material, which is taken from the material definitions in the **-a** input file(s).

g *gname* Group association. The following faces are associated with the named group. If no "usemtl" statement has been encountered, the current group is used for the surface material identifier.

f *v1/t1/n1 v2/t2/n2 v3/t3/n3 ..* A polygonal face. Polygon vertices are specified as three indices separated by slashes ('/'). The first index is the vertex location, the second index is the local (u,v) texture coordinate, and the third index is the vertex surface normal. Positive indices count from the beginning of the

input, where the first vertex position (*v* statement) is numbered 1, and likewise for the first texture coordinate and the first surface normal. Negative indices count backward from the current position in the input, where -1 is the last vertex encountered, -2 is the one before that, etc. An index of 0 may be used for the vertex texture or normal to indicate none, or these may be left off entirely. All faces will be broken into triangles in the final mesh.

All other statement types will be ignored on the input. Statements understood by *obj2rad(1)* will be ignored silently; other statements will generate a warning message after translation to indicate how much was missed.

DIAGNOSTICS

There are four basic error types reported by *obj2mesh*:

- warning - a non-fatal input-related error
- fatal - an unrecoverable input-related error
- system - a system-related error
- internal - a fatal error related to program limitations
- consistency - a program-caused error

Most errors are self-explanatory. However, the following internal errors should be mentioned:

Set overflow in *addobject* (id)

This error occurs when too many surfaces are close together in a scene. Sometimes a dense mesh can be accommodated by increasing the maximum resolution (by powers of two) using the *-r* option, but usually this error indicates something is wrong. Either too many surfaces are lying right on top of each other, or the bounding cube is inflated from disparate geometry in the input. Chances are, the face number "id" is near those causing the problem.

Hash table overflow in *fullnode*

This error is caused by too many surfaces, and there is little hope of compiling this mesh.

EXAMPLES

To create a compiled triangle mesh from the scene file *mesh.obj* using materials from the file *mesh.mat*:

```
obj2mesh -a mesh.mat mesh.obj mesh.rtm
```

To use local coordinates to place a square tiled image on a mesh object:

```
void colorpict tiled_pat
7 red green blue mytile.hdr . frac(Lu) frac(Lv)
0
0

tiled_pat plastic tiled_mat
0
0
5 .9 .9 .9 0 0

tiled_mat mesh tiled_mesh
1 mymesh.rtm
0
0
```

ENVIRONMENT

RAYPATH the directories to search for material files.

AUTHOR

Greg Ward

OBJ2MESH(1)

OBJ2MESH(1)

SEE ALSO

gensurf(1), getinfo(1), make(1), obj2rad(1), objutil(1) oconv(1), rpict(1), rvu(1), rtrace(1), xform(1)