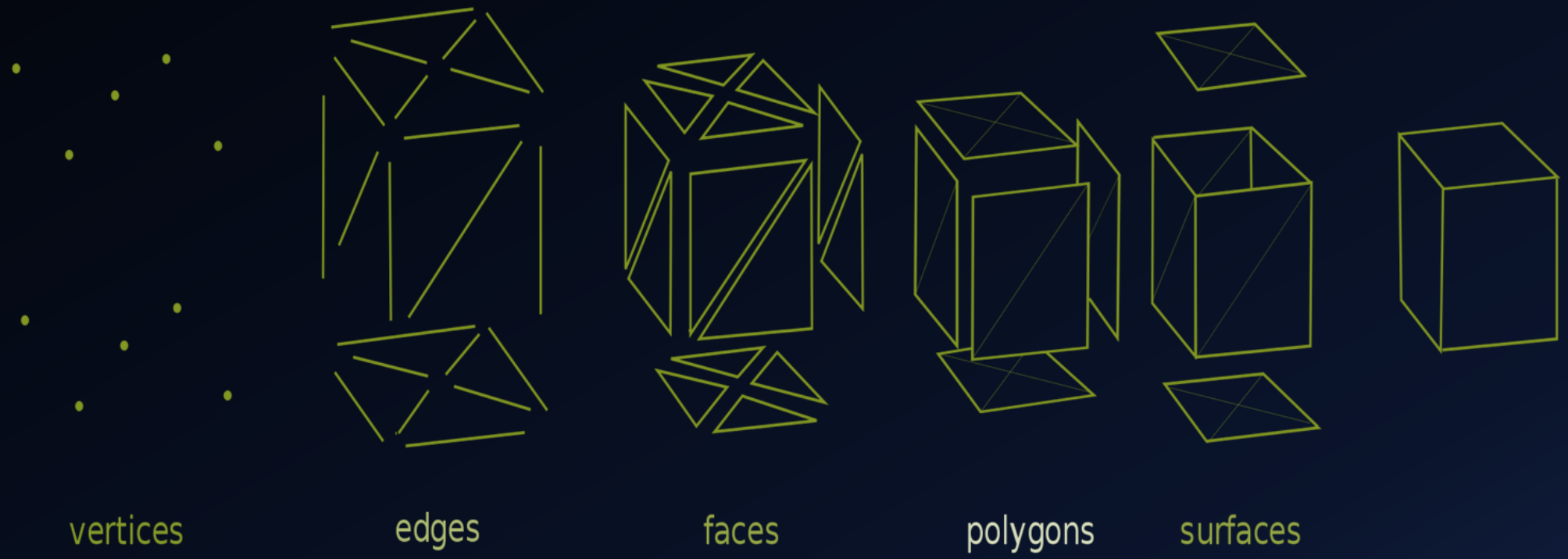
POLYGON MESHES

- Introduction
- Vertex-vertex meshes
- Face-vertex meshes
- Winged-edge meshes
- File structures for storing meshes

Author: prof. Yevhenii Borodavka

Introduction

In 3D computer graphics and solid modeling, a polygon mesh is a collection of vertices, edges, and faces that define the shape of a polyhedral object. The faces usually consist of triangles (triangle mesh), quadrilaterals (quads), or other simple convex polygons (n-gons), since this simplifies rendering, but may also be more generally composed of concave polygons, or even polygons with holes.



Introduction

Objects created with polygon meshes must store different types of elements. These include vertices, edges, faces, polygons, and surfaces. In many applications, only vertices, edges, and either faces or polygons are stored.

Vertex — a position (usually in 3D space) along with other information such as color, normal vector, and texture coordinates.

Edge — a connection between two vertices.

Face — a closed set of edges, in which a triangle face has three edges, and a quad face has four edges. A polygon is a coplanar set of faces. In systems that support multi-sided faces, polygons and faces are equivalent. However, most rendering hardware supports only 3- or 4-sided faces, so polygons are represented as multiple faces.

