Interactive Explainers for Geometry Processing Algorithms

Jerry Yin and Jeffrey Goh

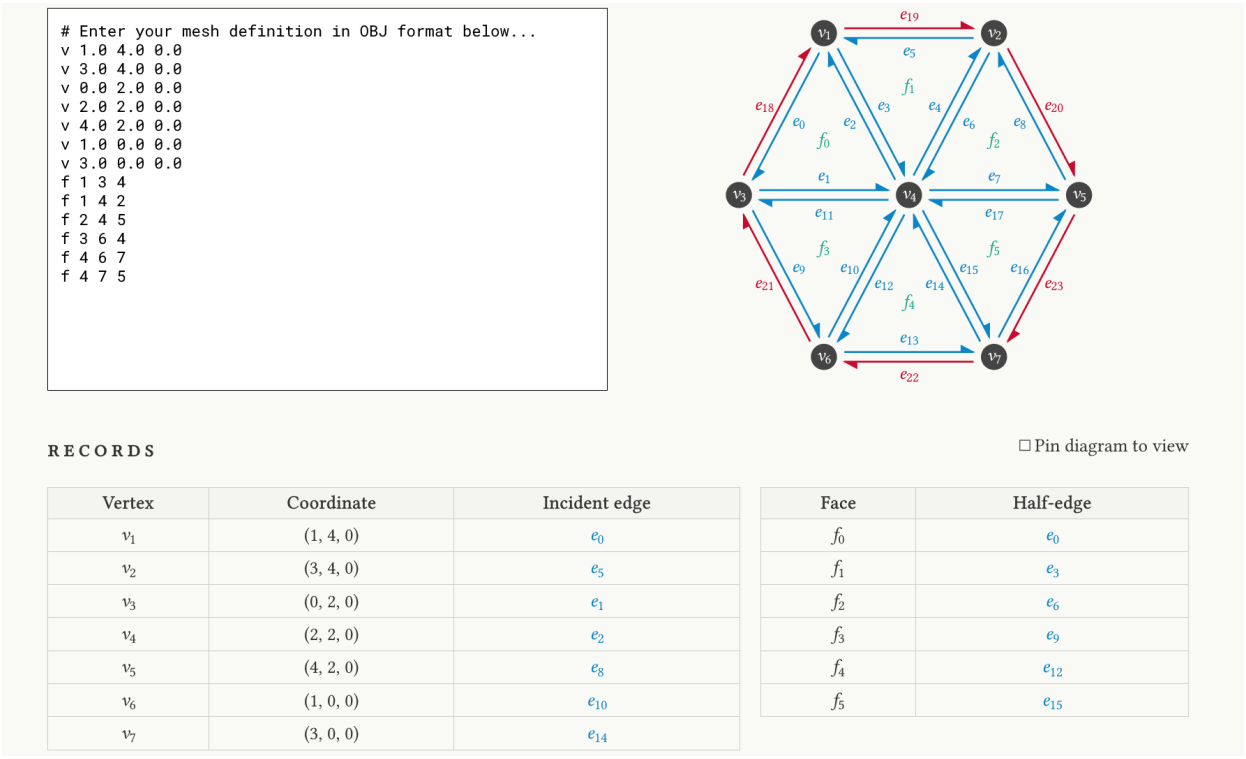


Fig. 1. The main visualization of our interactive explainer, which shows a half-edge diagram along with the records table stored in memory for the half-edge data structure. The input mesh is specified in the editor in the upper left, which automatically redraws the diagram and tables as edits are made.

Abstract—We create an interactive article introducing and exploring half-edge data structures, which are common in the field of computational geometry, for the purpose of teaching students in an undergraduate geometric modelling course. We visualize the half-edge data structure in both a table view, which represents the data structure as it is stored in memory, and as a diagram view, which visualizes the relationships between the half-edges and vertices in the mesh.

Index Terms—Interactive visualization, Interactive explainer, Computational geometry, Mesh, Education, Half-edge data structure, Winged edge data structure, Doubly connected edge list.

1 Introduction

Every two years, the University of British Columbia offers an undergraduate course in geometric modelling (CPSC 424). The course begins with a detailed foray into different ways of modelling curves, such as Bézier curves, Hermite curves, and B-splines. The course then moves from 2D to 3D by discussing methods for modelling surfaces and solids. The course differs from related courses in the mathematics department in that it explicitly discusses *meshes*: in particular triangle meshes, mesh data structures, and mesh algorithms. Meshes are ubiquitous in the field of computer graphics, and have a variety of practical applications.

