Real-Time Rendering

Doubly Connected Edge List (*DCEL*)

WS 2019/20

Prof. Rhadamés Carmona

Universidad Central de Venezuela
Computer Graphics Center

Bauhaus-Universität Weimar
Virtual Reality and Visualization Research

Literature

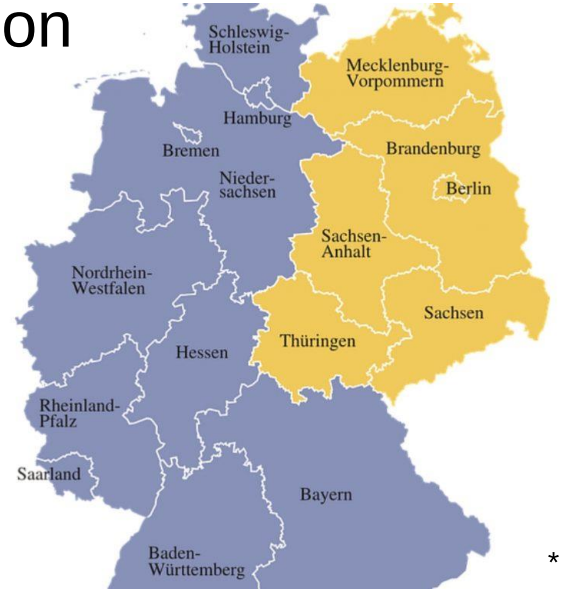
- Computational Geometry, Algorithms and Applications (3rd edition), Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars, Springer 2008
  Course follows partially this book
  Lecture design and slide set by M. v. Kreveld
- Inspired by further lectures
  - Bernd Fröhlich and Stephan Beck, Bauhaus Universität Weimar, Real Time Rendering 2019.
  - Pjotr Indyk, MIT
  - Thomas Ottmann, Freiburg
  - Bernd Gärtner, Michael Hoffmann, ETH
  - Prof. Stefan Schirra, Madgeburg

Motivation

- Networks, rivers, and some other map layers can be represented as collections of line segments.
- Layers representing labeled regions have a more complicated structure; e.g. a thematic map of vegetation
- For this kind of layers, storing a planar subdivision as a collection of line segments is not such a good idea
- Operations like reporting the boundary of a region, or accessing adjacent regions would be rather complicated

Motivation

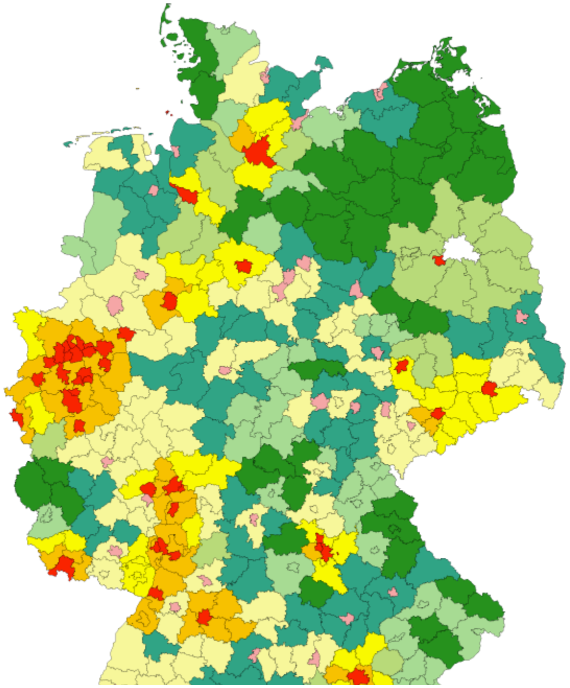
- We need to incorporate topological information
  - Which segments bound a given region
  - Visit all edges around a given vertex
  - Which regions are adjacent, and so on
- A simple data structure called Doubly Connected Edge List (DCEL) is commonly used to represent this kind of planar subdivision



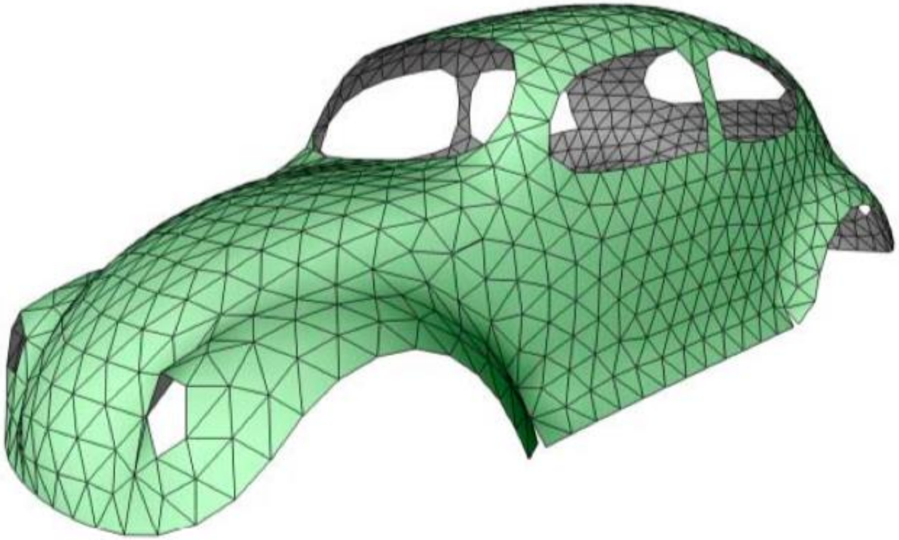
* from http://turismo.org/mapa-de-alemania/

General Subdivisions

- Subdivisions can be
  - in 2D plane (planar)
  - in 3D (mesh), also known as Boundary Representations (B-Rep)



2D planar subdivision (+)



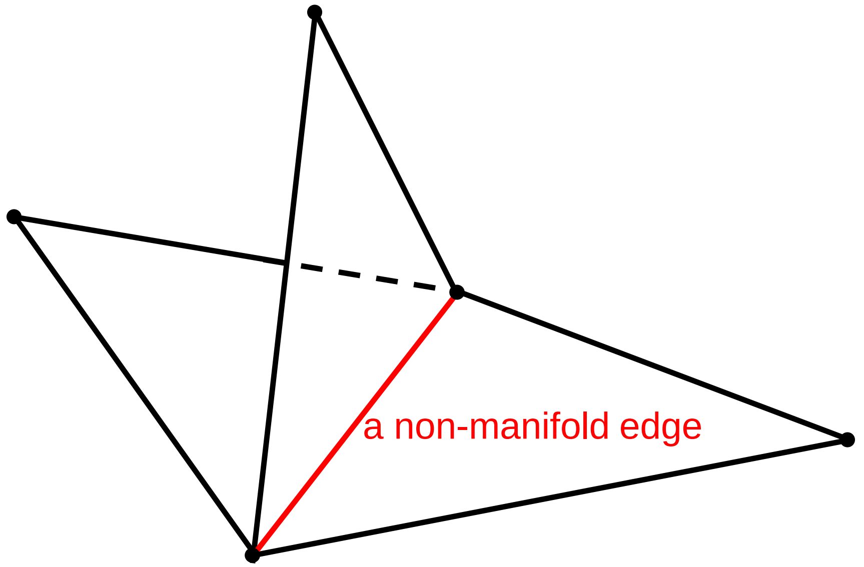
3D triangle mesh (*)

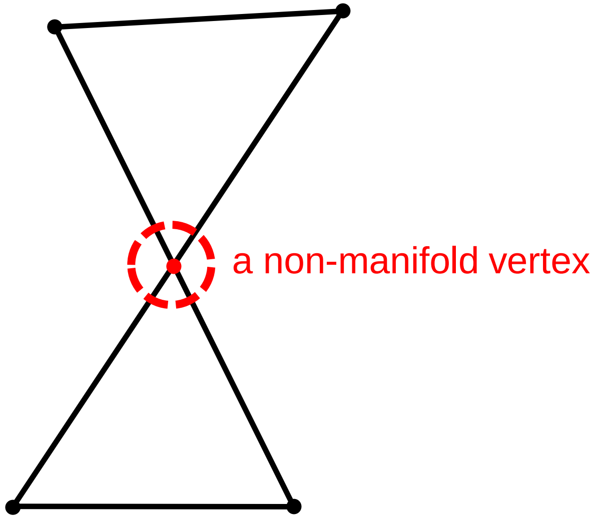
+ from M. Dörrbecker

* from http://www.pointclouds.org/blog/nvcs/martin/index.php

General Subdivisions

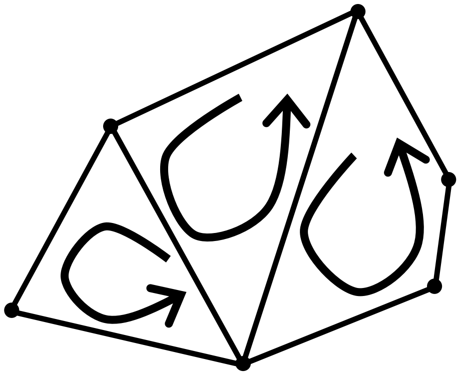
- A 3D mesh is **manifold** if:
  - every edge separates 2 faces
  - no three or more faces share an edge
  - faces are not connected by single vertex