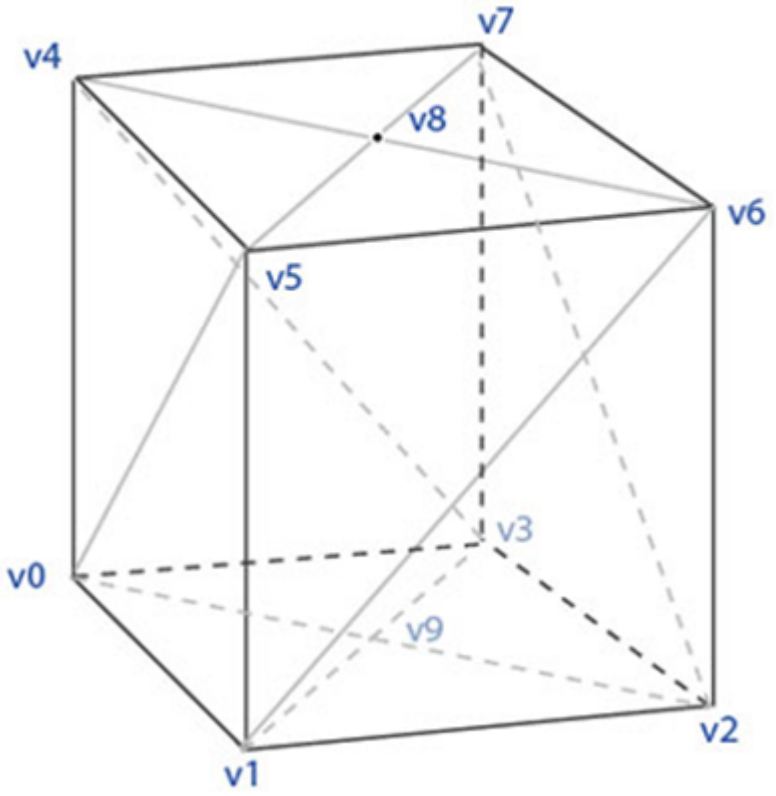
Polygon meshes may be represented in a variety of ways, using different methods to store the vertex, edge, and face data.

Vertex-vertex meshes

Vertex-Vertex Meshes (VV)

Vertex List

v0	0,0,0	v1 v5 v4 v3 v9
v1	1,0,0	v2 v6 v5 v0 v9
v2	1,1,0	v3 v7 v6 v1 v9
v3	0,1,0	v2 v6 v7 v4 v9
v4	0,0,1	v5 v0 v3 v7 v8
v5	1,0,1	v6 v1 v0 v4 v8
v6	1,1,1	v7 v2 v1 v5 v8
v7	0,1,1	v4 v3 v2 v6 v8
v8	.5,.5,1	v4 v5 v6 v7
v9	.5,.5,0	v0 v1 v2 v3



Vertex-vertex meshes

Vertex-vertex (VV) meshes represent an object as a set of vertices connected to other vertices. This is the simplest representation, but not widely used since the face and edge information is implicit. Thus, it is necessary to traverse the data in order to generate a list of faces for rendering. In addition, operations on edges and faces are not easily accomplished.

However, VV meshes benefit from small storage space and efficient morphing of shape. The above figure shows a four-sided box as represented by a VV mesh. Each vertex indexes its neighboring vertices. The last two vertices, 8 and 9 at the top and bottom center of the "box-cylinder", have four connected vertices rather than five. A general system must be able to handle an arbitrary number of vertices connected to any given vertex.