Because the course is offered infrequently and comparatively few students take the course (due to it being a fourth year course), the notes for the course can be somewhat sparse. In particular, the course lacks detailed notes for many topics, and often uses third-party interactive demos for the first half of the course (see Previous Work section for examples). However, there are a lack of interactive demos on the internet for the second half of the course (meshes), and students must download models and install (and learn) software such as MeshLab or a 3D modelling application in order to interactively explore some of these topics.

Recently, there has been an increasing number of interactive online articles on specific technical subjects (sometimes called "interactive explainers"), which differ from traditional articles in that static figures are replaced with interactive visualizations and animations. These interactive explainers are made possible by the proliferation of modern web technologies such as SVG and highly optimized JavaScript engines. For our project, we created a detailed interactive explainer on the subject of half-edge data structures, a common data structure in the field of computational geometry. The notes emphasize *visual explanations*, choosing to teach via interactivity and visuals wherever possible, rather than relying mainly on text as in a traditional textbook.

- *Jerry Yin is with The University of British Columbia. E-mail: jerryyin@cs.ubc.ca.*
- *Jeffrey Goh is with The University of British Columbia. E-mail: gohzenhao@gmail.com.*

2 Related Work

As mentioned previously, interactive explainer articles are in vogue. There are online journals which exclusively publish interactive explainers, such as the online machine learning journal *Distill* [4] and the digital magazine *Parametric Press* [9]. These interactive explainers differ from traditional articles in that they include *reactive diagrams* and animations to assist with the text explanations. Reactive diagrams are diagrams that can be interacted with by the reader, providing a controlled environment in which the reader can experiment with different parameters of the visualization as part of deepening their understanding of the topic being explained. Interactive explainers have been popular enough that Conlen and Heer [2] have created a markup language specifically for authoring interactive explainers, and have had undergraduate students use it to create interactive articles as part of a final course project. Examples of interactive explainers created with Idyll can be found in their gallery [5]. Our project also uses Idyll.

Many innovative ways to visualize geometry have also been invented over the years. For example, an interactive system has been implemented to view complex 3D models using exploded views [7]. The diagram of the complex 3D model changes based on the interaction of users. User interactions included clicking, dragging and rotating. An iterative algorithm was used in determining how to remove unblocked parts of the 3D model. The paper serves as a good guideline on creating and viewing interactive interactive exploded views with 3D models. Although our project will most likely not use exploded views, it will be similar to our project as our goal is to create an interactive system to allow easier understanding of meshes for students. There has been other work on exploded views over the years [6, 12]. However, our work focuses on viewing local mesh neighbourhoods in 2D rather than entire meshes in 3D, as we care more about how meshes are represented in memory and local topology.

A few interactive articles and demos are already being used in CPSC 424. For instance, an article on Lagrange curves demonstrating Runge's phenomenon [3] is shown in lecture. The site allows the user to modify the control points of a Lagrange curve and also vary its knots. A plot of the curves and a plot of the Lagrange basis functions for the user-specified knots are juxtaposed. The same site has many other interactive articles on splines. Other interactive demos include one on manipulating Hermite curves [8] and one visualizing the de Casteljau algorithm (made by Jerry) [13], although these are not articles because they do not contain any explanation.

The introductory section of our interactive explainer describing the motivation behind half-edge data structures is based on a CPSC 424 slide deck [10], although we go into more explicit detail in our article.

3 Data & Tasks

In our reactive diagrams, we plan on using hand-crafted geometric models that are designed to be instructive and simple enough to be interactive. The models will be in the form of meshes, which specify vertex positions and vertex connectivity. The surface of the mesh is defined by a set of planar faces, whose boundaries are defined by cycles of vertices connected by edges. Meshes thus fall under the category of geometric data, but they are also an example of a network, since they can be represented as vertices connected by edges. However, when understanding the shape of a model, it is usually not enough to consider the spatial or topological features of the mesh in isolation—one must consider both simultaneously.

The meshes are small enough that any changes the user makes to the mesh can be displayed instantly. For example, our main visualization starts with a six-sided vertex umbrella, which has seven vertices, twelve edges, 24 half-edges, and six faces. In a later section, we walk through the implementation of an edge-flip algorithm, which flips the orientation of an edge situated in the middle of a cycle of four vertices.