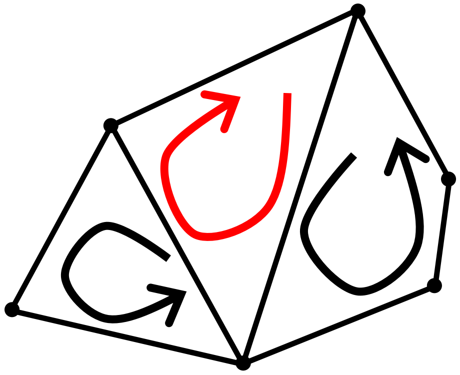


General Subdivisions

- A (2D/3D) mesh is **orientable** if**:**
    - orientation for all faces are consistent
    - inside or outside is defined by vertex ordering



orientable

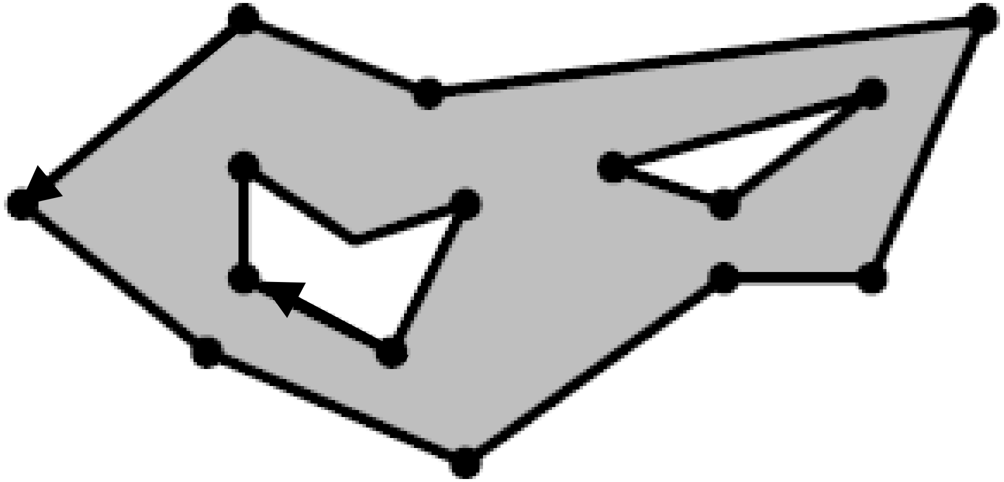


non-orientable

- We focus on **orientable and manifold** planar subdivisions

General Subdivisions

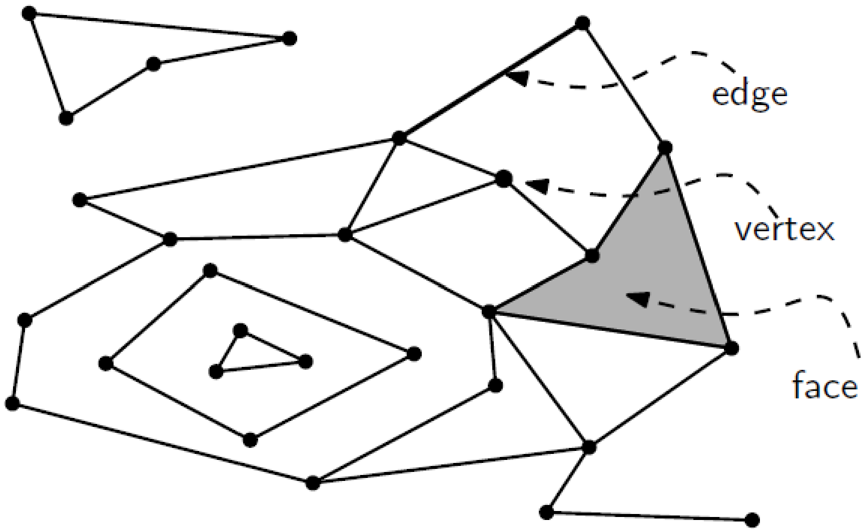
- Mesh polygons may contain holes

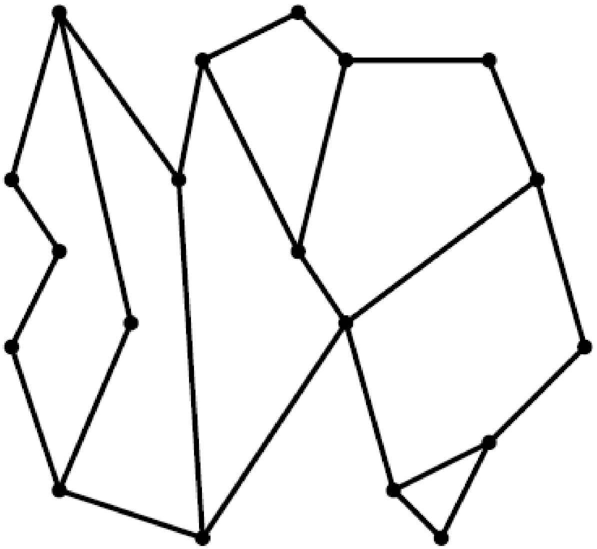


- Typically CCW is used for inside and CW for outside

Planar Subdivisions

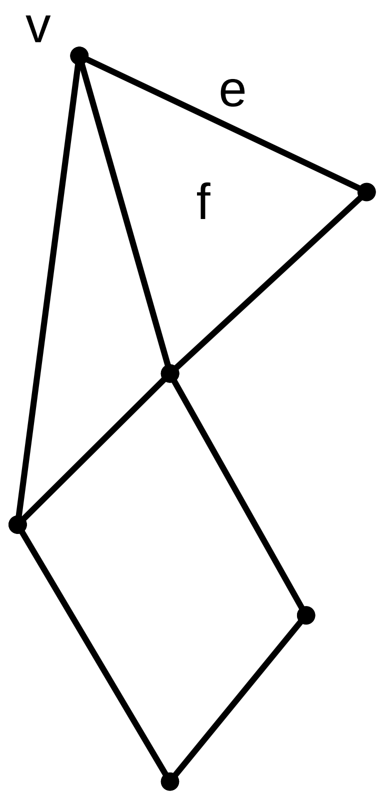
- A planar subdivision is a structure induced by a set of line segments in the plane
  - Line segments only intersect at common endpoints
  - Consists of vertices, edges and faces
- In computer graphics we often talk about polygon meshes instead of a planar subdivision – which is not exactly the same but close …





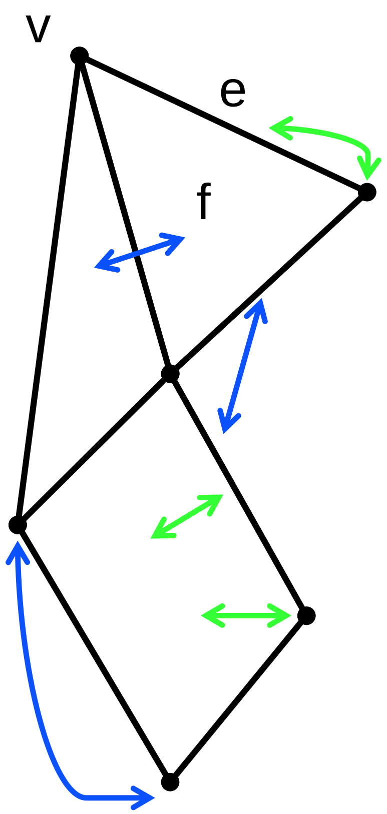
Representing Planar Subdivisions

- Vertices
  - Endpoints of line segments (nodes of the graph)
- Edge
  - Interiors of line segments (open interval, edges of a graph). It does not include endpoints.
- Faces
  - Interiors of connected two-dimensional regions that do not contain any point of any line segment. Open polygonal region whose boundary is formed by edges and vertices from the subdivision, but that boundary is not part of the face (open)



