Jürgen Ebert
University Koblenz-Landau
http://www.uni-koblenz.de/~ebert

POLYGONAL MODELS AND GRAPHS
An efficient data structure is proposed which integrates a geometric model with a graph-like semantics.
TABLE OF CONTENTS

- Introduction
- Graphs
- Subdivisions
- Integration
- Conclusion
- Introduction
- Graphs
- Subdivision
- Integration
- Conclusion
Discrete structures can be used as models for variety of application domains.

+ **trees** are models for many kinds of composite structures,
+ **graphs** are used to represent interconnected entities and their relations, and
+ **subdivisions** are used to model polygonal structures in the two-dimensional space.
To represent models in computer programs, efficient data structures are needed, that store the entities and their relationships in such a way that efficient retrieval and efficient traversal of data is supported.
All aspects pertaining to images
+ geometry,
+ topology,
+ appearance, and
+ semantics

have to be kept together in a smoothly integrated format which allows efficient treatment of all these aspects.
Introduction
Graphs
Subdivision
Integration
Conclusion
Graphs are a well known structure for discrete entities and their interrelations.
public static void main(String[] args) {
    int a = 26;
    int b = -5;
    System.out.println(compute(a, b));
}
EXAMPLE (SCHEMA)
TGRAPHS

TGraphs are

+ typed (supporting multiple inheritance):
  vertices and edges have a type
+ attributed (depending on the type):
  vertices and edges have valued attributes
+ ordered:
  vertices, edges, and incidences are ordered
+ directed:
  edges have a start and an end vertex
For working with graphs a powerful implementation is needed.

http://jgralab.uni-koblenz.de
The most important operation on large graphs is graph traversal. This is usually implemented using iterators.

```
1. VertexIterator getAllVertices ();
2. Edgelterator getAllEdges ();
3. IncEdgeIterator getAllIncEdges (Vertex g);
```

```
for (Edge e: getAllIncEdges (v)) {
    // process edge e
}
```
```java
void dfs (Vertex v) {
    v.setMark();
    // process vertex v
    for (Edge e: g.getAllOutEdges (v)) {
        w = e.omega();
        if (! w.isMarked()) {
            // process tree edge e
            dfs(w);
        }
    }
}
```
A edge e has two incident vertices:
+ e.this() is the vertex which supplied e and
+ e.that() is the other vertex.
void dfs (Vertex v) {
    v.setMark();
    // process vertex v
    for (Edge e: g.getAllIncEdges (v)) {
        w = e.theW();
        if (! w.isMarked()) {
            // process tree edge e
            dfs(w);
        }
    }
}
SYMMETRIC INCIDENCE LISTS

The implementation is efficient wrt traversal
PROPERTIES OF SIL

- Traversal of incidences of vertices in $O(\text{degree}(v))$
- Start and end vertex of edges in $O(1)$
- Edges as first class objects
- Typed and attributed vertex and edges objects retrievable in $O(1)$
- Simultaneous implementation of directed and underlying undirected graph
- Introduction
- Graphs
- Subdivisions
- Integration
- Conclusion
Polygonal structures in the plane consist of
+ a set of vertices,
+ a set of straight edges, and
+ a set of faces each of which is enclosed by edges.

Polygonal structures are used to approximate subdivisions and sometimes support actions like rendering or point location.
Voronoi diagrams and Delaunay triangulations are examples of such structures.
Subdivisions consist of vertices, edges and faces.

Geometrically they form a partition of the plane.
+ Vertices are points.
+ Edges are straight open lines.
+ Faces are open regions.
Planar graphs have a unique dual graph.
PROGRAMMING INTERFACE

```java
VerticesIterator getAllVertices();
EdgesIterator getAllEdges();
FacesIterator getAllFaces();

IncEdgesIteratorV getAllIncEdges(Vertex v);
IncEdgesIteratorF getAllIncEdges(Face f);

IncVerticesIterator getAllIncVertices(Face f);
IncFacesIterator getAllIncEdges(Vertex v);
```
# Doubly-Connected Edges Lists

## Vertices

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Coordinates</th>
<th>Incident Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>(0, 4)</td>
<td>$\vec{e}_{1,1}$</td>
</tr>
<tr>
<td>$v_2$</td>
<td>(2, 4)</td>
<td>$\vec{e}_{4,2}$</td>
</tr>
<tr>
<td>$v_3$</td>
<td>(2, 2)</td>
<td>$\vec{e}_{2,1}$</td>
</tr>
<tr>
<td>$v_4$</td>
<td>(1, 1)</td>
<td>$\vec{e}_{2,2}$</td>
</tr>
</tbody>
</table>

