

EECS 598-1
Lecture 1: Polygonal meshes

Igor Guskov

January 10, 2002

1 Definitions

Polygonal mesh is a collection of polygons in space. We will typically work in a d -dimensional Euclidean space with some fixed coordinate system, so that coordinates of a point is a vector in \mathbf{R}^d (typical values for d are two, three, and four).

More formally, we will use terms *triangular mesh*, *triangle soup*, *point cloud*, *simplicial complex*.

Point cloud is a set of points in \mathbf{R}^d .

Triangular soup is a set of triangles with vertices in \mathbf{R}^d . Formally, triangle is a *2-simplex*. See [Ede] lecture 2.1 for the definitions of d -simplex and for many other definitions as well. To make a long story short,

- a point is a *0-simplex*: $\{p_0\}$,
- a segment is a *1-simplex*: $\{p_0, p_1\}$,
- a triangle is a *2-simplex*: $\{p_0, p_1, p_2\}$,
- a tetrahedron is a *3-simplex*: $\{p_0, p_1, p_2, p_3\}$

Dimension $\dim\{p_0, \dots, p_d\} = d$ if points p_0, \dots, p_d are affinely independent.

Triangular mesh is an abstract 2-simplicial complex that is a manifold with boundary together with a coordinate mapping for its vertices $x : V \rightarrow \mathbf{R}^d$.

For definition of manifold see [Ede] Lecture 2.2. Here we describe it less formally.

Manifold property of a mesh: every vertex of the mesh has a neighborhood that is homeomorphic to a disc or a half disc. For a mesh this means

that for a vertex its neighboring faces are arranged into either an umbrella or a fan.

Oriented simplices: $[p_0, \dots, p_d]$ – order is important.

Two oriented simplices are equivalent if one is an even permutation of the other. For $d > 0$ that forms two equivalence classes. Like: clockwise and counter-clockwise for triangles.

Triangular mesh is orientable if we can orient all the triangles consistently such that any two adjacent across an edge encounter that edge in opposite order. For example, $[v_1, v_2, v_3]$ and $[v_3, v_2, v_4]$ share edge $\{v_2, v_3\}$.

Boundaries of solid objects are orientable manifolds.

2 Sample data structures

```
struct CVec { // three component vector/point
    float m_pxyz[3];
};
```

2.1 Face-based

```
namespace facebased {
    struct Vertex {
        Face* m_pf; // points to some adjacent face
        CVec m_coords; // coordinates
    };

    struct Face {
        Face* m_pneis[3]; // pointers to adjacent faces
        Vertex* m_pvs[3]; // pointers to vertices of this face
        // ... you may wanna store permutations here
    };

    struct Mesh {
        vector<Face*> m_fc; // face container
        vector<Vertex*> m_vc; // vertex container -- okay if no vertex removals
    };
};
```

Good if we store some information on faces (like quadtrees for subdivision).

2.2 Edge-based

```
namespace edgebased {
  struct Edge {
    Edge* m_adj_edges[4]; // adjacent edges with some convention
    Vertex* m_pvs[2]; // two adjacent vertices
  };
  struct Vertex {
    Edge* m_pe; // points to some adjacent edge
    CVec m_coords; // coordinates
  };
  struct Mesh {
    vector<Edge*> m_ec; // edge container
    vector<Vertex*> m_vc; // vertex container
  };
};
```

Can represent non-triangular meshes without any changes.

3 Navigating face-based mesh

A convenient navigation can be done via “directed edge-face” pairs (EFs). This can be stored as a (face, edge-index) pair in a face-based rep and (edge, direction) in an edge-based one.

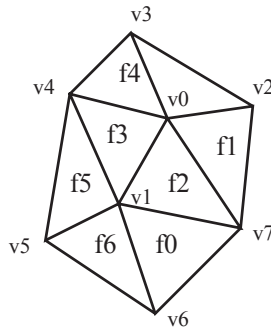


Figure 1: *A triangular mesh example.*

So that, in the mesh in Figure 1, we can have $EF(f_2, [v_7, v_0])$.