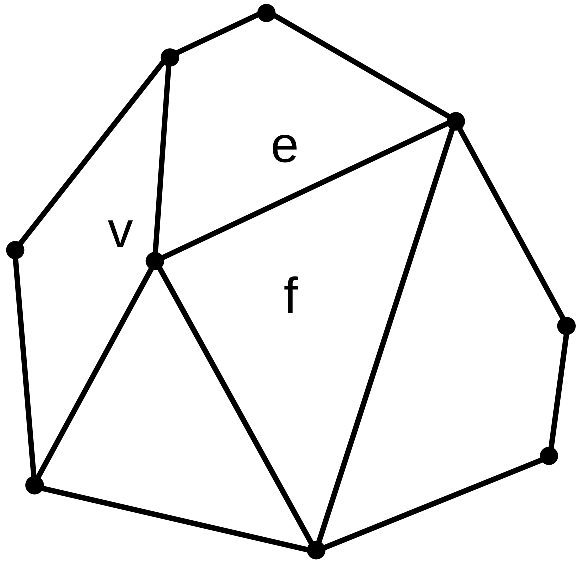
Representing Planar Subdivisions

- Objects of the same dimensionality are **adjacent** or not
- Objects of different dimensionality are **incident** or not



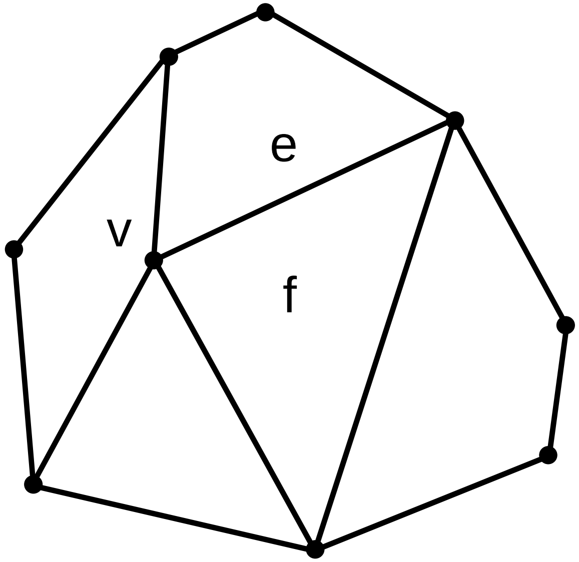
Planar Subdivisions

- We need a data structure to respond these useful queries:
  - Which faces use this vertex?
  - Which edges use this vertex?
  - Which faces share this edge?
  - Which edges border this face?
  - Which faces are adjacent to this face?



Planar Subdivisions

- We need a data structure to respond these useful queries:
  - Which faces use this vertex?
  - Which edges use this vertex?
  - Which faces share this edge?
  - Which edges border this face?
  - Which faces are adjacent to this face?

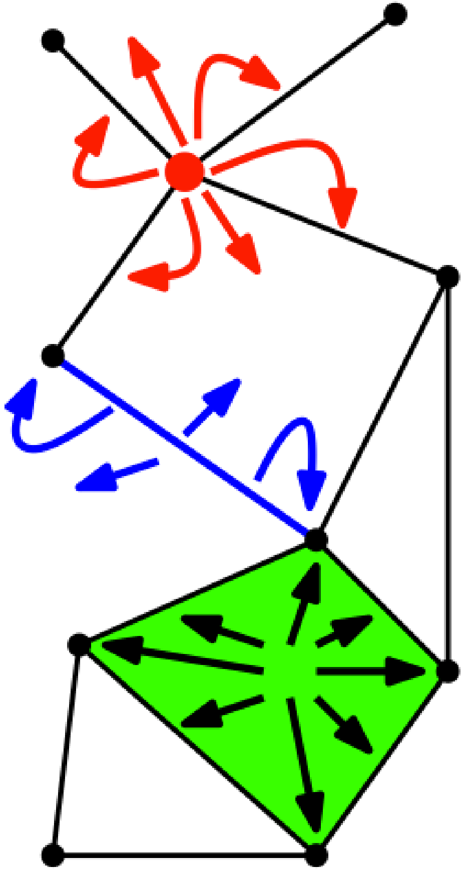


- We need a data structure that:
  - stores topology, geometry and attributes
  - incident and adjacent objects can be reached easily
  - enables traversal
  - with O(v+e+f) storage

Representing Planar Subdivisions

- A subdivision representation has:
  - a vertex class,
  - an edge class
  - a face class

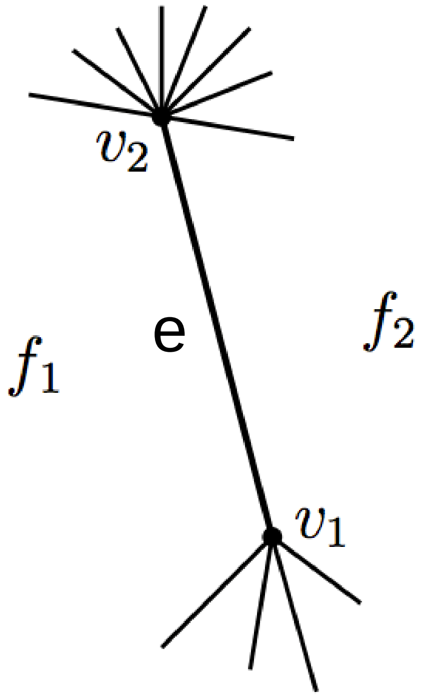
- It is a pointers-based structure where objects can reach incident (or adjacent) objects easily



Representing Planar Subdivisions

- Use the **edge** as the central object

- For any edge:
  - exactly two vertices are incident
  - one or two faces are incident
  - zero or more other edges are adjacent

