Computer Graphics II
4: Polygon Meshes

Computer Graphics and
Multimedia Systems Group

University of Siegen

Structure of this Chapter

Structure of Chapter

- *Subsection 1: Basics of polygon meshes*
- *Subsection 2: Data Structures for Polygon Meshes*
- *Subsection 3: Special Data Structure: Mesh Silhouettes*

Motivation

1. *Real-time-graphics: Convertion of other geometry types into polygons/triangles is required*
2. *Many geometries are only available in form of polygons*
3. *Until now: Focus of representation (for rendering), e.g. using vertex- and index-lists*
4. *But: Tasks like mesh manipulation and mesh processing require efficient access to the neighborhood of a vertex or face*

Notation (Polygon-Meshes)

- *Set of vertices:* $\mathcal{V} = \{\mathbf{V}_i\},\ i = 1, \ldots, N_V$
- *Set of edges:* $\mathcal{E} = \{\mathbf{E}_{ij}\}, \mathbf{E}_{ij} = \overline{\mathbf{V}_i\mathbf{V}_j},\ i, j \in \{1, \ldots, N_V\},\ N_E = |\mathcal{E}|$
- *Set of flat polygons/surfaces:* $\mathcal{F} = \{\mathbf{F}_i\},\ i = 1, \ldots, N_F$

Between these entities there exist the following relationships:

Adjacency (contiguous): $\overline{\mathbf{V}_i\mathbf{V}_j}$ *connetcs vertices* $\mathbf{V}_i, \mathbf{V}_j$*, polygon* $\mathbf{F}$ *connects the respective edges and vertices*

Valency: *The number of edges connected to a vertex*

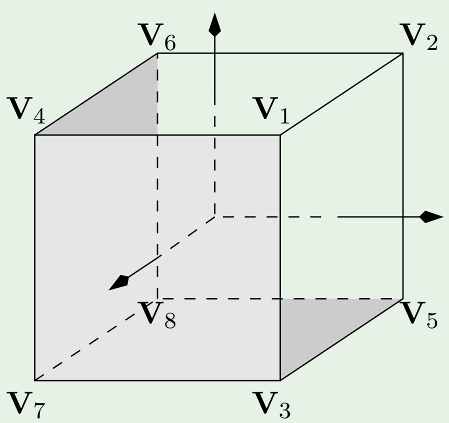
1-Neighbourhood of $\mathbf{V}$: *All (directly) connected polygons, edges and vertices*

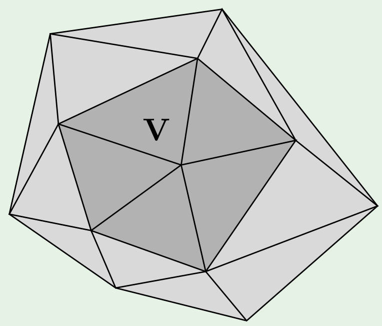
n-Neighbourhood of $\mathbf{V}$: *The 1-neighbourhood of the $(n-1)$-neighbourhood*

Example

Cube Vertices, edges and surfaces result in

- $\mathcal{V} = \{(1,1,1), (1,1,-1), (1,-1,1), (-1,1,1), (1,-1,-1), (-1,1,-1), (-1,-1,1), (-1,-1,-1)\}$
- $\mathcal{E} = \{\mathbf{E}_{1,2}, \mathbf{E}_{1,3}, \mathbf{E}_{1,4}, \mathbf{E}_{2,5}, \mathbf{E}_{2,6}, \mathbf{E}_{3,5}, \mathbf{E}_{3,7}, \mathbf{E}_{4,6}, \mathbf{E}_{4,7}, \mathbf{E}_{5,8}, \mathbf{E}_{6,8}, \mathbf{E}_{7,8}\}$
- $\mathcal{F} = \{\{1,3,5,2\}, \{1,4,7,3\}, \{1,2,6,4\}, \{8,5,3,7\}, \{8,6,2,5\}, \{8,7,4,6\}\}$



