Doubly Connect Edge List (DCEL)

Notes from the book by de Berg, Van Kreveld, Overmars, and Schwarzkpf.

pp. 29-39
Doubly Connected Edge List (DCEL)

• DCEL is one of the most commonly used representations for planar subdivisions such as Voronoi diagrams.

• It is an edge-based structure which links together the three sets of records:
  – Vertex
  – Edge
  – Face

• It facilitates traversing the faces of planar subdivision, visiting all the edges around a given vertex
Doubly Connected Edge List (DCEL)

- Record for each face, edge, and vertex
  - Geometric information
  - Topological information
  - Attribute information
- Half-edge structure
Doubly Connected Edge List (DCEL)

• Main ideas:
  – Edges are oriented counterclockwise inside each face
  – Since an edge borders two faces, each edge is replaced by two half-edges, one for each face
Doubly Connected Edge List (DCEL)

- The vertex record of a vertex \( v \) stores the coordinates of \( v \). It also stores a pointer \( \text{IncidentEdge}(v) \) to an arbitrary half-edge that has \( v \) as its origin.

- The face record of a face \( f \) stores a pointer to some half-edge on its boundary which can be used as a starting point to traverse \( f \) in counterclockwise order.

- The half-edge record of a half-edge \( e \) stores pointer to:
  - \( \text{Origin}(e) \)
  - \( \text{Twin of } e, \ e.\text{twin or twin}(e) \)
  - The face to its left ( \( \text{IncidentFace}(e) \) )
  - \( \text{Next}(e) : \) next half-edge on the boundary of \( \text{IncidentFace}(e) \)
  - \( \text{Previous}(e) : \) previous half-edge
**Doubly Connected Edge List (DCEL)**

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Coordinates</th>
<th>IncidentEdge</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>$(x_1, y_1)$</td>
<td>$e_{2,1}$</td>
</tr>
<tr>
<td>$v_2$</td>
<td>$(x_2, y_2)$</td>
<td>$e_{4,1}$</td>
</tr>
<tr>
<td>$v_3$</td>
<td>$(x_3, y_3)$</td>
<td>$e_{3,2}$</td>
</tr>
<tr>
<td>$v_4$</td>
<td>$(x_4, y_4)$</td>
<td>$e_{6,1}$</td>
</tr>
<tr>
<td>$v_5$</td>
<td>$(x_5, y_5)$</td>
<td>$e_{9,1}$</td>
</tr>
<tr>
<td>$v_6$</td>
<td>$(x_6, y_6)$</td>
<td>$e_{7,1}$</td>
</tr>
</tbody>
</table>
Doubly Connected Edge List (DCEL)

<table>
<thead>
<tr>
<th>Face</th>
<th>Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_1</td>
<td>e_{1,1}</td>
</tr>
<tr>
<td>f_2</td>
<td>e_{5,1}</td>
</tr>
<tr>
<td>f_3</td>
<td>e_{5,2}</td>
</tr>
<tr>
<td>f_4</td>
<td>e_{8,1}</td>
</tr>
<tr>
<td>f_5</td>
<td>e_{9,2}</td>
</tr>
</tbody>
</table>

\[ f_1, f_2, f_3, f_4, f_5 \]

\[ e_{1,1}, e_{5,1}, e_{5,2}, e_{8,1}, e_{9,2} \]
Doubly Connected Edge List (DCEL)

<table>
<thead>
<tr>
<th>Half-edge</th>
<th>Origin</th>
<th>Twin</th>
<th>IncidentFace</th>
<th>Next</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_{3,1}</td>
<td>v_2</td>
<td>e_{3,2}</td>
<td>f_1</td>
<td>e_{1,1}</td>
<td>e_{2,1}</td>
</tr>
<tr>
<td>e_{3,2}</td>
<td>v_3</td>
<td>e_{3,1}</td>
<td>f_2</td>
<td>e_{4,1}</td>
<td>e_{5,1}</td>
</tr>
<tr>
<td>e_{4,1}</td>
<td>v_2</td>
<td>e_{4,2}</td>
<td>f_2</td>
<td>e_{5,1}</td>
<td>e_{3,2}</td>
</tr>
<tr>
<td>e_{4,2}</td>
<td>v_4</td>
<td>e_{4,1}</td>
<td>f_5</td>
<td>e_{2,2}</td>
<td>e_{8,2}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Doubly Connected Edge List (DCEL)