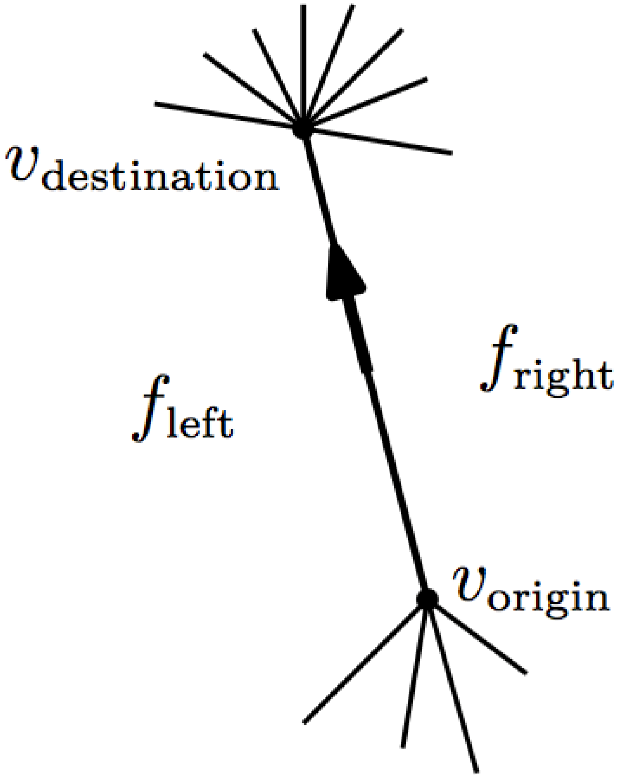


Representing Planar Subdivisions

- Use the **edge** as the central object

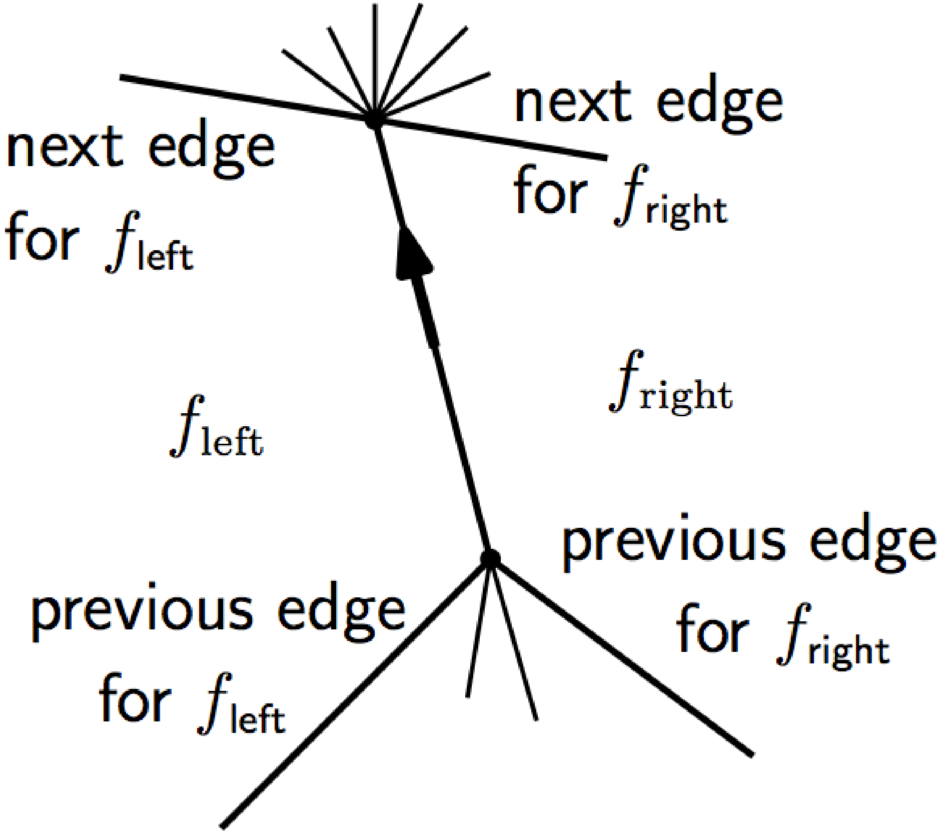
  ⇒ give it a direction!

- Any edge has:
  - an origin
  - a destination
  - a left face, and a right face



Representing Planar Subdivisions

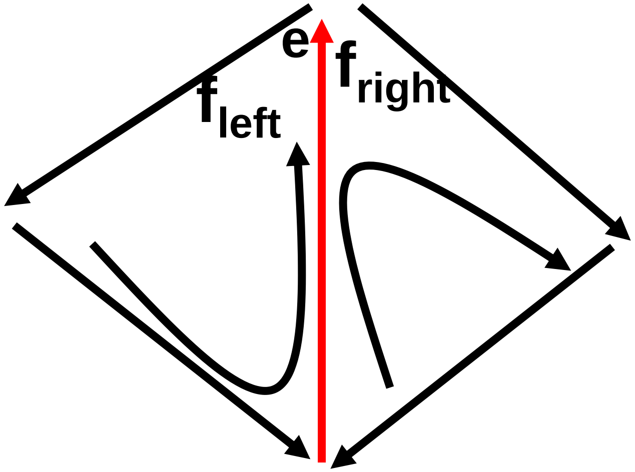
- **Four edges** are of special interest:


next edge for f_{left}
next edge for f_{right}
f_{left}
f_{right}
previous edge for f_{left}
previous edge for f_{right}

Representing Planar Subdivisions

- Each edge can be shared by 2 faces (**f_{left}** and **f_{right}**)

- It would be nice if we could traverse a **boundary cycle** in CCW order by continuously following the next edge for **f_{left}** or **f_{right}**



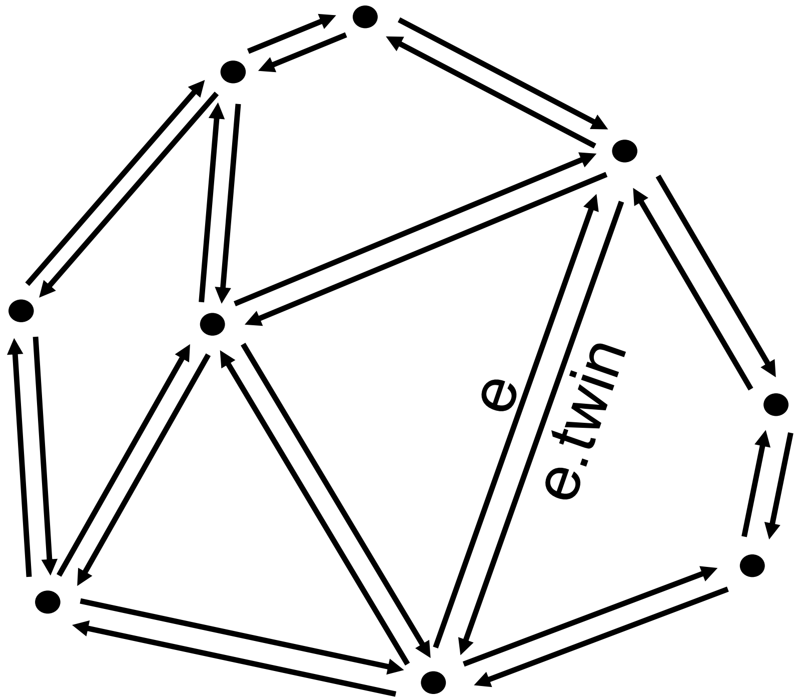
- ... but, the orientation needs to be consistent...

Representing Planar Subdivisions

- We apply a trick:

  split every edge into two **half-edges**

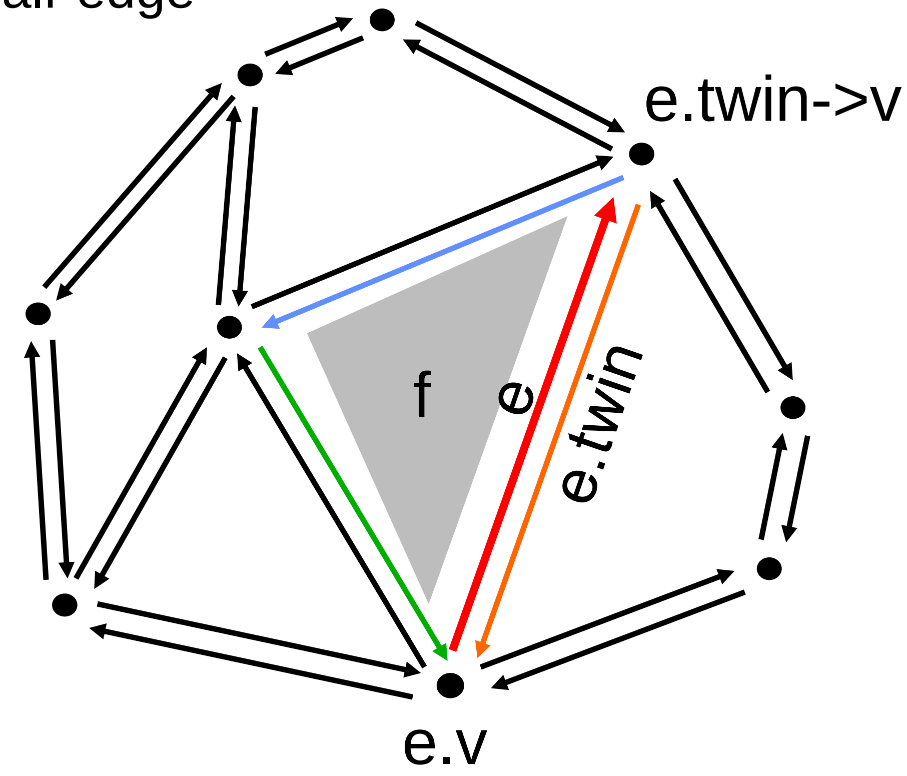
- Every shared **half-edge**:
  - has exactly one half-edge as twin
  - is directed opposite to its twin
  - is incident to only one face



Doubly-Connected Edge List (DCEL)

- **Half-edge** objects store pointers to:
  - incident starting vertex
  - adjacent next and previous half-edge
  - adjacent twin half-edge
  - incident left face

