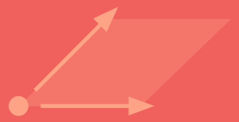
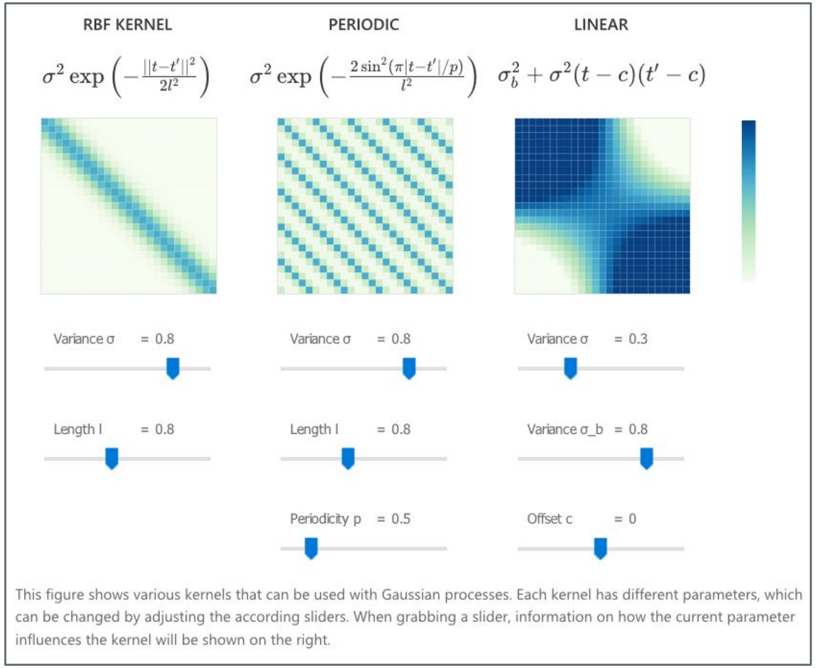


Interactive Explainers for Geometry Processing Algorithms

Jerry Yin and Jeffrey Goh

What are interactive explainers?

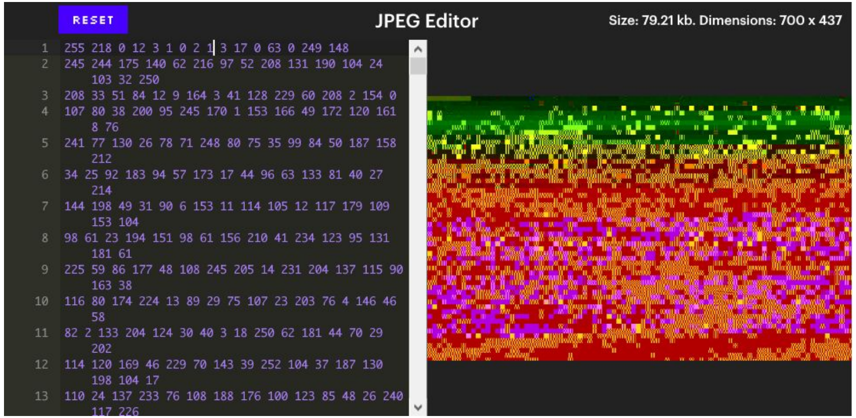
- Recent trend of interactive online articles on technical subjects
- Interactive visualizations and animations replace traditional static figures
- Online journals *Distill* (for ML) and *Parametric Press* are examples
- Made possible by modern web technologies



Interactive figure from "A Visual Exploration of Gaussian Processes" by Görtler et al, published in *Distill*.

What are interactive explainers?