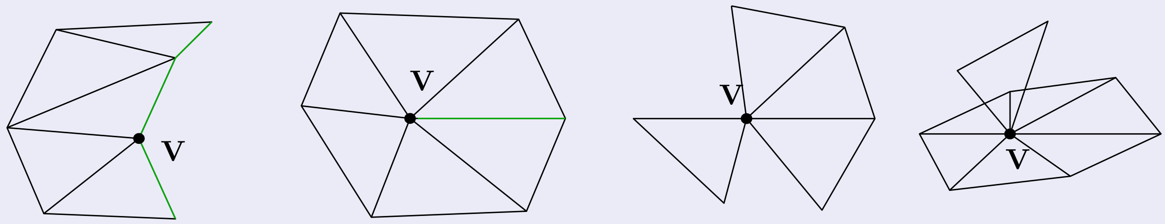


Links: 1-neighbourhood of $\mathbf{V}_7$; right: 1- and 2-neighbourhood

Property (2-Manifold Polygon Mesh)

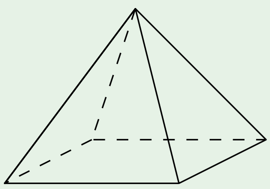
A 2-manifold polygon-mesh has the following properties:

- *No penetration: Intersection of two polygons is either a vertex, an edge or empty*
- *On one edge lies an (outer edge) or two polygons (inner edge)*
- *Polygons around a vertex: Open (boundary vertex) or closed fan (inner vertex)*
- *Mesh is orientable & consists of a single connectivity component*
- *and the Eulers formula is valid:* $\boxed{N_V - N_E + N_F = 2(1 - N_G) - N_B}$ *whereby* N_G *# penetrating holes (genus),* N_B *# boundary polylines*

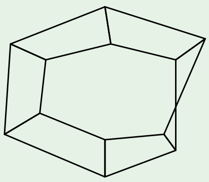


Left: Border- and inner vertices and edge, respectively; right: non-manifold

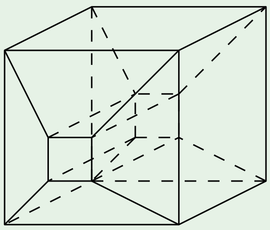
Example (Eulers Formula)



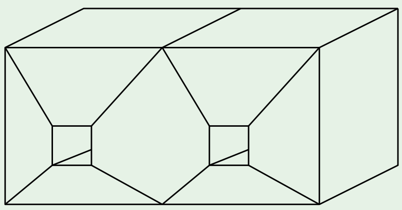
pyramid (w.o. bottom)



Möbuis strip



Cube (1 hole)



Box (2 holes)

Euler formula:

Bottomless pyramids:

$$\begin{array}{rrrl} N_V - & N_E + & N_F & = 2 - 2 \cdot N_G - N_B \\ 5 - & 8 + & 4 & = 2 - \quad\quad 0 - \quad 1 \end{array}$$

Cube with hole:

$$\begin{array}{rrrl} N_V - & N_E + & N_F & = 2 - 2 \cdot N_G - N_B \\ 16 - & 32 + & 16 & = 2 - \quad\quad 2 - \quad 0 \end{array}$$

M"obuis strip:

$$\begin{array}{rrrl} N_V - & N_E + & N_F & \stackrel{?}{=} 2 - 2 \cdot N_G - N_B \\ 12 - & 18 + & 6 & \neq 2 - \quad\quad 0 - \quad 1 \end{array}$$

Cuboid with 2 holes:

$$\begin{array}{rrrl} N_V - & N_E + & N_F & = 2 - 2 \cdot N_G - N_B \\ 28 - & 58 + & 28 & = 2 - \quad\quad 4 - \quad 0 \end{array}$$

Objective

Goal: *Applications specific structure data structure, e.g.*

- *Efficient rendering: Only triangles, efficient data transfer*
- *Mesh-editing: Efficient access to polygon data, e.g. neighbours*

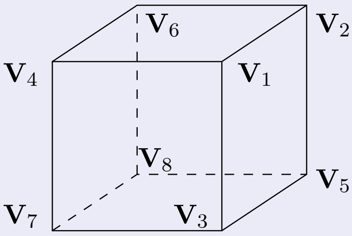
Separate storage *of geometry (vertex coord.) and topology (connectivity)*

Reminder (Shared vertex format (also: Indexed face-set))

Application: *Rendering*

Approach: *Explicit storage of the geometry, referencing for topology*

Optimization in OpenGL: `GL_TRIANGLE_STRIP`, `GL_TRIANGLE_FAN`, `GL_QUAD_STRIP` *or vertex-arrays (CG I)*



