Lecturer: Justin Solomon
Scribe: Adrian Buganza-Tepole

# CS 468 (Spring 2013) — Discrete Differential Geometry

Lecture 6: Discrete Surfaces

## Manifold meshes

To study the differential geometry of discrete surfaces we first need to introduce a way to represent them. The most obvious choice is the use of polygonal meshes and, in particular, manifold polygonal meshes. A polygonal mesh consists of three types of elements: vertices, edges and faces. The information that describes the elements includes connectivity and geometry. The geometrical information refers to the position of the vertices in space, while the topological information relates to the properties that remain invariant under certain transformations, such as the edges that define a face or the vertices that define and edge.

A mesh is a manifold if it satisfies:

- Each edge is incident to only one or two faces
- The faces incident to a vertex form a closed or an open fan

For now we will make the assumption that the surfaces we are interested in are manifolds.

One key consideration in order to move forward is to determine whether or not a polygonal mesh is a good approximation of a continuous surface. In a polygonal mesh, each of the edges can be thought of as the approximation of a curve over the surface. As discussed for the case of curves, piece-wise linear functions are reasonable building blocks since they approximate the exact curve with order $O(h^2)$, where $h$ is the distance between two vertices.
This approximation property comes from the use of the mean value theorem. Without loss of generality let's consider $f(t) := \mathbb{R} \to \mathbb{R}$, a curve in one dimension. We approximate $f(t)$ with piece-wise linear functions $g_i(t)$ where the subscript $i$ denotes a particular interval. Let $[a, b] \in \mathbb{R}$ be one of such intervals and $g(t)$ the linear approximation of $f(t)$ in that interval, then, the mean value theorem states that there is a point $c \in [a, b]$ such that $f'(c) = g'(c)$. Further, this implies that if we expand $f(t)$ in Taylor series around $f(c)$ we get $f(c) = g(c) + O(h^2)$ where $h = b - a$ is the length of the interval we are interested in. Therefore, at $c$, our linear approximation $g(t)$ converges to the exact curve $f(t)$ quadratically. The three dimensional case follows directly, but now we have three functions $\mathbf{f}(t) = [f_x(t), f_y(t), f_z(t)]$, one for each component. Thus, intuitively, using piece-wise linear building blocks to create polygonal meshes should converge to the exact surface as the polygons diminish in size. The advantages of polygonal meshes from a practical standpoint are:

- Simple to render
- Arbitrary topology possible
- Basis for subdivision refinement

## Triangular meshes

We narrow our scope to the study of triangular manifold meshes. One of the advantages of considering this representation is that we can gain insight about some global and local properties of the mesh easily. For example, narrowing the scope to triangular meshes we can relate the Euler characteristic, which is a global property of the mesh, to the valence of a vertex which is a local property. The valence is the number of edges incident to a vertex. The Euler characterstic relates to the number of holes in the mesh. The Euler characteristic can be obtained from:

$$\chi = V - E + F \tag{1}$$

where $\chi$ is the Euler characteristic, $V$ is the number of vertices, $E$ the number of edges and $F$ the number of faces. It can be shown that:

$$\chi = 2 - 2g \tag{2}$$

where $g$ is the number of holes in the surface. For example for a sphere $g = 0$ and for a donut $g = 1$. For triangle meshes, we can relate the Euler characteristic to the average valence of a vertex. For a closed triangular mesh it is easy to see that $2E = 3F$ and thus $V - F/2 = \chi$. Further, we can assume that $\chi$ is a small number and thus for a large mesh we can estimate $F \approx 2V$. This yields $E \approx 3V$ or $6V \approx 2E$; $2E$ and thus the average valence must be 6 because if we count the edges incident to a vertex we will end up counting each edge twice.

## Mesh storage and queries

There are several strategies to store the information of a mesh, and the choice of one over another depends on the application. The most common data structures for triangular meshes are:

- Triangle soup: stores, for each triangle, the position of its three vertices
- Shared vertex structure: stores, for each vertex, its position, additionally, for each face, it stores its three vertices.
- Half-edge: stores vertices and faces but it introduces the half-edge which will be covered in more detailed below, but briefly, for each edge it stores two half-edges, each associated with a single face.
- Quad-edge: similar to the half-edge but, in addition, it stores the dual mesh by considering the dual edges connecting pairs of faces.