- Recent trend of interactive online articles on technical subjects
- Interactive visualizations and animations replace traditional static figures
- Online journals *Distill* (for ML) and *Parametric Press* are examples
- Made possible by modern web technologies



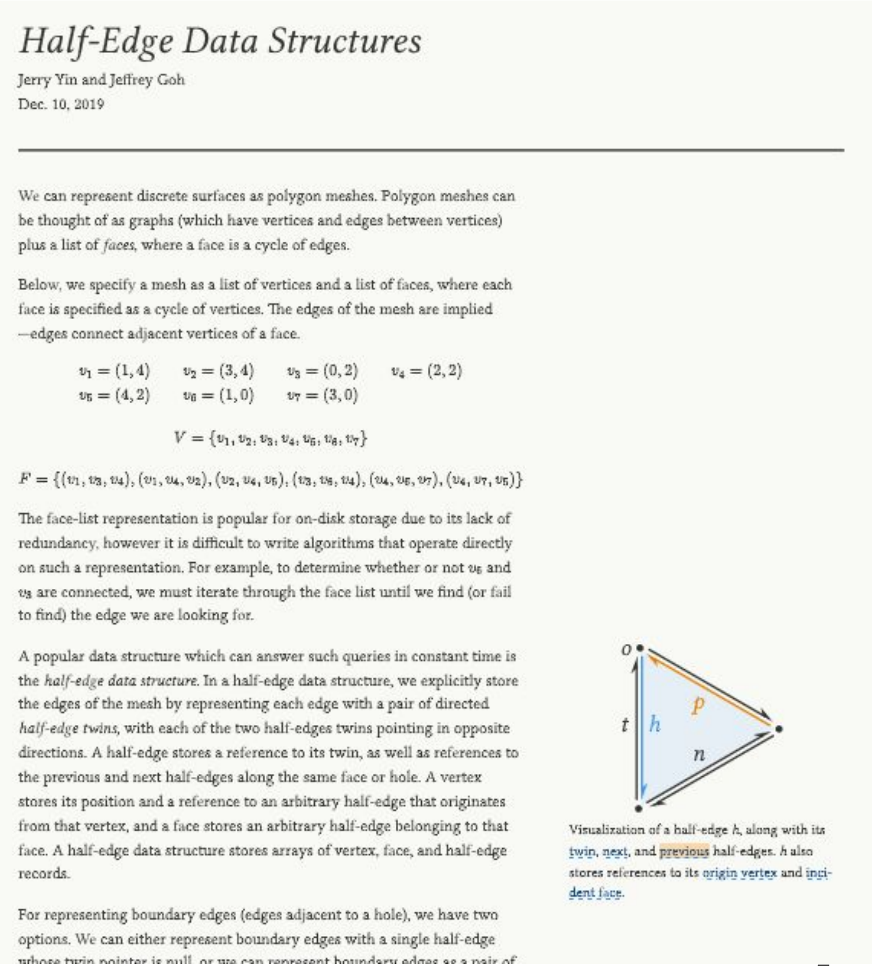
Interactive figure from “Unraveling the JPEG” by Omar Shehata, published in *Parametric Press*.

Why geometry?

- Geometry is a visual subject; creating visualizations for it makes sense.
- The undergraduate geometric modeling course (CPSC 424) could benefit from nicer notes.
- We created an article on *half-edge data structures*.

Implementation

- Used the Idyll markup language to create explainer
- Write article in Markdown-like syntax, and implement visualizations as React components in JavaScript
- Used D3 for visualizations
- Idyll converts markup to single HTML page



Half-Edge Data Structures

Jerry Yin and Jeffrey Goh

Dec. 10, 2019

We can represent discrete surfaces as polygon meshes. Polygon meshes can be thought of as graphs (which have vertices and edges between vertices) plus a list of *faces*, where a face is a cycle of edges.

Below, we specify a mesh as a list of vertices and a list of faces, where each face is specified as a cycle of vertices. The edges of the mesh are implied —edges connect adjacent vertices of a face.

$$v_1 = (1,4) \quad v_2 = (3,4) \quad v_3 = (0,2) \quad v_4 = (2,2)$$
$$v_5 = (4,2) \quad v_6 = (1,0) \quad v_7 = (3,0)$$

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$$

$$F = \{(v_1, v_3, v_4), (v_1, v_4, v_2), (v_2, v_4, v_5), (v_3, v_6, v_4), (v_4, v_6, v_7), (v_4, v_7, v_5)\}$$

The face-list representation is popular for on-disk storage due to its lack of redundancy, however it is difficult to write algorithms that operate directly on such a representation. For example, to determine whether or not v_5 and v_3 are connected, we must iterate through the face list until we find (or fail to find) the edge we are looking for.