- *Vertices:* $\mathbf{V}_i = (x_i, y_i, z_i),\ i = 1, \ldots, 8$
- *Surfaces:* $\mathcal{F} = \{\{1,3,5,2\}, \{1,4,7,3\}, \{1,2,6,4\}, \{8,5,3,7\}, \{8,6,2,5\}, \{8,7,4,6\}\}$

Approach (Winged-Edge Data Structure)

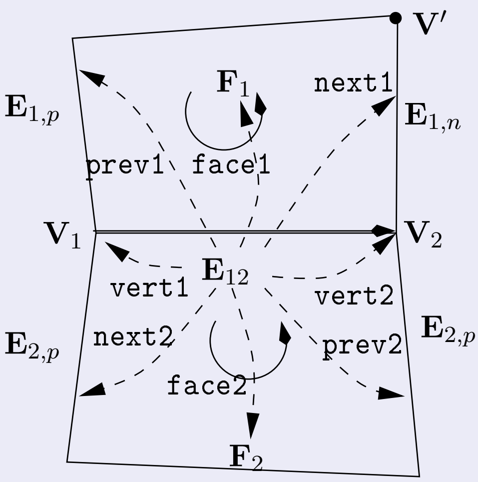
Store the following information per mesh entity

Vertex $\mathbf{V}_i$: *Coordinates & link to an edge* (`edge`)

Polygon $\mathbf{F}$: *Links to an adjacent edge* (`edge`)

Edge $\mathbf{E}_{ij}$: *Administers links to the adjacent*

- *Vertices* `vert1, vert2`
- *Polygons* `face1, face2`
- *Adjacent edges* `prev1`, `next1` *for* `face1` *(counterclockwise; analog* `face2`*)*



Remark

Fast access *to polygon edges/-corners and edges around a corner*

Problem: *Orientiented edge forces if-statement; example access to* $\mathbf{V}'$*:*

```
if( E_12->prev2->vert1 == E_12->vert2 ) V' = E_12->prev2->vert2;
else                                    V' = E_12->prev2->vert1;
```

Approach (Half-Edge Data Structure)

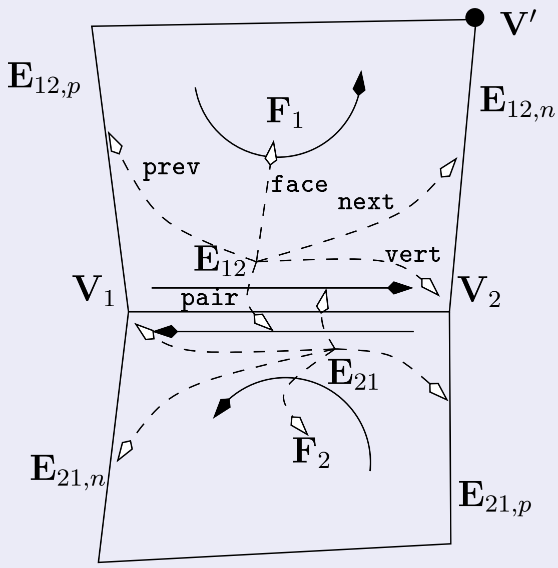
Approach: *Same as winged-edge data structure, only that edge information is distributed to half edges per each vertex*

Data per half edge:

1. *Link to end vertex* (`vert`)
2. *Link to another half-edge* (`pair`)
3. *Link to associated polygon* (`face`)
4. *Link to neighbouring half-edges* (`prev`, `next`)

Access to $\mathbf{V}'$ *without an if-statement:*

```
V' = E_12->next->vert
```



Compared to winged-edge, *the* `pair` *links require two additional pointers per edge, one for each half-edges*

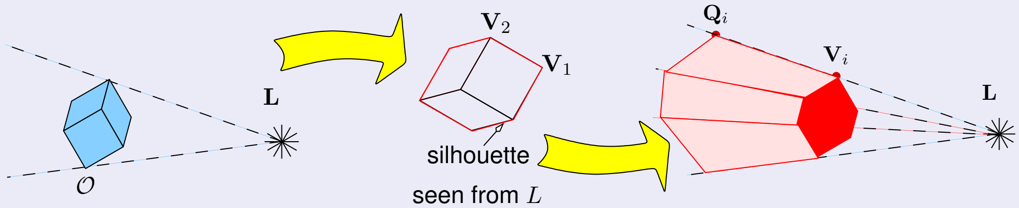
Objective (Shadow Volumes)

