wrl2smol

November 2008
Steven Andrews

wrl2smol is a very short program that converts triangle mesh data from a wrl file format to a Smoldyn configuration file format.

Following is a sample wrl file for a tetrahedron:

Input file

```
#VRML V2.0 utf8
#
#  tetrahedron.wrl  10 August 1999
#
#  Rikk Carey and Gavin Bell,
#  The Annotated VRML 2.0 Reference Manual,
#  pages 198-199
#

Viewpoint { description "Initial view" position 0 0 9 }

NavigationInfo { type "EXAMINE" }

#
#  Tetrahedron, 4 vertices, 4 faces and a color at each vertex.
#
Transform {
  translation 1.5 -1.5 0.0
  children Shape {
    appearance DEF A Appearance { material Material { } }
    geometry IndexedFaceSet {
      coord Coordinate {
        point [
           1.0  1.0  1.0,
           1.0 -1.0 -1.0,
          -1.0  1.0 -1.0,
          -1.0 -1.0  1.0,
        ]
      }
      coordIndex [
        3, 2, 1, -1,
        2, 3, 0, -1,
        1, 0, 3, -1,
        0, 1, 2, -1,
      ]
      color Color {
        color [
          0.0 1.0 0.0,
          1.0 1.0 1.0,
          0.0 0.0 1.0,
          1.0 0.0 0.0
        ]
      }
      colorPerVertex TRUE
    }
  }
}
```

The only parts of this file that wrl2smol reads are: the one that starts with "`point`", the following list of points, the one that starts with "coordIndex", and the following lists of coordinate indicies. Points must be space-separated, although comma-terminated is fine. Coordinate indicies may be comma and space-separated, as shown above, or may be just space-separated. Lines of text above the "`point`" word, between the close bracket after the last point and above the "`coordIndex`" word, and after the close bracket after the last coordinate index are all ignored.

wrl2smol asks the user a few questions (all of which have to be answered in response to the queries, and cannot be included on the command line).

"`Enter name of input wrl file or 'done' if finished:`" Enter the name of the input file, with path information if appropriate. If you want to concatanate multiple wrl files, each of which describes part of the surface, then enter subsequent file names when this question is asked again. When you are done listing the files to be concatanated together, enter "done".

"`Join any adjacent but currently disjoint triangles (y/n):`" If two triangles share corner points that have identical values but are listed separately, then they are not registered as being neighboring triangles. This can lead to surfaces that are physically continuous but logically disjoint. This situation can occur with either a single or multiple wrl files. Enter 'y' for wrl2smol to address it and make the entire surface into a single contiguous region, or 'n' for wrl2smol to leave the regions as the wrl file stored them.

"`Align orientations of neighboring triangles (y/n):`" Typically, it is best for all of the triangles that compose a surface to be aligned, such that all front sides point inward, or all front sides point outward. However, some wrl files are not generated this way, so wrl2smol allows the option of automatically flipping triangles as required to achieve this alignment. Enter 'y' for wrl2smol to align the triangles and 'n' if they should be left as they were.

"`flip triangle 0:` (x_0 y_0 z_0) (x_1 y_1 z_1) (x_2 y_2 z_2)?" If you requested aligned triangles in the previous question, then wrl2smol will ask whether the first triangle is correct as it is, or if it should be flipped; it presents the three triangle vertices, so you can see if it is correct or not. Orientations use the right-hand rule: if the fingers of the right hand curl in the direction from point 0 to point 1 to point 2, then the thumb points towards the front side of the triangle. If the input wrl file describes disjoint sets of triangles (and they were not corrected with the joining operation), then it wrl2smol will ask this question once for each disjoint set.

“`Enter name for Smoldyn output file:`” Enter the name for the output file, including a suffix if desired (my standard is .txt). Note that wrl2smol does not check that this output file does not overwriting any pre-existing files, including the input file.