A popular data structure which can answer such queries in constant time is the *half-edge data structure*. In a half-edge data structure, we explicitly store the edges of the mesh by representing each edge with a pair of directed *half-edge twins*, with each of the two half-edges twins pointing in opposite directions. A half-edge stores a reference to its twin, as well as references to the previous and next half-edges along the same face or hole. A vertex stores its position and a reference to an arbitrary half-edge that originates from that vertex, and a face stores an arbitrary half-edge belonging to that face. A half-edge data structure stores arrays of vertex, face, and half-edge records.

Visualization of a half-edge *h*, along with its twin, next, and previous half-edges. *h* also stores references to its origin vertex and incident face.

For representing boundary edges (edges adjacent to a hole), we have two options. We can either represent boundary edges with a single half-edge

Meshes

- Meshes are graphs with vertices and edges, plus a set of faces.
- Each face is a cycle of vertices.
- Representing faces as a set of cycles is compact (good for storage) but bad for mesh algorithms.
    - Asking questions like "are v_3 and v_5 connected?" requires searching through all the faces!

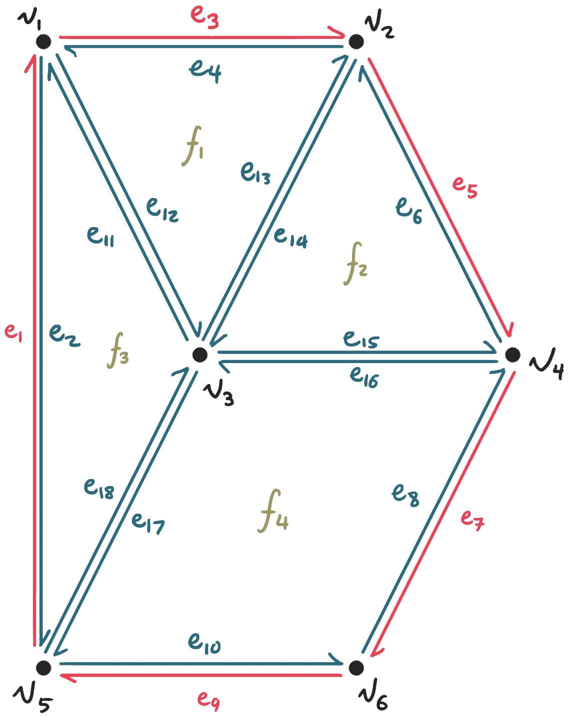
$$v_1 = (1,4) \quad v_2 = (3,4) \quad v_3 = (2,2)$$
$$v_4 = (4,2) \quad v_5 = (1,0) \quad v_6 = (3,0)$$

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$$

$$F = \{(v_1, v_3, v_2), (v_2, v_3, v_4), (v_1, v_5, v_3), (v_3, v_5, v_6, v_4)\}$$

Half-edge data structures

- Represent each edge as a pair of *half-edges*, each going in opposite directions.
- Each face is represented by a counter-clockwise cycle of half-edges.
- Boundary is represented by a clockwise cycle of half-edges.
- Each half-edge stores next and previous half-edges, its twin, its origin vertex, and its corresponding face.
    - Can answer most common queries in ~constant time.