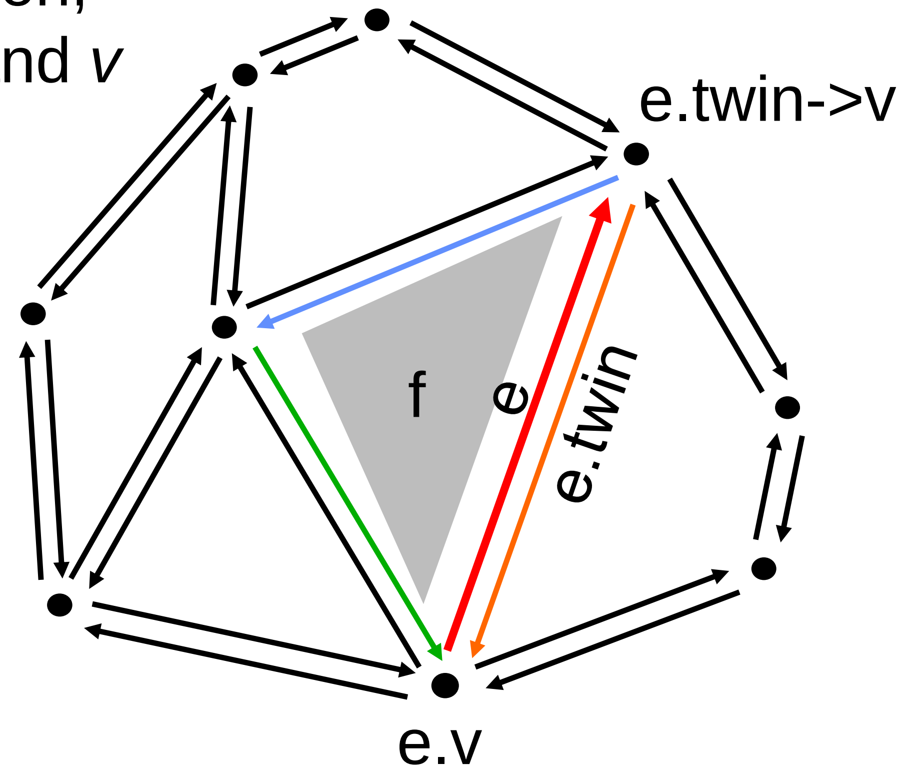
```
class h_edge {
    vertex* v;
    h_edge* twin;
    face*   f;

    h_edge* next;
    h_edge* prev;
};
```



Doubly-Connected Edge List (DCEL)

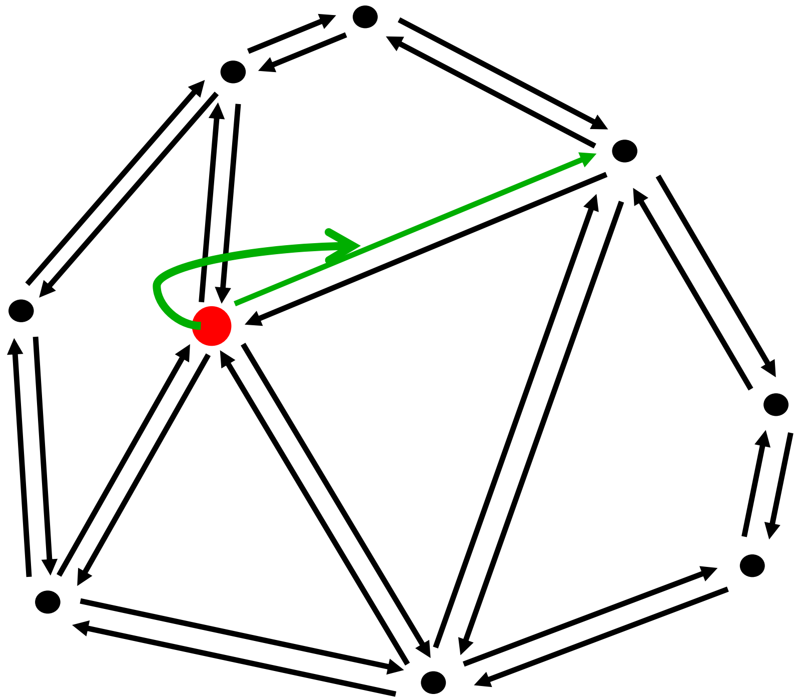
- **Half-edge** are oriented: *origin* → *destination*. If *e* has *v* as its origin and *w* as its destination, e.*Twin* has *w* as its origin and *v* as its destination.

```
class h_edge {
   vertex* v;
   h_edge* twin;
   face*   f;

   h_edge* next;
   h_edge* prev;
};
```



Doubly-Connected Edge List (DCEL)

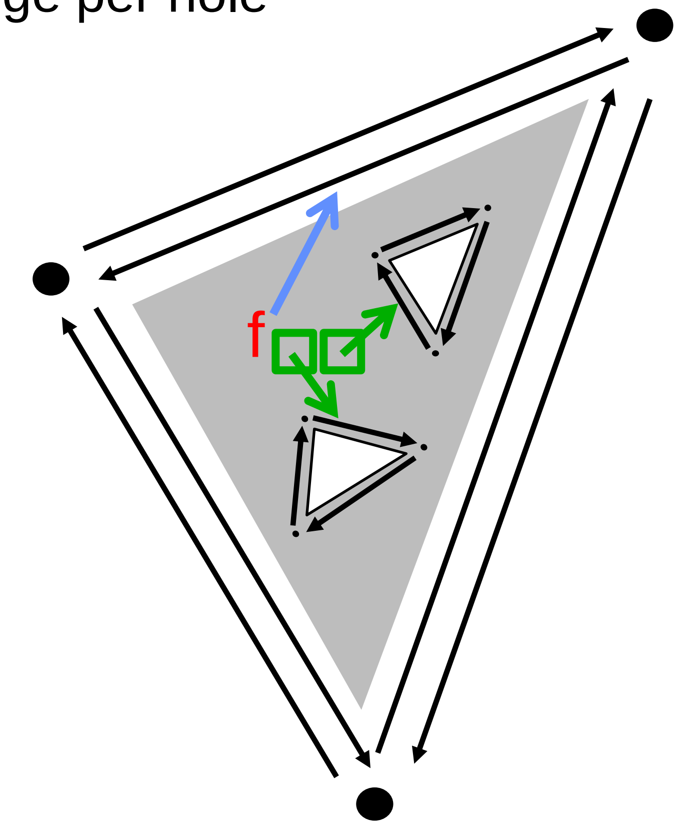
- **Vertex** objects store:
  - coordinates
  - pointer to one of the outgoing half edges



```
class vertex {
        point2(3)D  p;
        h_edge*  e;
};
```

Doubly-Connected Edge List (DCEL)

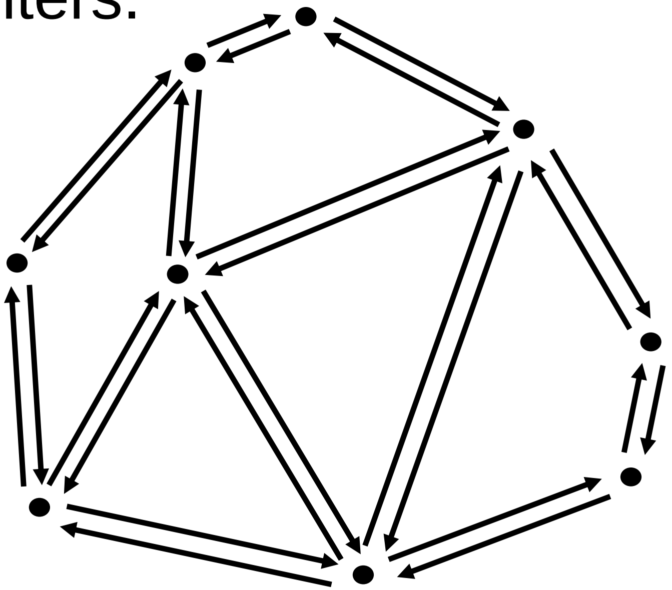
- **Face** objects store pointers to:
  - outer component (one half-edge of outer cycle)
  - Inner components: one half edge per hole



```
class face {
    h_edge* outer;
    list<h_edge*> inner;
}
```

Doubly-Connected Edge List (DCEL)

- A DCEL data structure is a collection of vertex, half-edge and face objects that are **connected correctly** by pointers.