FACE-VERTEX MESHES

Face-Vertex Meshes

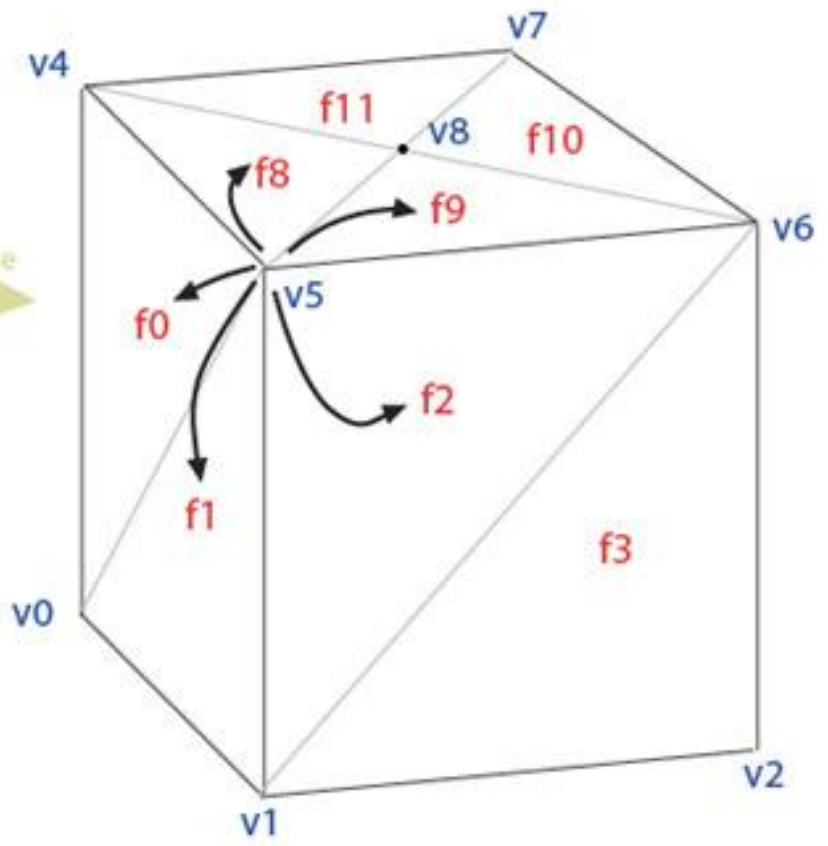
Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

Vertex List

v0	0,0,0	f0 f1 f12 f15 f7
v1	1,0,0	f2 f3 f13 f12 f1
v2	1,1,0	f4 f5 f14 f13 f3
v3	0,1,0	f6 f7 f15 f14 f5
v4	0,0,1	f6 f7 f0 f8 f11
v5	1,0,1	f0 f1 f2 f9 f8
v6	1,1,1	f2 f3 f4 f10 f9
v7	0,1,1	f4 f5 f6 f11 f10
v8	.5,.5,0	f8 f9 f10 f11
v9	.5,.5,1	f12 13 14 15