$v_1 = (1,4)$ $v_2 = (3,4)$ $v_3 = (2,2)$

$v_4 = (4,2)$ $v_5 = (1,0)$ $v_6 = (3,0)$

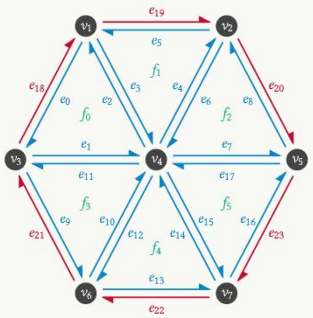
$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$$

$$F = \{(v_1, v_3, v_2), (v_2, v_3, v_4), (v_1, v_5, v_3), (v_3, v_5, v_6, v_4)\}$$

Half-edge vis

- Shows high-level (diagram) and low-level (records table) representations of a mesh
- Students need to understand both: think about algorithms at high level, but implement algorithms at the low level

```
# Enter your mesh definition in OBJ format below...
v 1.0 4.0 0.0
v 3.0 4.0 0.0
v 0.0 2.0 0.0
v 2.0 2.0 0.0
v 4.0 2.0 0.0
v 1.0 0.0 0.0
v 3.0 0.0 0.0
f 1 3 4
f 1 4 2
f 2 4 5
f 3 6 4
f 4 6 7
f 4 7 5
```



RECORDS

☐ Pin diagram to view

Vertex	Coordinate	Incident edge
v_1	(1, 4, 0)	e_0
v_2	(3, 4, 0)	e_5
v_3	(0, 2, 0)	e_1
v_4	(2, 2, 0)	e_2
v_5	(4, 2, 0)	e_8
v_6	(1, 0, 0)	e_{10}
v_7	(3, 0, 0)	e_{14}

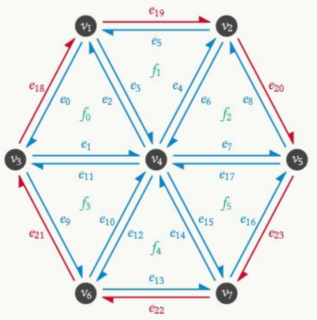
Face	Half-edge
f_0	e_0
f_1	e_3
f_2	e_6
f_3	e_9
f_4	e_{12}
f_5	e_{15}

Half-edge	Origin	Twin	Incident face	Next	Prev
e_0	v_1	e_{18}	f_0	e_1	e_2
e_1	v_3	e_{11}	f_0	e_2	e_0

Half-edge vis

- OBJ Editor view allows user to edit a mesh defined in the popular OBJ format.
    - Specify positions and connectivity
- Visual view shows a half-edge diagram.
    - Colour encodes boundary / interior half-edge

```
# Enter your mesh definition in OBJ format below...
v 1.0 4.0 0.0
v 3.0 4.0 0.0
v 0.0 2.0 0.0
v 2.0 2.0 0.0
v 4.0 2.0 0.0
v 1.0 0.0 0.0
v 3.0 0.0 0.0
f 1 3 4
f 1 4 2
f 2 4 5
f 3 6 4
f 4 6 7
f 4 7 5
```



RECORDS

☐ Pin diagram to view

Vertex	Coordinate	Incident edge
v_1	(1, 4, 0)	e_0
v_2	(3, 4, 0)	e_5
v_3	(0, 2, 0)	e_1
v_4	(2, 2, 0)	e_2
v_5	(4, 2, 0)	e_8
v_6	(1, 0, 0)	e_{10}
v_7	(3, 0, 0)	e_{14}

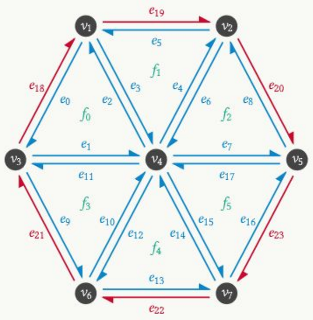
Face	Half-edge
f_0	e_0
f_1	e_3
f_2	e_6
f_3	e_9
f_4	e_{12}
f_5	e_{15}

Half-edge	Origin	Twin	Incident face	Next	Prev
e_0	v_1	e_{18}	f_0	e_1	e_2
e_1	v_3	e_{11}	f_0	e_2	e_0