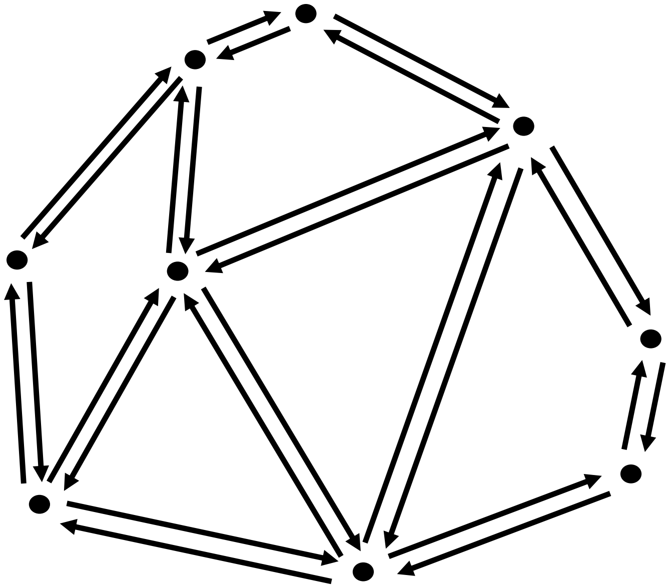


```
class vertex {
          point2(3)D  p;
          h_edge*   e;
}

class h_edge {
    vertex* v;
    h_edge* twin;
    face*   f;

    h_edge* next;
    h_edge* prev;
}
```

```
class face {
    h_edge* outer;
    list<h_edge*> inner;
}
```

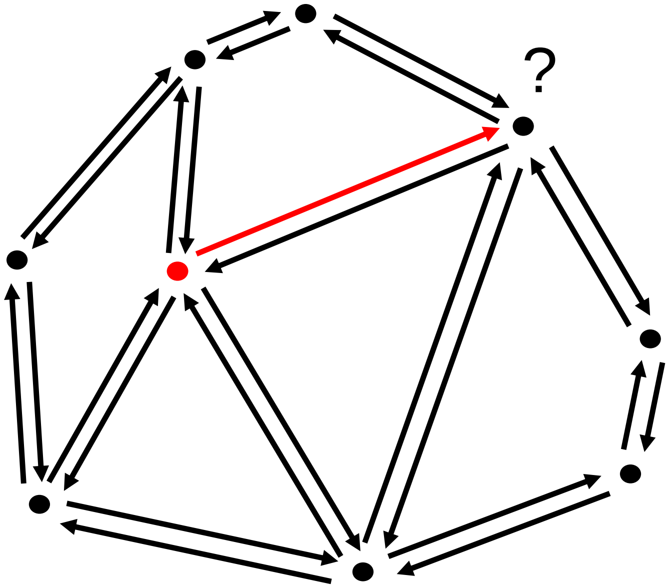
DCEL Supports Adjacency Queries

- Recap some questions:
  - Which is the destination vertex for a half-edge?
  - Which faces use this vertex?
  - Which edges use this vertex?
  - Which faces border this edge?
  - Which edges border this face?
  - Which are all adjacent faces of a face?



DCEL Supports Adjacency Queries

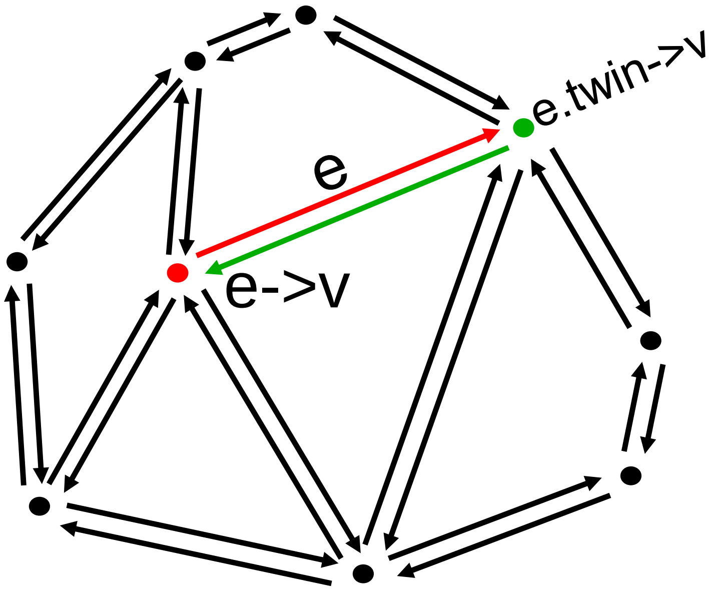
- Recap some questions:
  - Which is the destination vertex for a half-edge?
  - Which faces use this vertex?
  - Which edges use this vertex?
  - Which faces border this edge?
  - Which edges border this face?
  - Which are all adjacent faces of a face?



DCEL Supports Adjacency Queries

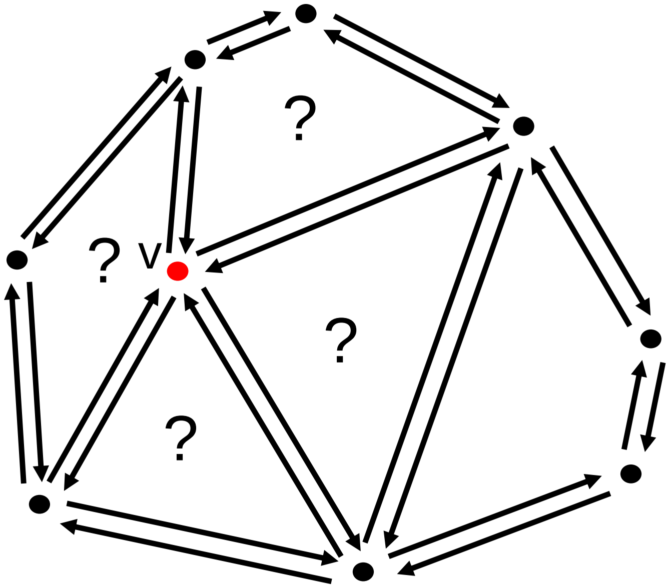
- Recap some questions:
  - Which is the destination vertex for a half-edge?

    Use pointer to twin half-edge
  - Which faces use this vertex?
  - Which edges use this vertex?
  - Which faces border this edge?
  - Which edges border this face?
  - Which are all adjacent faces of a face?



DCEL Supports Adjacency Queries

- Recap some questions:
  - Which is the destination vertex for a half-edge?
  - Which faces use this vertex?
  - Which edges use this vertex?
  - Which faces border this edge?
  - Which edges border this face?
  - Which are all adjacent faces of a face?

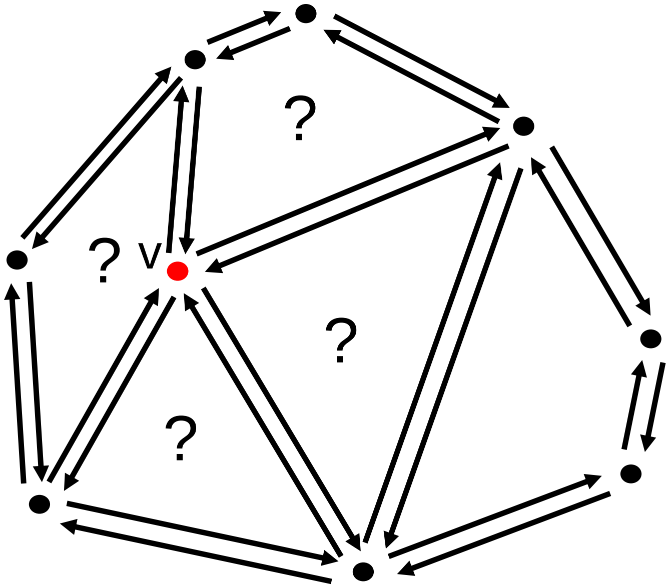


Bauhaus-Universität Weimar

DCEL Example Queries

- Which faces use this vertex?
  - Iterate over the outgoing and incoming half-edges of vertex v

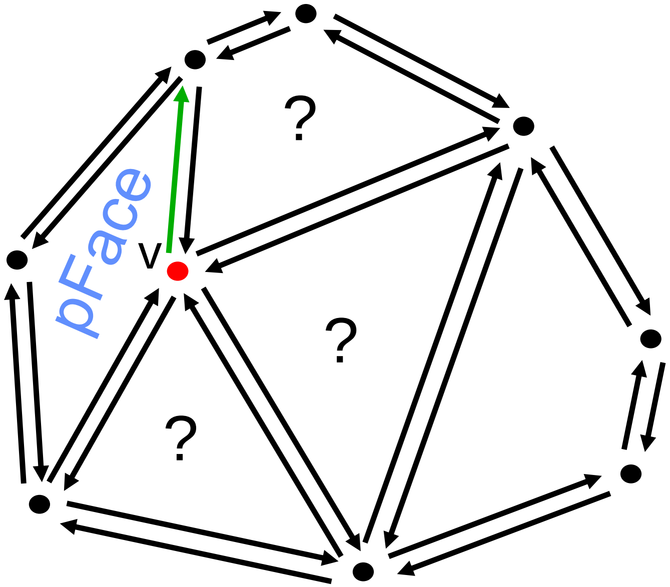
```
h_edge* pEdge = v.e;
do
{
   face* pFace = pEdge->f;
   if (pFace != NULL)
       report(pFace);
   pEdge = pEdge->twin->next;
} while (pEdge != v.e);
```