3.1 Half-edge data structures

We have created an article on *half-edge data structures*. Half-edge data structures, also known as *winged edge data structures* or *doubly-connected edge lists*, are a common way of representing meshes in memory, since they allow fast, constant-time access to common queries necessary for implementing most mesh algorithms [1].

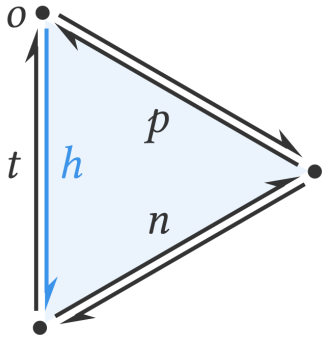


Fig. 2. Visualization of a half-edge h, along with its twin t, previous half-edge p, next half-edge n, origin vertex o, and incident face (shaded light blue). This exact figure appears in our interactive explainer, and features an interactive caption. When parts of the caption are hovered over with the cursor, the relevant parts of the diagram are highlighted in orange.

For example, consider a mesh with vertices V and faces F represented in the following face-list representation:

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

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

In order to determine whether the vertices v_3 and v_6 are connected by an edge, one must iterate through the entire face list until one finds (or fails to find) the desired edge.

A half-edge data structure allows for efficient query operations. In a half-edge data structure, each edge is represented by a pair of half-edge twins, with both twins pointing in opposite directions. Faces are represented as an oriented (usually counter-clockwise with respect to the viewer) cycle of half-edges. A half-edge data structure stores three types of records: half-edge records, vertex records, and face records.

A half-edge record represents a single half-edge, and stores a pointer to its corresponding twin half-edge, as well as the next and previous half-edges along the same face or hole. A half-edge record also stores pointers to the vertex and face records corresponding to the origin vertex and incident face of that half-edge, respectively. Fig. 2 shows a half-edge in relation to the objects its record points to.

A face record stores a pointer to an arbitrary half-edge incident on that face, and a vertex record stores a pointer to an arbitrary half-edge which originates from that vertex.

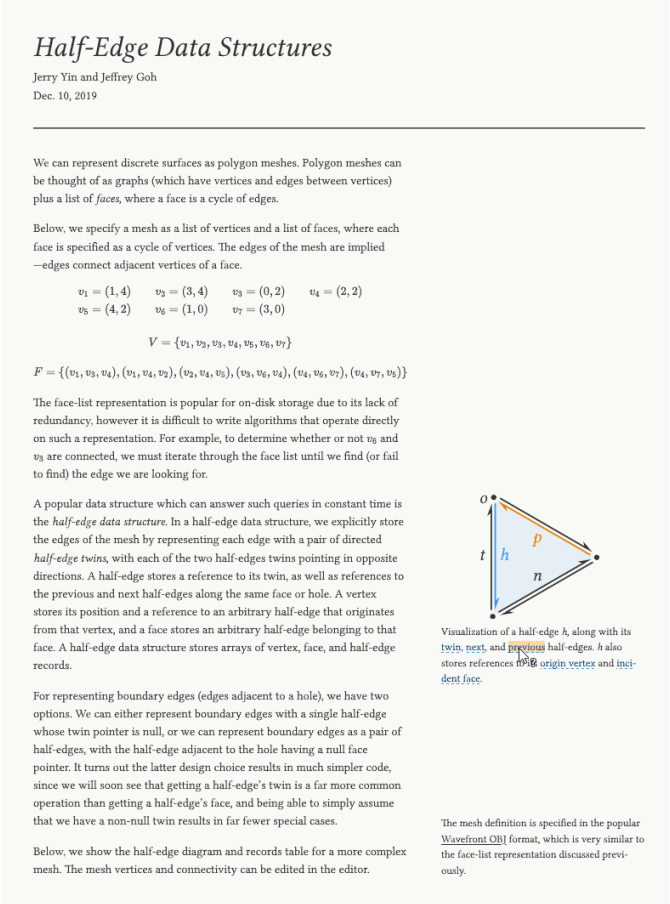
3.2 Goals

Students must understand both the low-level representation of a half-edge data structure as a collection of records with pointers as well as the high-level representation of a half-edge data structure as the abstract representation of some shape in space. This is because students need to understand and reason about meshes at a high-level, but also need to be able to implement mesh algorithms, which requires a deep understanding of the low-level.