Half-edge vis

- Records layout view shows all the records stored in the data structure.
    - Colours are the same as in the half-edge diagram.
- Linked highlighting

```
# Enter your mesh definition in OBJ format below...
v 1.0 4.0 0.0
v 3.0 4.0 0.0
v 0.0 2.0 0.0
v 2.0 2.0 0.0
v 4.0 2.0 0.0
v 1.0 0.0 0.0
v 3.0 0.0 0.0
f 1 3 4
f 1 4 2
f 2 4 5
f 3 6 4
f 4 6 7
f 4 7 5
```



RECORDS

☐ Pin diagram to view

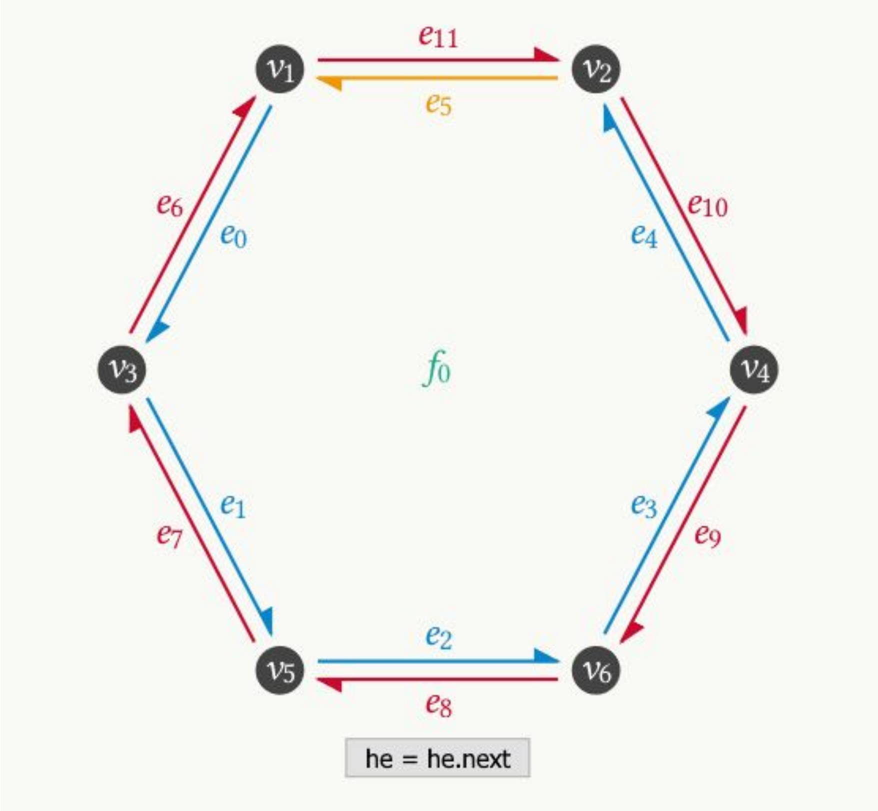
Vertex	Coordinate	Incident edge
v_1	(1, 4, 0)	e_0
v_2	(3, 4, 0)	e_5
v_3	(0, 2, 0)	e_1
v_4	(2, 2, 0)	e_2
v_5	(4, 2, 0)	e_8
v_6	(1, 0, 0)	e_{10}
v_7	(3, 0, 0)	e_{14}

Face	Half-edge
f_0	e_0
f_1	e_3
f_2	e_6
f_3	e_9
f_4	e_{12}
f_5	e_{15}

Half-edge	Origin	Twin	Incident face	Next	Prev
e_0	v_1	e_{18}	f_0	e_1	e_2
e_1	v_3	e_{11}	f_0	e_2	e_0