DCEL Example Queries

- Which faces use this vertex?
  - Iterate over the outgoing and incoming half-edges of vertex v

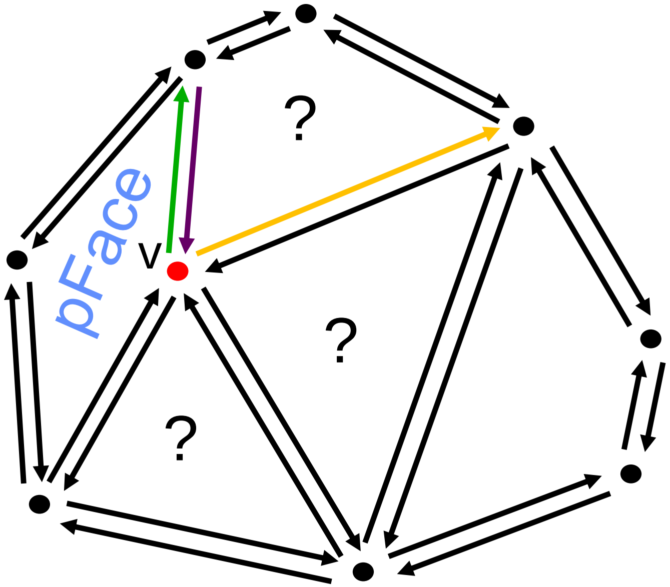
```
h_edge* pEdge = v.e;
do
{
   face* pFace = pEdge->f;
   if (pFace != NULL)
       report(pFace);
   pEdge = pEdge->twin->next;
} while (pEdge != v.e);
```



DCEL Example Queries

- Which faces use this vertex?
  - Iterate over the outgoing and incoming half-edges of vertex v

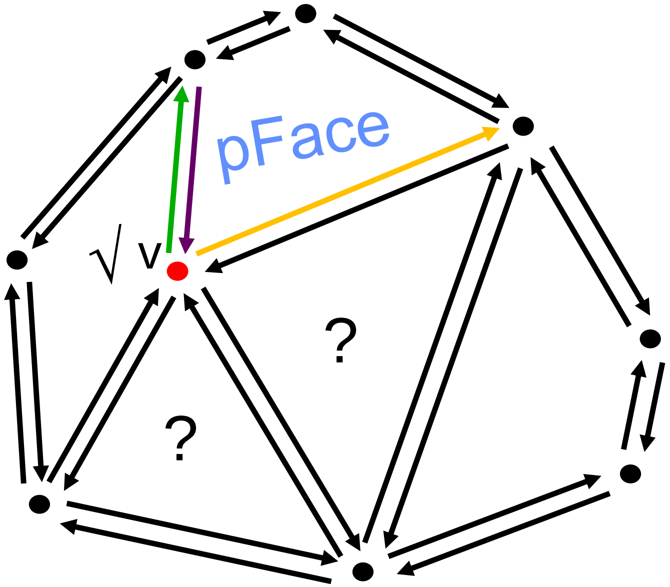
```
h_edge* pEdge = v.e;
do
{
   face* pFace = pEdge->f;
   if (pFace != NULL)
       report(pFace);
   pEdge = pEdge->twin->next;
} while (pEdge != v.e);
```



DCEL Example Queries

- Which faces use this vertex?
  - Iterate over the outgoing and incoming half-edges of vertex v

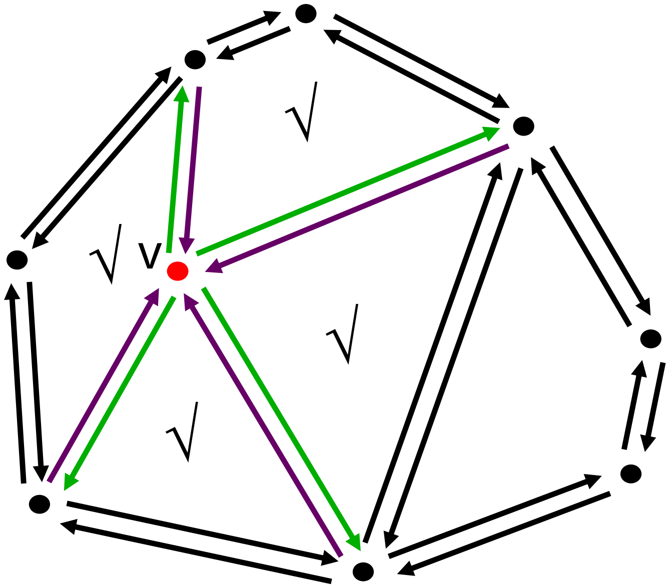
```
h_edge* pEdge = v.e;
do
{
   face* pFace = pEdge->f;
   if (pFace != NULL)
       report(pFace);
   pEdge = pEdge->twin->next;
} while (pEdge != v.e);
```



DCEL Example Queries

- Which faces use this vertex?
  - Iterate over the outgoing and incoming half-edges of vertex v

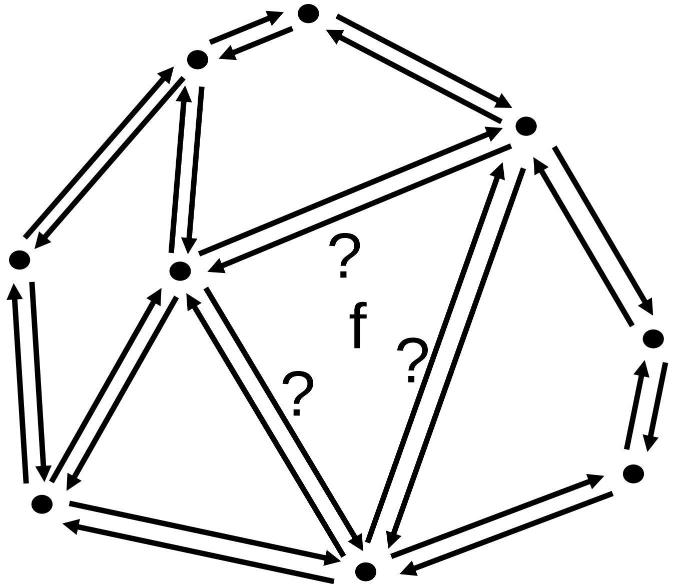
```
h_edge* pEdge = v.e;
do
{
   face* pFace = pEdge->f;
   if (pFace != NULL)
       report(pFace);
   pEdge = pEdge->twin->next;
} while (pEdge != v.e);
```



After 4 iterations v.e is reached and all faces around v were reported (√)

DCEL Supports Adjacency Queries

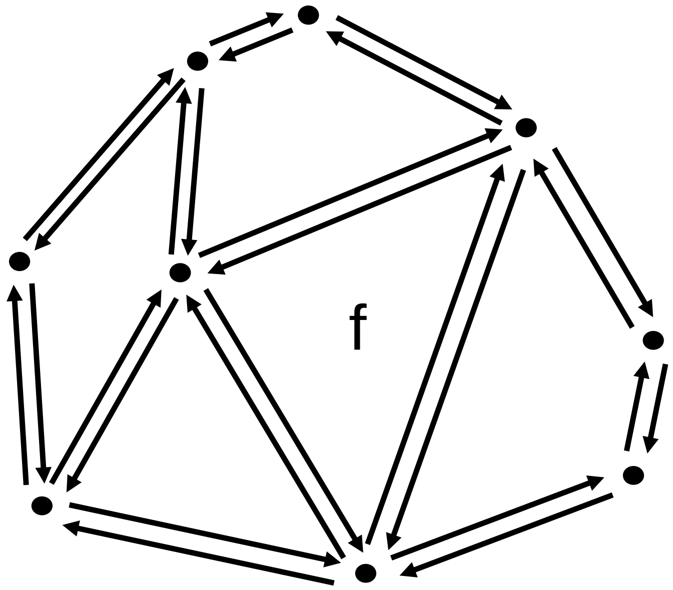
- Recap some questions:
  - Which is the destination vertex for a half-edge?
  - Which faces use this vertex?
  - Which edges use this vertex?
  - Which faces border this edge?
  - Which edges border this face?
  - Which are all adjacent faces of a face?