Thus, our goals are for students to understand how to

1. given a mesh specified in a compact representation such as the face-list format, draw out the half-edge diagram corresponding to that mesh;
2. given a mesh specified in a compact representation, fill in the table of half-edge, vertex, and face records corresponding to that mesh;

3. efficiently perform common queries on a half-edge data structure; and
4. write algorithms which modify a half-edge data structure without leaving it in an inconsistent state.



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.

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

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

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

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_6 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.

For representing boundary edges (edges adjacent to a hole), we have two options. We can either represent boundary edges with a single half-edge whose twin pointer is null, or we can represent boundary edges as a pair of half-edges, with the half-edge adjacent to the hole having a null face pointer. It turns out the latter design choice results in much simpler code, since we will soon see that getting a half-edge's twin is a far more common operation than getting a half-edge's face, and being able to simply assume that we have a non-null twin results in far fewer special cases.

Below, we show the half-edge diagram and records table for a more complex mesh. The mesh vertices and connectivity can be edited in the editor.

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.

The mesh definition is specified in the popular Wavefront OBJ format, which is very similar to the face-list representation discussed previously.

Fig. 4. The view the reader is greeted with when the article loads.

4 Solution

Our article begins with a description of the motivation behind half-edge data structures, and why they are preferred over face-lists when used to implement geometry processing algorithms. Fig. 4 shows the view that the reader is presented with when they load the article in their web browser.

The design of the article mimics the design of the existing tutorial notes written by Jerry, which are in turn based on the design of Edward Tufte handouts.

4.1 Main visualization

For our main half-edge data structure visualization, we implemented the visualization shown in Fig. 1 and Fig. 3. This visualization is intended to fulfill goals 1 and 2 listed in Sect. 3.2. The editor in the top left, inspired by the live-reloading JPEG editor in "Unravelling the JPEG" [11], displays the contents of a mesh in the simple and popular OBJ text format. The user can edit the contents of the OBJ file, and the half-edge diagram (top right) and memory layout tables (bottom) will update in real-time.

The half-edge diagram in the top right shows the mesh laid out in 2D. The idea of representing half-edge twins as parallel arrows is not new; most discussions of half-edges use similar encodings to depict the structure. The marks in our diagram are coloured based on type: boundary half-edges are coloured blue, interior half-edges are coloured red, faces are coloured teal, and vertices are coloured black. In addition, vertices, half-edges, and faces are labelled based on their names in the memory layout table. The positions of the vertices are determined by the positions specified in the editor.

If the user hovers their cursor over any vertex, edge, or face in either the half-edge diagram or the memory layout table, the same item is highlighted in all other places it appears. This highlighting makes the structure of the half-edge data structure more apparent: for example, by hovering over a half-edge in the "Twin" column, it becomes apparent that the twin of a half-edge's twin is the original half-edge.

4.2 Margin figure

In the introduction of our interactive explainer, we include a small margin figure with an interactive caption. See the caption of Fig. 2 for a description of how it works. Since we created an interactive explainer, it is important for figures to be well-integrated with the text.

4.3 Steppers

In order to achieve goal 3, our article has two sections on how to perform queries efficiently. One is on how to find all of the half-edges incident on a face (remember that a face record only stores a pointer to a single, arbitrary incident half-edge), and the other is on how to find all of the half-edges originating from a given vertex, also known as the spokes of the vertex umbrella. In the article text, pseudo-code implementations of these iterators are given.

Our visualizations in these sections consist of a diagram portraying a local neighbourhood of a mesh, and a button which the reader can press to move on to the next iteration of the loop. We use the same visual encodings as the half-edge diagram in the main visualization. The half-edge in the current iteration is highlighted, so for the visualizations in this section, we disable hover highlighting, which would be useless anyway since we do not display the records table in this section. After pressing the button, an animation will play which shows the pointer indirections which are followed in order to proceed to the next iteration of the loop. The reader can press the button repeatedly to iterate over the loop.

Our local neighbourhood in the vertex umbrella case is carefully chosen to include both non-triangular faces and boundary half-edges, in order to show that our iterators are robust to these cases.