A neighbor information can be thought of as an operation on EFs. We can define operations on EFs:

1. $Org : EF \rightarrow V$. Origin vertex of an EF. Ex: $Org(EF(f3, [v1, v0])) = v1$.
2. $Dest : EF \rightarrow V$. Destination vertex of an EF. Ex: $Org(EF(f3, [v1, v0])) = v0$.
3. $Fnext : EF \rightarrow EF$. Keep the same directed edge but change the face to the neighboring one. Ex: $Fnext(EF(f3, [v1, v0])) = EF(f2, [v1, v0])$.
4. $Enext : EF \rightarrow EF$. Keep the same face but change the edge to the next edge so that $Org(Enext(ef)) == Dest(ef)$ and $Enext(Enext(Enext(ef))) == ef$. Ex: $Enext(EF(f3, [v1, v0])) = EF(f3, [v0, v4])$.
5. $Sym : EF \rightarrow EF$. Keep the same face but flip the edge's direction to opposite so that $Org(Sym(ef)) == Dest(ef)$ and $Dest(Sym(ef)) == Org(ef)$. Ex: $Enext(EF(f3, [v1, v0])) = EF(f3, [v0, v1])$.

With EFs we can go around a vertex using $ef = Enext(Sym(Fnext(ef)))$. This keeps the origin of ef intact. This can be useful when computing a normal for a vertex.

We can store “permutations” for neighbors.

4 Local mesh operations: bisection, edge flip, edge collapse

- Bisection
- Edge flip
- Edge collapse: more on this one later

5 Euler formula

Euler characteristic $\chi K = s_0 - s_1 + s_2 - \dots + (-1)^d s_d$ where $d = \dim K$ and s_i is the number of i -simplices in K . See [Ede] Lecture 2.3.

For a triangular mesh we have $\chi(M) = N_V - N_E + N_F$.

For any triangulation T of the closed disk we have $\chi(T) = 1$.

For a connected orientable manifold without boundary, the number of handles is called its *genus* and can be computed as $g(M) = 1 - \chi(M)/2$.

If we have boundaries then we got $g(M) = 1 - (\chi(M) + N_B)/2$ where N_B is the number of different boundaries.

6 Normal at a vertex

Following Siggraph 2000 subdivision course notes [ZS00] page 71, the normal at a vertex can be computed as $t_1 \times t_2$ where t_i are tangent vectors computed as:

$$t_1 = \sum_{i=0}^{k-1} \cos \frac{2\pi i}{k} p_i$$
$$t_2 = \sum_{i=0}^{k-1} \sin \frac{2\pi i}{k} p_i$$

Example: valence four. Masks $[1, 0, -1, 0]$ and $[0, 1, 0 - 1]$.

7 Wavefront OBJ file format

```
# OBJ file format with ext .obj
v 1.0 0.0 0.0
v 0.0 1.0 0.0
v 0.0 0.0 1.0
v 0.0 0.0 0.0
f 2 4 3
f 4 2 1
f 3 1 2
f 1 3 4
```

8 Rendering in OpenGL

Flat shaded

```
glShadeModel(GL_FLAT);
glEnable(GL_LIGHTING);
...
glBegin(GL_TRIANGLES);
for(...) {
    ....
    glNormal3fv(n);
    glVertex3fv(v1);
    glVertex3fv(v2);
    glVertex3fv(v3);
}
```

```
}  
glEnd();
```

Smooth shaded

```
glShadeModel(GL_SMOOTH);  
glEnable(GL_LIGHTING);  
...  
glBegin(GL_TRIANGLES);  
for(....) {  
    ....  
    glNormal3fv(n1);  
    glVertex3fv(v1);  
    glNormal3fv(n2);  
    glVertex3fv(v2);  
    glNormal3fv(n3);  
    glVertex3fv(v3);  
    ....  
}  
glEnd();
```

9 Next time

Progressive meshes. Mesh simplification. Curvature operator.

References

- [Ede] H. Edelsbrunner. [Set of lecture notes](#).
- [ZS00] Denis Zorin and Peter Schröder, editors. [Subdivision for Modeling and Animation](#). Course Notes. ACM Siggraph, 2000.