DCEL Example Queries

- Which edges border this face?
  - Iterating over the half-edges of a given face f

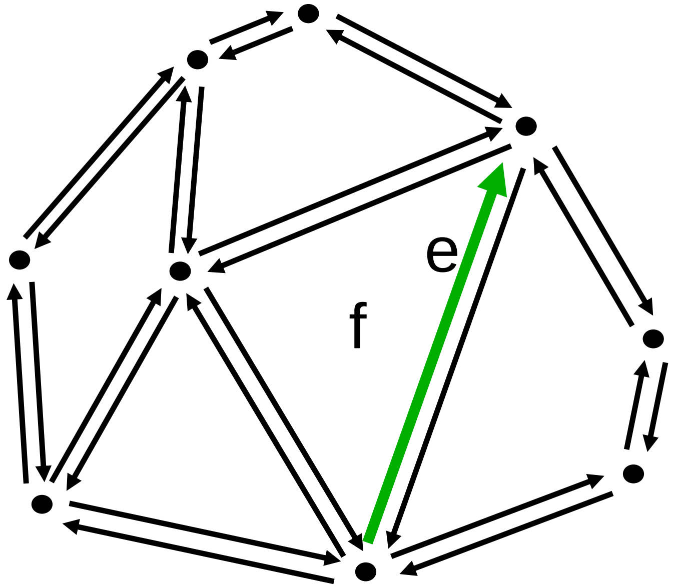
```
h_edge* e = f.outer;
do {
    Report(e);
    e = e->next;
} while (e!= f.outer);
```



DCEL Example Queries

- Which edges border this face?
  - Iterating over the half-edges of a given face f

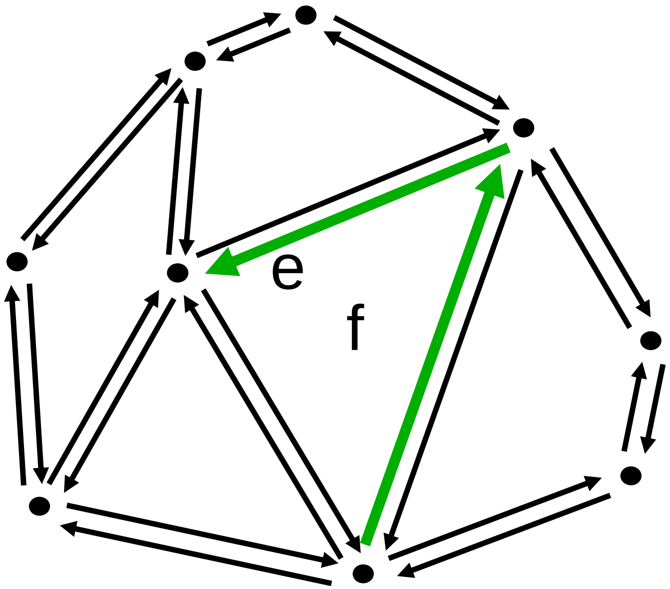
```
h_edge* e = f.outer;
do {
     Report(e);
     e = e->next;
} while (e!= f.outer);
```



DCEL Example Queries

- Which edges border this face?
  - Iterating over the half-edges of a given face f

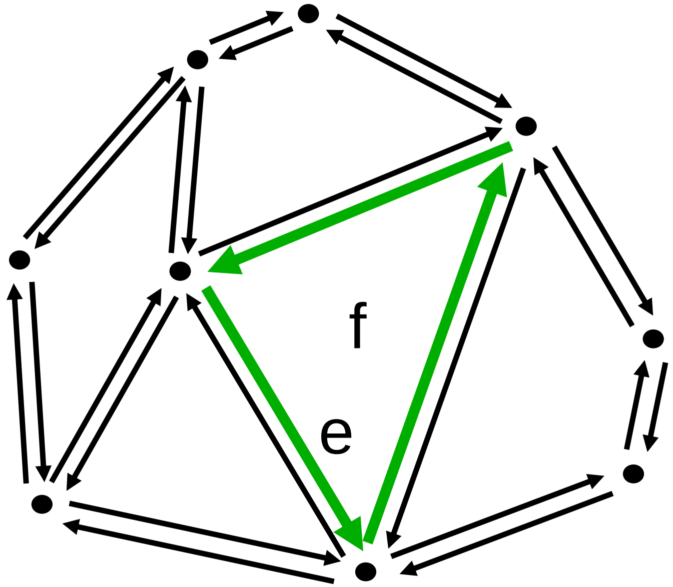
```
h_edge* e = f.outer;
do {
     Report(e);
     e = e->next;
} while (e!= f.outer);
```



DCEL Example Queries

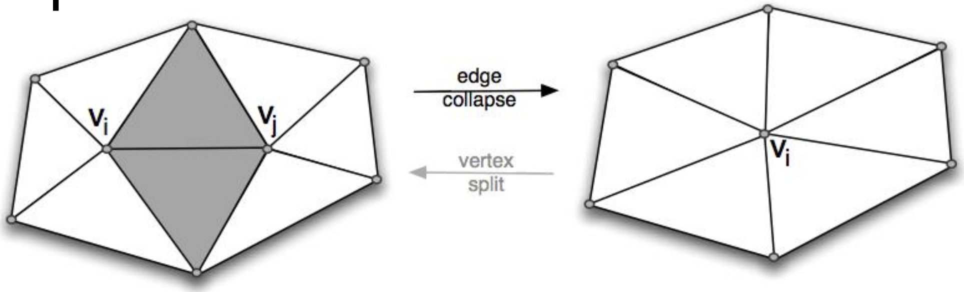
- Which edges border this face?
    - Iterating over the half-edges of a given face f

```
h_edge* e = f.outer;
do {
     Report(e);
     e = e->next;
} while (e!= f.outer);
```



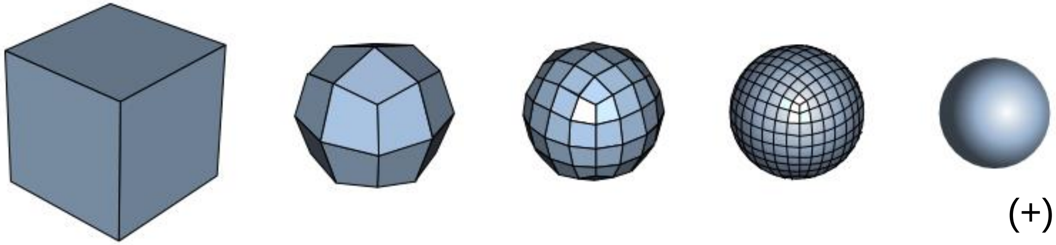
DCEL Applications

- Mesh processing (e.g. calculate normal per face, per vertex)
- Mesh simplification



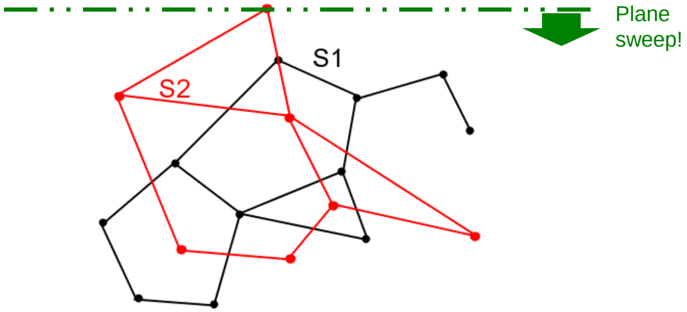
(*)

- Subdivision surfaces (Catmull-Clark)



(+)

- Map Overlay



+ from R. Behar

* from M. Reichl

Summary

- Doubly-Connected Edge List
    - Common data structure for planar subdivisions
    - Boundary representation (B-rep) of a planar subdivision
    - Forms a graph of connected vertices, half-edges and faces
    - Half-edge is the central component for traversal
    - Supports adjacency queries
    - Storage is O(e+v+f) (if no holes are present)
    - Many applications

End