example



Face-vertex meshes

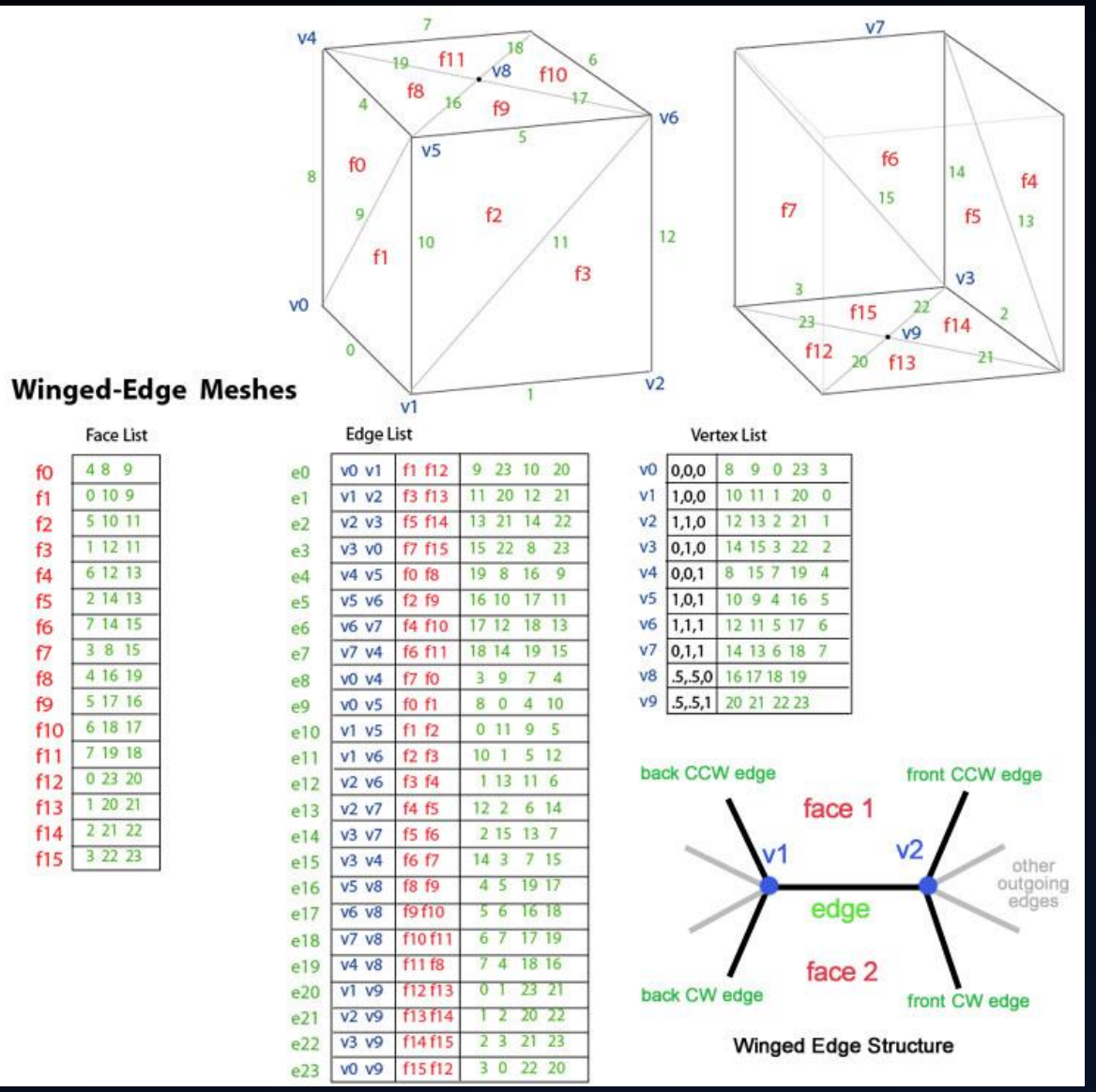
Face-vertex (FV) meshes represent an object as a set of faces and a set of vertices. This is the most widely used mesh representation, being the input typically accepted by modern graphics hardware.

Face-vertex meshes improve on VV-mesh for modeling in that they allow explicit lookup of the vertices of a face, and the faces surrounding a vertex. The above figure shows the "box-cylinder" example as an FV mesh. Vertex **v5** is highlighted to show the faces that surround it. Notice that, in this example, every face is required to have exactly 3 vertices. However, this does not mean every vertex has the same number of surrounding faces.

For rendering, the face list is usually transmitted to the GPU as a set of indices to vertices, and the vertices are sent as position/color/normal structures (in the figure, only position is given). This has the benefit that changes in shape, but not geometry can be dynamically updated by simply resending the vertex data without updating the face connectivity.

Modeling requires easy traversal of all structures. With face-vertex meshes, it is easy to find the vertices of a face. Also, the vertex list contains a list of faces connected to each vertex. Unlike VV meshes, both faces and vertices are explicit, so locating neighboring faces and vertices is constant time. However, the edges are implicit, so a search is still needed to find all the faces surrounding a given face. Other dynamic operations, such as splitting or merging a face, are also difficult with face-vertex meshes.