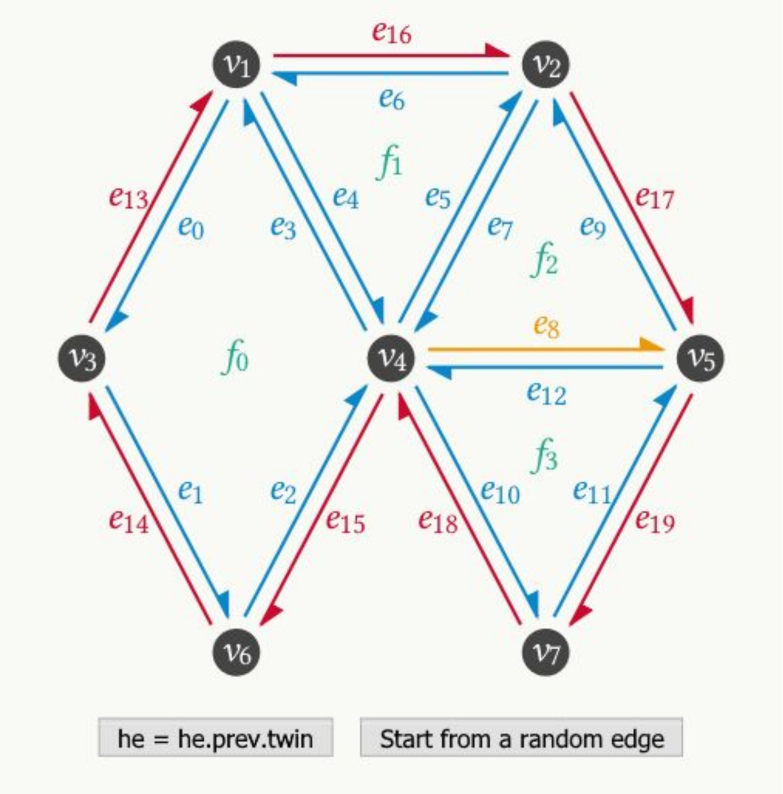
Half-edge vis

- Faces only need to store one half-edge to get all the other half-edges and vertices incident on that face.
- Stepper shows how to algorithmically traverse the face.



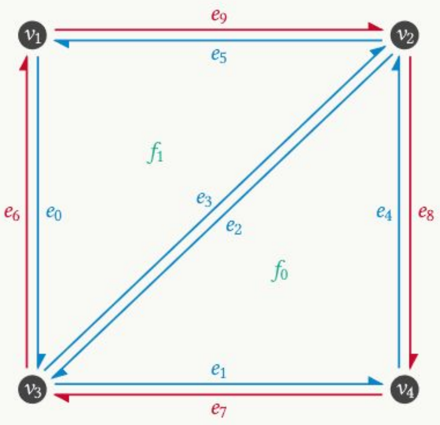
Half-edge vis

- Similarly, vertices only need to store one half-edge going out of it.
- Stepper shows how to algorithmically traverse the vertex umbrella.



Half-edge vis

- Last section walks through how to modify a half-edge data structure
- Visualizes how the mesh becomes inconsistent during the modification process, then is fixed to become consistent



RECORDS

Vertex	Coordinate	Incident edge
v_1	(0, 1, 0)	e_0
v_2	(1, 1, 0)	e_5
v_3	(0, 0, 0)	e_1
v_4	(1, 0, 0)	e_4

Face	Half-edge
f_0	e_1
f_1	e_5

Half-edge	Origin	Twin	Incident face	Next	Prev
e_0	v_1	e_6	f_0	e_1	e_2
e_1	v_3	e_7	f_0	e_2	e_0
e_2	v_2	e_3	f_0	e_1	e_4
e_3	v_3	e_2	f_1	e_5	e_0
e_4	v_4	e_8	f_1	e_5	e_3
e_5	v_2	e_9	f_1	e_3	e_4
e_6	v_3	e_0	∅	e_9	e_7
e_7	v_4	e_1	∅	e_6	e_8
e_8	v_2	e_4	∅	e_7	e_9
e_9	v_1	e_5	∅	e_8	e_6

Thank you!

www.students.cs.ubc.ca/~cs-424/tutorials/half-edge

github.com/enjmiah/interactive-geometry