Stefan Johansson
stefanj@cs.umu.se

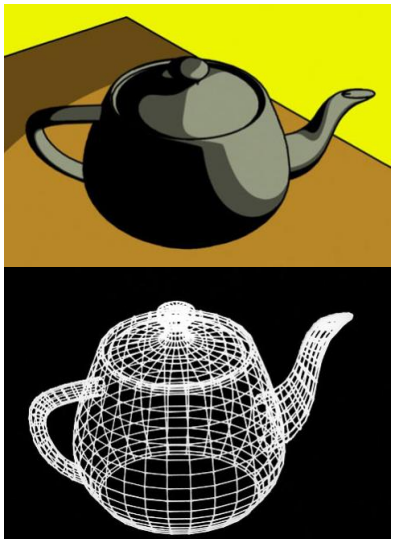
UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE

Exercise

Read Wavefront OBJ files

Please, read the introduction for the project first (found under 'Files/Project Resources' on Canvas). This exercise is part of the project and nothing should be handed in.

Storing Polygonal Data



A 3D model that is to be visualized is usually represented as a polygon mesh. This is a collection of edges, vertices, and polygons connected such that each edge is shared by at least one but at most two polygons. In normal rendering only polygons are important, but in some situations (e.g., when doing sub-division or drawing with wire-frame or cartoon shading) edges are also important. For our purpose, speed is of importance, i.e. we must optimize our data representation for fast access.

Polygon meshes can be represented in many different ways and are evaluated according to space and time. More complex representations such as the Half-edge or Wing-edge data representation also exist, but below we look at three basic variants where the second one is the one we recommend to use in the course.

Explicit Representation

Each polygon is represented by a list of vertex coordinates.

$P = ((x_1, y_1, z_1), (x_2, y_2, z_2), \ldots , (x_n, y_n, z_n))$

Advantages

- Easy to implement.

Disadvantages

- Large storage space. For more than one polygon, space is wasted because vertices are duplicated. It is also risky to duplicate data when the data may change.
- Takes time since there is no explicit representation of edges and vertices, an interactive move of a vertex involves finding all polygons that share the vertex.
- Polygon edges are drawn twice since there is no easy way to find edges.

Stefan Johansson
stefanj@cs.umu.se

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE

Pointers to a Vertex List (recommended)

Each vertex V_i is stored once in a vertex list, and a polygon, P_j, is a list of indices to a vertex list.

$V = (V_1, V_2, V_3, V_4) = ((x_1, y_1, z_1), \dots , (x_4 , y_4 , z_4))$

$P_1 = (1, 2, 4)$ (the polygon consists of the vertices V_1, V_2, and V_4.)
$P_2 = (4, 2, 3)$

Advantages

- Space saved since each vertex is stored once.
- Coordinates of a vertex can be changed easily.

Disadvantage

- Difficult to find polygons that share edges.
- Polygon edges are drawn twice since there is no easy way to find edges.

Pointers to an Edge List

Each polygon is represented by a pointer to the edge list

$V = (V_1, V_2, V_3, V_4) = ((x_1, y_1, z_1), \dots , (x_4 , y_4 , z_4))$

$E_1 = (V_1, V_2, P_1, \lambda)$
$E_2 = (V_2, V_3, P_2, \lambda)$
$E_3 = (V_3, V_4, P_2, \lambda)$
$E_4 = (V_4, V_2, P_1, P_2)$
$E_5 = (V_4, V_1, P_1, \lambda)$

$P_1 = (E_1, E_4, E_5)$
$P_2 = (E_2, E_3, E_4)$

Advantages

- Displays edges rather than polygons.
- Eliminates redundant clipping, transformation, and scan conversion.
- Filled polygons are more easily clipped.

Disadvantage

- The order of vertices may be important in drawing, i.e. the vertices in all polygons should be in clock-wise or counter clock-wise order. This is tricky in this representation.

In all three cases, determining which edges are incident to a vertex is not easy. All edges must be inspected. If this is important (optional in part 3 of the project), the mentioned Half-edge representation is probably better.

Wavefront OBJ File Format

As with data representation, many formats exist to store 3D models in files. In this course, we will use the widely available Wavefront's OBJ format. This format is basically pointers

Stefan Johansson
stefanj@cs.umu.se

to a vertex list as described above. In addition, it can also store normals, texture coordinates, parametric curves, surfaces, and in some versions also simple color data. On the other hand it has limited support for representing advanced model properties. The official specification can be found at:

`http://www.martinreddy.net/gfx/3d/OBJ.spec`

See also Appendix B1 of the manual for version 3 of Advanced Visualizer software by Wavefront (a copy can be found at 'Files/Project Resources/OBJ_spec.pdf' on Canvas).

An OBJ file is a text file where each line starts with a tag denoting the element type or a hash (#) for a comment. On this course, you only need to consider polygonal geometries which are defined by the elements for vertices (v), vertex normals (vn), polygonal faces (f), and (optional) texture coordinates (vt). For the other existing element types see the specification. Note that the tags can come in *any* order and even intermixed.