### Triangle soup

- Pros:
  - Little information to store
  - Linear memory access
- Cons:
  - No topology
  - Redundant information (repeated vertices)
- Applications:
  - Simple rendering, such as openGL where topology is not needed and linear memory access is advantageous for rapid rendering

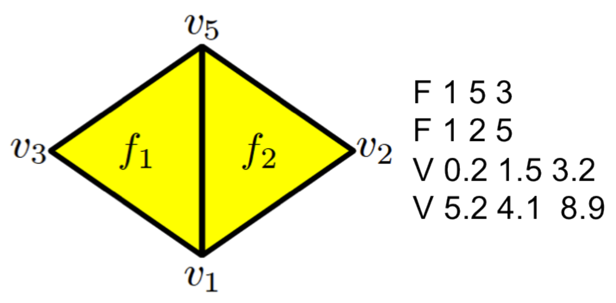


Figure 1: shared vertex

**Shared vertex**

Figure 1 illustrates this data structure.

- Pros:
  - Topology is stored
  - Still little storage requirements
  - Intuitive, easy to interpret
- Cons:
  - Memory access is random, this results in non cash friendly programs
  - Simple queries need too much computation time because even though the topology is stored only the vertices of a face are available directly, any other query requires search over all faces.
- Applications:
  - Very common to share meshes because it includes the topological information, it s simple to store and easy to read out from a file. Many CAD and modeling software process .obj format which uses the shared vertex data structure.

**Half-edge**

As discussed above, the main limitation of the shared-vertex data structure is the inability to perform queries easily. Some examples of common queries are:

- Neighboring vertices to a vertex
- Neighboring faces to an edge
- Edges adjacent to a face
- ...

This queries are localized, thus, only local information storage would be sufficient. The half-edge is an excellent strategy because it stores only slightly more information than the shared-vertex structure but it allows to traverse the mesh easily. The idea behind this data structure is to split each edge into two half-edges, each of with have a single face associated to them as illustrated in Figure 2. The data is stored in the following manner:

- Vertex: stores coordinates and an arbitrary outgoing half edge

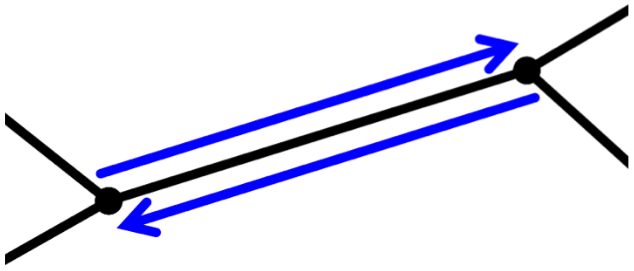


Figure 2: half edge

- Face: stores its three vertices and an arbitrary half edge
- Half-edge: stores
  - Vertex: the origin vertex of the half-edge
  - Flip: the opposite half-edge
  - Next: the half-edge of the vertex the half-edge points to
  - Face: the associated face of the half-edge

Figure 3 illustrates all the data that is stored for a single triangle.

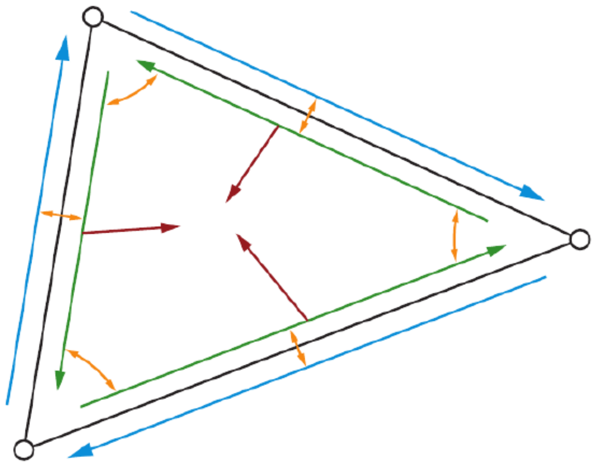
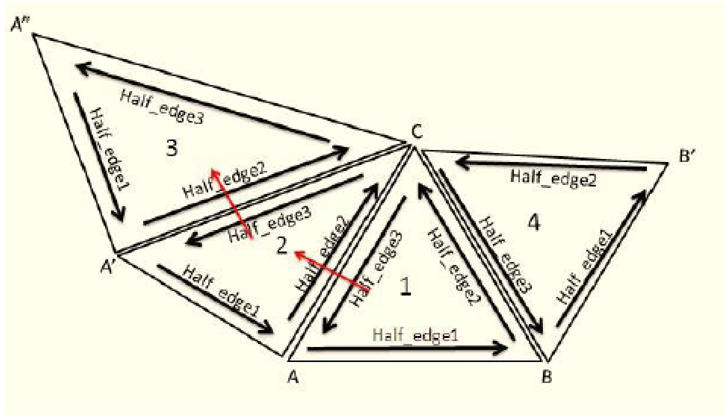