4.4 Consistency checker

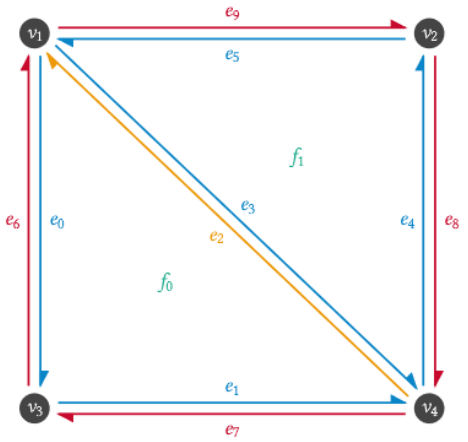
In the last section of our article, we walk through how to modify a half-edge data structure, which achieves goal 4. Modifying a half-edge data structure usually involves corrupting it, and then uncorrupting it through a series of operations. The article walks through a concrete example of implementing an edge flip operation, which flips the orientation of an edge in the middle of a cycle of four vertices.

In the visualizations in this section, we re-enable linked highlighting, and we display the half-edge diagram and records table side-by-side. The article intersperses pseudo-code with visualizations of the mesh at each state of the visualization. We also highlight newly modified cells in the records table light blue. In the second-last visualization, we must visualize the mesh whilst it is in an inconsistent state. We outline the inconsistent cells in the records table in red, which also serves to remind us which cells need to be made consistent in the algorithm implementation. Fig. 6 shows such a view.

5 Implementation

Our half-edge article is created using the Idyll markup language [2]. The content of the article is written in a Markdown-like format, which is then converted into a single HTML page by the Idyll compiler. The half-edge diagram is drawn using SVG elements, whose structure is created using the D3 library. Idyll allows the author to create custom interactive components by writing React components in JavaScript, and then embed them in the document using a special syntax. We implemented all of our components and interactivity using React. We did not use any of the built-in components or stylesheets that come with Idyll. We based our half-edge data structure implementation loosely on a stripped-down version of the implementation provided with Assignment 7 of CPSC 424, with our main modification being that

```
v 0.0 1.0 0.0
v 1.0 1.0 0.0
v 0.0 0.0 0.0
v 1.0 0.0 0.0
f 1 3 4
f 1 4 2
```



RECORDS

☐ Pin diagram to view

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_2

Face	Half-edge
f_0	e_0
f_1	e_3

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_4	e_3	f_0	e_0	e_1
e_3	v_1	e_2	f_1	e_4	e_5
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

Fig. 3. Visualization of a small mesh, with the half-edge corresponding to the cell under the cursor highlighted in both the records table and the half-edge diagram.

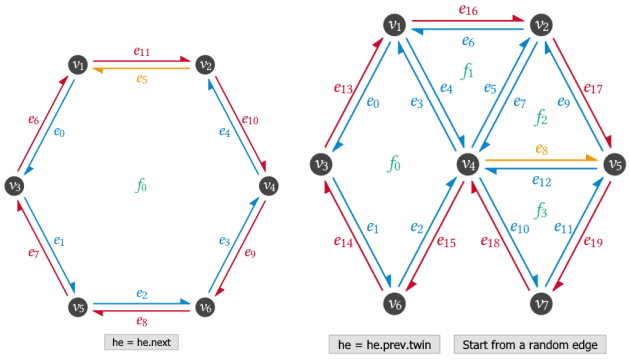
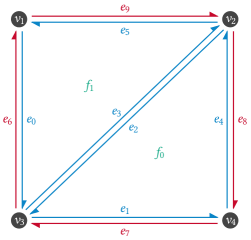


Fig. 5. Left: Face iterator visualization, with half-edge at the current iteration highlighted in orange. Right: Vertex umbrella iterator visualization, with half-edge at the current iteration highlighted.



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

Fig. 6. Visualization of an inconsistent half-edge data structure. Newly modified cells are in light blue, and inconsistent cells are outlined in red.

we support non-triangular faces and non-closed (open) meshes. We also used their 3D vector class.

The main component was the most challenging to implement. The half-edge diagram had to be drawn from scratch. The linked highlighting was difficult to implement due to the difficulty of determining which object was being moused over and propagating that information to all the other components as well as updating the display. The difficulty of understanding the enter-exit-update model of D3 was also the cause of many subtle bugs which popped up while we were implementing the article. Label and half-edge placement was also difficult, since the drawing of the faces needs to be consistently oriented throughout the entire mesh.