The format of the file looks like:

```
# List of vertices in the format (x,y,z[,w]),
# where w is optional with default value 1.0.
# Optionally a color for the vertex can be
# defined in the format (R,G,B) after the
# vertex data.
v VX[1] VY[1] VZ[1] VW[1]
    :     :     :     :

# With vertex color
v VX[1] VY[1] VZ[1] VW[1] VR[1] VG[1] VB[1]
    :     :     :     :     :     :     :

# List of texture coordinates in the format
# (u,v[,w]), where w is optional with default
# value 0.0
vt VTU[1]  VTV[1]  VTW[1]
      :       :       :

# List of vertex normals in the format (i,j,k)
vn NI[1]  NJ[1]  NK[1]
     :      :      :

# List of polygonal face elements
f F1[1] F2[1] F3[1] ... Fn[1]
    :     :     :       :
```

```
v 0.0 0.0 0.0
v 1.0 0.0 0.0
v 1.0 1.0 0.0
v 0.0 1.0 0.0
v 0.5 0.5 1.6

f 2 1 4
f 2 4 3
f 1 2 5
f 1 5 4
f 4 5 3
f 2 3 5
```

An example of an OBJ file of a pyramid with only vertices and polygonal faces.

Below follows a short description of each element type. For a full description see pages 7–8 and 16–18 in `OBJ_spec.pdf`.

The vertex coordinates (*x,y,z*) are given as floating point numbers. The optional color is specified in red, green, and blue by a triple (R,G,B), where the color values must be in the range [0, 1].

Stefan Johansson
stefanj@cs.umu.se

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE

The texture coordinates are in the range [0, 1]. Depending on the texture v and/or w are optional.

The normals are given as floating point numbers and do not need to be normalized (unit vectors). The normal points in the *direction of the face* and are used instead of the actual face normal. To compute the vertex normal, all face normals for one vertex must be added and normalized. However, in some obj files the normal is the actual vertex normal, in that case the normal indices are different in each face (see below).

The polygonal faces are defined with three or more vertices, where each vertex is referred to with its offset index in the vertex list, starting at 1. A negative index instead starts from the end of the list, i.e. -1 is the last vertex. For each vertex the corresponding texture coordinate index and/or normal index can be specified as well (separated with a slash, `/`). To summarize, the four possible cases are:

1. `f v1 v2 v3 ...` (only vertex indices, see also the example above)
2. `f v1/vt1 v2/vt2 v3/vt3 ...` (vertex and texture coord. indices)
3. `f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3 ...` (vertex, texture coord., and normal indices)
4. `f v1//vn1 v2//vn2 v3//vn3 ...` (vertex and normal indices)

In addition to the object data specified in the OBJ file, material data can be specified in an external MTL material file. The file and the material are referenced in the OBJ file as

mtllib [.mtl file name]

and

usemtl [name of the material to use]

The MTL file is optional to use in the project.

Triangulation

Drawing polygon data is much faster if all polygons have the same number of vertices. Since there can be any number of vertices in a polygon face according to the OBJ format we should fall back to triangles, i.e. each polygon face read from the OBJ file should be converted to a triangle. There are many algorithms for this, but a simple one will do. Below is a suggestion in pseudo code (V_i is the i:th vertex in a polygon P with n vertices).

```
for i=2 to n -1 do:
    store(new Polygon(V0, Vi-1, Vi))
```

Stefan Johansson
stefanj@cs.umu.se

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE

Task – OBJ file reader

For use in the project, **you should write your own OBJ file reader *or* find an existing library to use, and choose a good internal data representation for the 3D model**. The file reader should be implemented in C++. You can find some examples of OBJ files under Files on Canvas to test on, but feel free to search for more models on the web.

Consider the following:

- Avoid duplicated data.
- No need for fast access to edge data.
- It should be fast to copy data to an OpenGL buffer, especially vertex data. Avoid linked lists for that. The vector class in C++ standard library is suitable (`std::vector`).
- Triangulation is a requirement.
- Color information can be fun but not a requirement.
- It is recommended to normalize the object to fit into a unit cube (1.0×1.0×1.0). This is a requirement for the project.
- Vertex data is usually 3 floats (not double) data. 4 floats are also possible to use, where the 4th element is called w and usually set to 1.0 (or 0.0 for vectors).
- Prepare for additional vector data, like normals and texture coordinates, to your polygons (used later on in the project).
- Also look at the other exercise on Matrix routines for inspiration for data storage.

Examples of available OBJ file readers that can be used are:

- Tiny obj loader (`https://github.com/tinyobjloader/`),
- Obj loader (`http://www.kixor.net/dev/objloader/`), and
- The Open-Asset-Importer-Lib (`http://www.assimp.org/`, installed on the Linux computers at CS)