Given: *Polygonal object and a point light source at position* $\mathbf{L}$

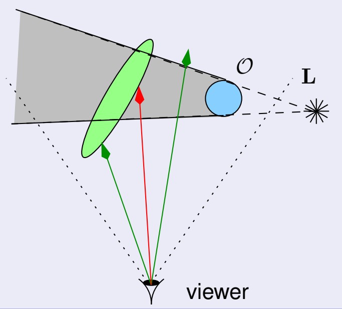
Wanted: *Casting shadows of the object* $\mathcal{O}$ *into the scene*

Shadow Volume Approach:

1. *Determine silhouettes edges* $(\mathbf{V}_1, \ldots, \mathbf{V}_k)$ *of* $\mathcal{O}$
2. *Extrude silhouettes:* $\mathbf{Q}_i = \mathbf{L} + \alpha(\mathbf{V}_i - \mathbf{L})$, α *large (shadow volume)*



3. *Rendering: First render scene normally*
4. *Add shadows to pixel, if frontface of shadow volume but not backface is visible (use z-buffer and stencil buffer)*



Algorithm

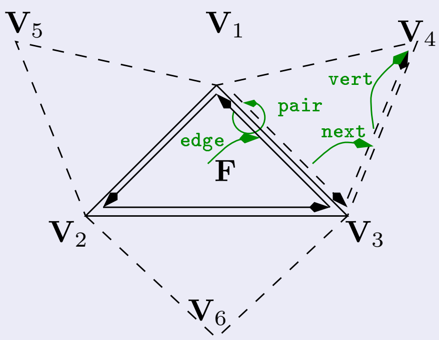
Goal: *Given a closed triangle mesh, we want to compute the silhouette egdes*

Characterization: *Edge $\mathbf{E}_{ij}$, connecting triangles $\mathbf{F}_k, \mathbf{F}_l$ is part of the silhouette, if the normals $\hat{\mathbf{n}}_k, \hat{\mathbf{n}}_l$ of the triangles point in different directions w.r.t. the light direction $\hat{\mathbf{l}}$, i.e.* $\boxed{\left(\hat{\mathbf{n}}_k \cdot \hat{\mathbf{l}}\right) \cdot \left(\hat{\mathbf{n}}_l \cdot \hat{\mathbf{l}}\right) < 0}$

Algorithm *for given triangle $\mathbf{F}$*

1. *Collect triangle vertices $\mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3$ and adjacent vertices $\mathbf{V}_4, \mathbf{V}_5, \mathbf{V}_6$*

```
E_31 = F->edge;
V_1  = E_31->vert;
V_4  = E_31->pair->next->vert;
E_12 = E_31->next;
V_2  = E_12->vert;
V_5  = E_12->pair->next->vert;
E_23 = E_12->next;
V_3  = E_23->vert;
V_6  = E_23->pair->next->vert;
```



Algorithm

Algorithm (continued)

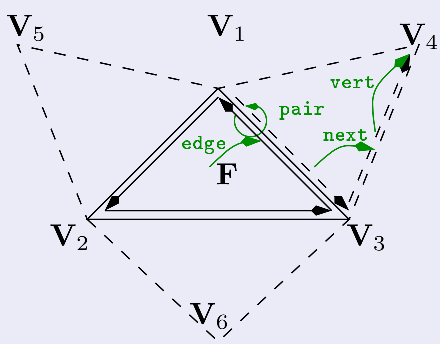
2. *Check if $\mathbf{F}$ faces towards $\mathbf{L}$:*

```
n = cross(V_2 - V_1, V_3 - V_1);
if ( n*l<0 ) continue;
```

Note: Thus each edge is drawn only once.

3. *Draw edge $\mathbf{E}_{i,(i+1)\%3}$ if neighbouring triangle faces backwards*

```
n_31 = cross(V_3 - V_1,  V_4 - V_1);
if ( n_31*l < 0 ) draw(E_31);
n_12 = cross(V_1 - V_2,  V_5 - V_2);
if ( n_12*l < 0 ) draw(E_12);
n_23 = cross(V_2 - V_3,  V_6 - V_3);
if ( n_23*l < 0 ) draw(E_23);
```



Silhouettes on the Stanford bunny

Note: *This approach draws also hidden contour lines.*