- **Storage space requirement:**
  - Linear in the number of vertices, edges, and faces
Doubly Connected Edge List (DCEL)

- **Operations:**
  - Walk around the boundary of a given face in CCW order
  - Access a face from an adjacent one
  - Visit all the edges around a given vertex
Doubly Connected Edge List (DCEL)

• **Interesting Queries:**
  – Given a DCEL description, a line $L$ and a half-edge that this line cuts, efficiently find all the faces cut by $L$. 
Doubly Connected Edge List (DCEL)

- **Traversing face** $f$:
  - Given: an edge of $f$
    1. Determine the half-edge $e$ incident on $f$
    2. Start_edge $\leftarrow e$
    3. While next($e$) $\neq$ start_edge then
       $\quad e \leftarrow$ next ($e$)
Doubly Connected Edge List (DCEL)

- **Traversing all edges incident on a vertex** $v$
  - Note: we only output the half-edges whose origin is $v$
  - Given: a half-edge $e$ with the origin at $v$
    1. Start_edge $\leftarrow e$
    2. While next( twin(e) ) $\neq$ start_edge then
       e $\leftarrow$ next( twin(e) )
Adding a Vertex
Adding a Vertex

• New vertex x
• New edges: $e_{1,2}'$ and $e_{1,2}''$

• $\text{IncidentEdge}(x) = e_{1,2}'$

• $\text{Origin}(e_{1,2}') = x$
• $\text{Next}(e_{1,2}') = \text{next}(e_{1,2})$
• $\text{Prev}(e_{1,2}') = e_{1,2}''$
• $\text{IncidentFace}(e_{1,2}') = f_2$

• $\text{Origin}(e_{1,2}'') = \text{origin}(e_{1,2})$
• $\text{Next}(e_{1,2}'') = e_{1,2}'$
• $\text{Prev}(e_{1,2}'') = \text{prev}(e_{1,2})$
• $\text{IncidentFace}(e_{1,2}'') = f_2$

• $\text{Next(Prev}(e_{1,2})) = e_{1,2}''$
• $\text{Prev(Next}(e_{1,2})) = e_{1,2}'$

• Delete edge $e_{1,2}$
Adding a Vertex

• New edges: \(e_{1,1}'\) and \(e_{1,1}''\)

• \(\text{Origin}(e_{1,1}') = \text{origin}(e_{1,1})\)
• \(\text{Next}(e_{1,1}') = e_{1,1}''\)
• \(\text{Prev}(e_{1,1}') = \text{prev}(e_{1,1})\)
• \(\text{IncidentFace}(e_{1,1}') = f_1\)

• \(\text{Origin}(e_{1,1}'') = e_{1,1}'\)
• \(\text{Next}(e_{1,1}'') = \text{next}(e_{1,1})\)
• \(\text{Prev}(e_{1,1}'') = e_{1,1}'\)
• \(\text{IncidentFace}(e_{1,1}'') = f_1\)

• \(\text{Next}(\text{prev}(e_{1,1})) = e_{1,1}'\)
• \(\text{Prev}(\text{next}(e_{1,1})) = e_{1,1}''\)

• \(\text{Twin}(e_{1,2}') = e_{1,1}'\)
• \(\text{Twin}(e_{1,1}') = e_{1,2}'\)
• \(\text{Twin}(e_{1,2}'') = e_{1,1}''\)
• \(\text{Twin}(e_{1,1}'') = e_{1,2}''\)
• Delete edge \(e_{1,1}\)
Adding a Vertex

- If $e_{1,1}$ was starting edge of $f_1$, need to change it to either one of the new edges
- If $e_{1,2}$ was starting edge of $f_2$, need to change it to either one of the new edges
Other Operations on DCEL

• **Add an Edge**
  – Planar subdivision
  – e is added
  – DCEL can be updated in constant time once the edges a and b are known