## Faces and Components

<table>
<thead>
<tr>
<th>Face</th>
<th>Outer Component</th>
<th>Inner Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>nil</td>
<td>$\vec{e}_{1,1}$</td>
</tr>
<tr>
<td>$f_2$</td>
<td>$\vec{e}_{4,1}$</td>
<td>nil</td>
</tr>
</tbody>
</table>

## Half-Edges

<table>
<thead>
<tr>
<th>Half-edge</th>
<th>Origin</th>
<th>Twin</th>
<th>Incident Face</th>
<th>Next</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\vec{e}_{1,1}$</td>
<td>$v_1$</td>
<td>$\vec{e}_{1,2}$</td>
<td>$f_1$</td>
<td>$\vec{e}_{4,2}$</td>
<td>$\vec{e}_{3,1}$</td>
</tr>
<tr>
<td>$\vec{e}_{1,2}$</td>
<td>$v_2$</td>
<td>$\vec{e}_{1,1}$</td>
<td>$f_2$</td>
<td>$\vec{e}_{3,2}$</td>
<td>$\vec{e}_{4,1}$</td>
</tr>
<tr>
<td>$\vec{e}_{2,1}$</td>
<td>$v_3$</td>
<td>$\vec{e}_{2,2}$</td>
<td>$f_1$</td>
<td>$\vec{e}_{2,2}$</td>
<td>$\vec{e}_{4,2}$</td>
</tr>
<tr>
<td>$\vec{e}_{2,2}$</td>
<td>$v_4$</td>
<td>$\vec{e}_{2,1}$</td>
<td>$f_1$</td>
<td>$\vec{e}_{3,1}$</td>
<td>$\vec{e}_{2,1}$</td>
</tr>
<tr>
<td>$\vec{e}_{3,1}$</td>
<td>$v_3$</td>
<td>$\vec{e}_{3,2}$</td>
<td>$f_1$</td>
<td>$\vec{e}_{1,1}$</td>
<td>$\vec{e}_{2,2}$</td>
</tr>
<tr>
<td>$\vec{e}_{3,2}$</td>
<td>$v_1$</td>
<td>$\vec{e}_{3,1}$</td>
<td>$f_2$</td>
<td>$\vec{e}_{4,1}$</td>
<td>$\vec{e}_{1,2}$</td>
</tr>
<tr>
<td>$\vec{e}_{4,1}$</td>
<td>$v_3$</td>
<td>$\vec{e}_{4,2}$</td>
<td>$f_2$</td>
<td>$\vec{e}_{1,2}$</td>
<td>$\vec{e}_{3,2}$</td>
</tr>
<tr>
<td>$\vec{e}_{4,2}$</td>
<td>$v_2$</td>
<td>$\vec{e}_{4,1}$</td>
<td>$f_1$</td>
<td>$\vec{e}_{2,1}$</td>
<td>$\vec{e}_{1,1}$</td>
</tr>
</tbody>
</table>
Introduction

Graphs

Subdivision

Integration

Conclusion
The subdivision and its two graphs, the point graph and the face graph (dual graph), should be accessible efficiently.
REPRESENTATION (DSIL)

\[
\begin{array}{cccccc}
0 & 5 & 4 & 1 & & \\
\hline
0 & 0 & -4 & -1 & -3 & 0 & 2 & -5 & 0 & 3 & -2 & & \text{fFirst} \\
1 & 2 & 2 & 3 & 3 & 0 & 1 & 1 & 1 & 1 & 3 & & \text{fNext} \\
1 & 4 & 5 & 2 & 1 & 0 & 2 & 3 & 3 & 4 & 5 & 3 & \text{fThat} \\
0 & 0 & 4 & -5 & 2 & 0 & 5 & 0 & -4 & 0 & 0 & & \text{vNext} \\
0 & 1 & -1 & -2 & -3 & 3 & & \text{vFirst} \\
\end{array}
\]
PROPERTIES OF DSIL

- Storage of vertex graph and face graph
- Traversal of incidences in $O(\text{degree}(v))$
- Start and end vertex of edges in $O(1)$
- Left and right face of edges in $O(1)$
- Edges and Faces as first class objects
- Typed and attributed vertex, edge and face objects retrievable in $O(1)$
- Simultaneous implementation of directed and underlying undirected graphs
FURTHER REMARKS

- For non-planar graphs, ‘faces’ correspond to (ordered) hyperedges.
- Additional subgraphs (like composite vertices, composite faces, or search trees) can be added.
Introduction
Graphs
Subdivision
Integration
Conclusion
An efficient data structure was proposed. It
+ stores a subdivision with its two graphs (topology),
+ allows typing of vertices, edges, and faces by a
schema (light-weight semantics), and
+ supports attribution of data to vertices, edges, and
faces (geometry, appearance).