“`Enter triangle name and starting number (e.g. tri 1):`” Panels in Smoldyn are named, so wrl2smol needs to know what the panel name and starting number should be. For example, a response of “tri 1” would yield panel names tri1, tri2, tri3, etc.

“`Print (1) no neighbors, (2) edge neighbors, (3) all neighbors:`” This neighbor printout lists neighboring triangles with the Smoldyn `neighbor` statment, so that surface-bound molecules can diffuse from between neighboring panels. If there are no diffusing surface-bound molecules in the model, then specify that no neighbors should be printed. Edge neighbors are those that share complete edges, and “all” neighbors includes all panel pairs that share at least one vertex. If a simulation uses diffusing surface-bound molecules, then it’s most accurate to list all neighbors, although some speed-up can probably be achieved with just edge neighbors.

Running wrl2smol on the above tetrahedron wrl file, with all neighbors listed, results in the following output:

wrl2smol output file

```
# Smoldyn surface data file automatically generated by wrl2smol
# input file: tetrahedron.wrl, output file: tetrahedron.txt

max_panels tri 4

panel tri  -1 -1 1  -1 1 -1  1 -1 -1  tet1
panel tri  -1 1 -1  -1 -1 1  1 1 1  tet2
panel tri  1 -1 -1  1 1 1  -1 -1 1  tet3
panel tri  1 1 1  1 -1 -1  -1 1 -1  tet4
neighbor tet1 tet2 tet3 tet4
neighbor tet2 tet1 tet3 tet4
neighbor tet3 tet1 tet2 tet4
neighbor tet4 tet1 tet2 tet3

end_file
```

This file can be read by Smoldyn. It needs to be called from another configuration file using a `read_file` command. Note that Smoldyn does not allow multiple `max_panels` statements for the same panel shape and the same surface. If this causes a problem, then simply put the `max_panels` statement in the configuration file that lists the basic surface definitions, and comment out `max_panels` statements in wrl2smol output files.

Code documentation

Include files: only standard library files

Functions:

```
double **expandpoints(int nnew,int nold,double **points);
```

Allocates memory for points. If this is the first time it is called, send in `nnew` with the number of points to allocate and points as `NULL`. If it has been called before, send in `nnew` with the new number that are desired, `nold` with the previous allocated size, and `points` with the previous list. This allocates and initiallizes `nnew` points, copies old ones into the new list if appropriate, and frees the old list if one was entered. It returns the list of new points. If it cannot allocate sufficient memory, it frees any old points and returns `NULL`.

```
void pointsfree(int npts,double **points);
```

Frees a list of `points`, which was allocated to size `npts`.

```
int **expandcoords(int nnew,int nold,int **coords);
```

Identical to `expandpoints`, but for coordinate indicies rather than points.

```
void coordsfree(int ncds,int **coords);
```

Frees a lits of coordinates, which was allocated to size `ncds`.

```
int fliptriangles(int **coords,int *aligndone,int ncds,int *nflipptr);
```

Flips triangles as needed to align them. `coords` is a list of triangle coordinate indicies, `aligndone` lists whether alignment has been done for each triangle or not, `ncds` is the number of coordinates (triangles), and `nflipptr`, if it is not sent in as `NULL`, is returned pointing to the number of triangles that got flipped during this function call. This returns the number of triangles that were aligned during the function call, regardless of whether they required flipping or not.

This function goes through all triangles sequentially. For each one that has not been aligned yet, it looks to see if it has a neighbor that has already been aligned. It aligns it with the neighbor if so, and otherwise just leaves it alone. Because this function does not go back to align the triangles that it passed over previously, it needs to be called multiple times, until it returns 0. If it returns 0, this means that any triangles that remain unaligned cannot be aligned without further information.

```
int main(void);
```

Essentially the entire program is here. This gets information from the user, reads the list of points, reads the list of coordinate indicies, aligns triangles if needed, outputs the triangle data, outputs the neighbor data, and exits.