Figure 3: data stored with the half edge approach for a single triangle

It can be appreciated that obtaining the local topology does not longer require loops over the entire list of faces like in the case of the shared-vertex structure. Using the half-edge one can, for example, easily loop over the surrounding vertices of a vertex. This is illustrated in the pseudocode and the illustration provided in figure 4 where it is shown than starting from an arbitrary half-edge store at the vertex of interest we can get the next half-edge without loops, we have immediate access to the `flip` and the `next` information stored at the half-edge and that information is enough to navigate through the half-edges incident to the vertex of interest. Retrieving the opposite vertex of a half-edge is also immediate.

The use of this structure, however, carries some limitations as well. Summarizing, the pros and cons of this strategy are:

- Pros:



```
Iterate(v):
startEdge = v.out;
e = startEdge;
do
    process(e.flip.from)
    e = e.flip.next
while e != startEdge
```

Figure 4: Left: circulating over the vertices that surround a vertex of interest using the half-edge. Right: pseudocode.

  - Queries and movement across the mesh are easy to implement
  - Queries and movement across the mesh require less computational time than the shared-vertex structure

- Cons:
  - Requires more memory than the shared vertex structure
  - It still leads to random memory access, so it is not the best option for fast rendering
  - Less intuitive, more sophisticated strategy, making it more difficult to share models widely

- Applications:
  - Extremely useful when queries are needed, for example, in discrete differential geometry of surfaces.

### Quad-edge

At a first glance, it seems like the half-edge solves most of the needs for surface representation, storage and queries, however, in the realm of discrete differential geometry, more questions arise that could need yet a more specialized data structure. One natural example comes from the need to represent functions over the surfaces. Let us consider a function $f := \mathbb{R}^3 \to \mathbb{R}$ defined over a surface $S$. Let $S^*$ be the approximation of the surface $S$ by means of a triangular manifold mesh. The most natural way to represent $f$ over $S^*$ is to take the values of the function at single points, namely, the vertices, this leads to representing $f$ via the set of values over the vertices $f^* := \mathbb{R}^{[V]} \to \mathbb{R}$. Now, we might be interested in doing analysis on $f$ or rather, on the set $f^*$. For example, a simple operation would be to integrate the function over the surface, we want to calculate:

$$\int_{S^*} f^* \mathrm{dA} \approx \int_{\mathrm{S}} \mathrm{fdA} \tag{3}$$

One possible way would be to define hat functions on the vertices which is an approach used in finite element analysis. Hat functions are particular examples of the more general local support functions. Local support functions are unit valued at a single vertex and they vanish in the rest of the vertices. This is illustrated in figure 5. Let $N^i$ be the local support function associated with vertex $i$ and $f_i^*$ be the value of $f$ at the same vertex. Then we can reconstruct a continuous function $\hat{f} \approx f$ using $f^*$ and $N^i$:

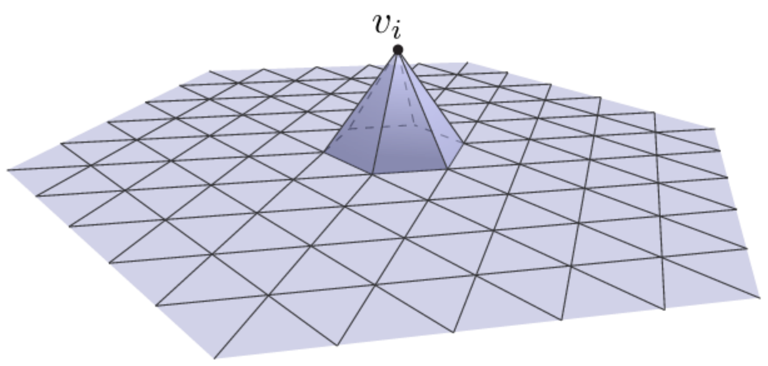


Figure 5: hat function of vertex i

$$\hat{f} = \sum_i N^i f_i^* \tag{4}$$

Since the functions $N^i$ have local support, we can analyze $\hat{f}$ locally over each triangle $F_j$:

$$\hat{f}_j = \sum_{i=1}^{3} N_j^i f_i^* \tag{5}$$

In this manner, integration of $f$ over a triangle $F_j$ can be approximated as:

$$\int_{F_j} f \mathrm{dA} \approx \int_{F_j} \hat{f} \mathrm{dA} \tag{6}$$

This is not the only possibility. An alternative is to used directly the value of $f$ at each vertex and associate some area to that vertex, this would lead to:

$$\int_S f \mathrm{dA} \approx \sum_{\mathrm{i}} \mathrm{f}_{\mathrm{i}}^* \mathrm{da}_{\mathrm{i}} \tag{7}$$

where $da_i$ is the area associated with vertex $i$. This poses the problem of adequately determining $da$ for each vertex, to accomplish this we consider the dual complex. The dual of a mesh is another mesh in which each face of the original mesh is replaced by a vertex. Edges in the dual mesh connect faces in the original mesh. This is illustrated in figure 6 where we can see the original mesh in green, the dual mesh is shown in blue. Extending the idea of the half-edge to edges in the dual mesh we obtain the quad-edge depicted in figure 7, where the quad edge is formed with the two pairs of half edges, one pair corresponding to the original mesh shown in red and another pair corresponding to the dual mesh colored in yellow. The quad-edge stores the information stored for the half-edge plus the *Rot* of the half-edge. The *Rot* operation takes us from the original to the dual and from the dual to the original as illustrated in figure 8. In conclusion, the quad-edge stores two meshes, both the primal and the dual. Summarizing:

- Pros:
  - Queries and movement across the mesh are easy to implement
  - The dual mesh is stored and can also be traversed easily
  - Calculations that require the dual mesh are possible, such as integration over the surface
- Cons:

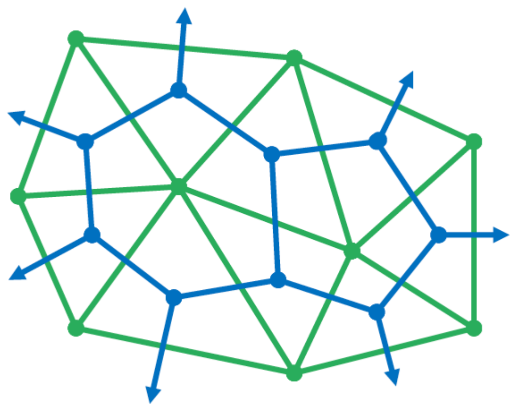


Figure 6: Green: original mesh. Blue: dual mesh

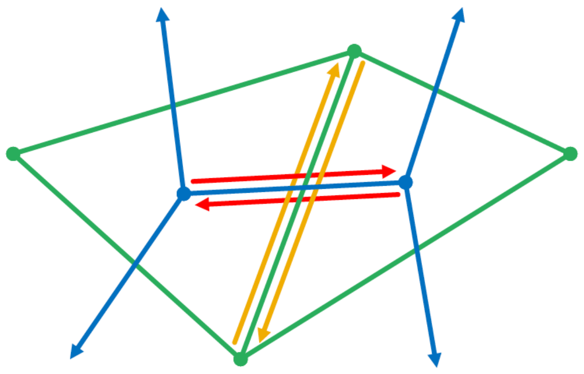


Figure 7: Quad edge. Original mesh is shown in green, dual mesh is colored in blue. The quad edge is formed with the two pairs of half edges, one pair corresponding to the original mesh shown in red and another pair corresponding to the dual mesh colored in yellow.

  - Requires even more memory than the half-edge structure
  - It maintains the random memory access of the half-edge
  - It is even less intuitive than the half-edge

- Applications:
  - Useful when queries over the dual are needed, for example, for integration over the surface.

## Obtaining geometry

Meshes are not the only way to represent geometries. Alternatives are:

- Implicit surfaces: for example, approximating the surface using the distance function.
- Point clouds: easily available from various imaging techniques
- Splines: from CAD software

However, it is desirable to end up with meshes for the purpose of analysis, and many times it is required to go from point clouds to triangulations. There are several algorithms to do this, to name a few we have:

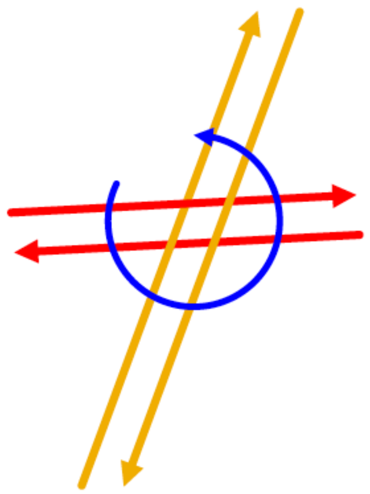


Figure 8: Rot operation

- Marching cubes
- Delaunay triangulation
- Poisson reconstruction