Winged-edge meshes



Introduced by Baumgart in 1975, winged-edge meshes explicitly represent the vertices, faces, and edges of a mesh. This representation is widely used in modeling programs to provide the greatest flexibility in dynamically changing the mesh geometry because split and merge operations can be done quickly. Their primary drawback is large storage requirements and increased complexity due to maintaining many indices.

Winged-edge meshes

Winged-edge meshes address the issue of traversing from edge to edge and providing an ordered set of faces around an edge. For any given edge, the number of outgoing edges may be arbitrary. To simplify this, winged-edge meshes provide only four, the nearest clockwise and counter-clockwise edges at each end. The other edges may be traversed incrementally. The information for each edge therefore resembles a butterfly, hence "winged-edge" meshes. The above figure shows the "box-cylinder" as a winged-edge mesh. The total data for an edge consists of 2 vertices (endpoints), 2 faces (on each side), and 4 edges (winged-edge).

Rendering of winged-edge meshes for graphics hardware requires generating a Face index list. This is usually done only when the geometry changes. Winged-edge meshes are ideally suited for dynamic geometry, such as subdivision surfaces and interactive modeling since changes to the mesh can occur locally. Traversal across the mesh, as might be needed for collision detection, can be accomplished efficiently.

File structures for storing meshes

There are several file structures to store meshes. We will consider three of them: explicit representation, list of vertices, and list of edges.

In explicit representation, an object consists of a set of faces, each of which is a polygon consisting of a sequence of vertex coordinates:

Faces	Coordinates
f_1	$x_1y_1z_1$, $x_2y_2z_2$, $x_6y_6z_6$, $x_5y_5z_5$
f_2	$x_2y_2z_2$, $x_3y_3z_3$, $x_7y_7z_7$, $x_6y_6z_6$

The disadvantages of this representation are that, firstly, the relationships of the faces are set implicitly, and secondly, the coordinates of each vertex appear as many times as the faces have this vertex.

File structures for storing meshes

To avoid repeating the coordinates of the vertices, you can separate them into an independent structure. In this case, not the coordinates of the vertices, as in the previous case, but their indices in the array of vertex coordinates are associated with the faces. Example:

Vertices	Coordinates	Faces	Vertices
v_1	$x_1y_1z_1$	f_1	$v_1v_2v_3v_4$
v_2	$x_2y_2z_2$	f_2	$v_6v_2v_1v_5$
v_3	$x_3y_3z_3$	f_3	$v_7v_3v_2v_6$
v_4	$x_4y_4z_4$	f_4	$v_8v_4v_3v_7$
v_5	$x_5y_5z_5$	f_5	$v_5v_1v_4v_8$
v_6	$x_6y_6z_6$	f_6	$v_8v_7v_6v_5$
v_7	$x_7y_7z_7$		
v_8	$x_8y_8z_8$		

File structures for storing meshes

The list of edges describes faces through the edges. The faces vertices are defined explicitly via edges. Example:

Edges	Vertices	Vertices	Coordinates	Faces	Edges
e_1	v_1v_2	v_1	$x_1y_1z_1$	f_1	$e_1e_2e_3e_4$
e_2	v_2v_3	v_2	$x_2y_2z_2$	f_2	$e_9e_6e_1e_5$
e_3	v_3v_4	v_3	$x_3y_3z_3$	f_3	$e_{10}e_7e_2e_6$
e_4	v_4v_1	v_4	$x_4y_4z_4$	f_4	$e_{11}e_8e_7e_3$
e_5	v_1v_5	v_5	$x_5y_5z_5$	f_5	$e_{12}e_5e_4e_8$
e_6	v_2v_6	v_6	$x_6y_6z_6$	f_6	$e_{12}e_{11}e_{10}e_9$
e_7	v_3v_7	v_7	$x_7y_7z_7$		
e_8	v_4v_8	v_8	$x_8y_8z_8$		
e_9	v_5v_6				
e_{10}	v_6v_7				
e_{11}	v_7v_8				
e_{12}	v_8v_5				

Thank you!