During the writing of the article, we also ran into many issues with Idyll's lexer. Despite seeming very Markdown-like, Idyll's parser differs greatly from a standard Markdown implementation, with even things like whether or not a paragraph is ended with terminating punctuation affecting the structure of the output HTML. In the end, to deal with some corner cases, we had to implement a custom Idyll component which accepts input in the form of an HTML string and outputs HTML corresponding to that string. This is a standard feature of Markdown due to Markdown being a superset of HTML, but is not supported natively in Idyll.

Table 1 breaks down the split of the work done for the project.

Task	Jerry (%)	Jeff (%)
Project proposal	80	20
Records table component	20	80
Records table highlighting	0	100
Half-edge diagram highlighting	0	100
Half-edge diagram labels	0	100
Half-edge diagram layout	90	10
OBJ editor component	100	0
Article text	100	0
Supplementary vis (margin figure, steppers, consistency checker)	100	0
Last minute polish, refactoring, bug fixes, animations	75	25
Peer review & presentation slides	80	20
Presentation	100	0
Final report	60	40

Table 1: Work breakdown.

6 Evaluation

Upon being posted to the CPSC 424 Piazza discussion board, students responded with positive comments. One student was intrigued by the technology behind the implementation, and wanted to know more about Idyll.

7 Discussions & Future Work

Our main visualization draws a half-edge diagram in 2D, essentially projecting the mesh onto the XY plane by ignoring the Z component specified in the mesh. We felt justified in doing this in part because half-edge data structures generalize to the 3D case without any special considerations, and 3D diagrams suffer from visualization problems such as occlusion and crowding. Another reason for visualizing half-edge diagrams in 2D is that none of our explanations require the reader to consider more than a single vertex umbrella, and vertex umbrellas are almost always topologically planar. Few algorithms need to operate on more than a small number of half-edge umbrellas at once, thus we do not believe that a 3D half-edge diagram would be more insightful. However, our projection of the mesh into 2D space could be made more intelligent by finding the optimal projection onto a plane instead of using a fixed projection.

One weakness of our main visualization is scalability. As the number of faces in the mesh increases, the records table can get quite large—a mesh with n faces has at least $3n$ half-edges. Although a pinning function which keeps the half-edge diagram pinned to the top half of the browser window was implemented, this does not fix the problem where the user wants to look at two different rows of the half-edge table at once when both rows do not fit on screen simultaneously.

The highlighting could also be made more sophisticated. Instead of merely highlighting the component under the cursor, we could also have implemented automatic highlighting of related components as well. For example, if the user hovers over a half-edge, the hovered half-edge's twin, previous, and next half-edges could also be highlighted in the table in different colours. This could make the symmetries between relations even more apparent.

Another improvement that can be made is to implement editable code and tables. Rather than have the steppers implement a static code snippet, an interactive editor can be added so that students can figure out the correct algorithm by themselves. As for the records table, instead of limiting interaction to highlighting, the table can be made editable to allow students to corrupt and then repair the mesh. Should these features be implemented, a reset button that resets the editor, diagram and code snippets back to their initial states would be also be a useful feature.

Due to time constraints and the complexity of implementing the half-edge visualization, only one article was completed for the entire duration of this project. We were not able to implement a second article on mesh subdivision as stated in our initial project proposal. We decided to produce a single high quality article rather than two articles with buggy interactions. One thing we underestimated was that every new figure required some modifications of the existing components in order to address some concept discussed in the text. Instead of trying to create a single visualization which tries to address every use case beforehand, we would have been able to architect our components better if we had written the article text at the same time as we created the visualizations, instead of writing all of the article text at the end.

If we had more time, we would want to create more articles on geometry processing topics, such as mesh subdivision, simplification, and parameterization. We believe that an entire interactive textbook on computational geometry would be helpful to educators and students everywhere.

8 Conclusion

We created an article exploring half-edge data structures through interactive visualizations, with the purpose of teaching students in an undergraduate geometric modelling course. We implemented a 2D half-edge diagram with linked highlighting between the diagram and records tables, with the intent of bridging the gap between high-level and low-level representations of half-edge data structures. The responses received from the students were positive and suggested that the interactive explainer was helpful.

References

[1] B. G. Baumgart. Winged edge polyhedron representation. Technical report, Stanford Dept. of CS, 1972.

[2] M. Conlen and J. Heer. Idyll: A markup language for authoring and publishing interactive articles on the web. In *ACM User Interface Software & Technology (UIST)*, pp. 977–989, 2018.

[3] E. Demidov. Interpolating Lagrange curve. `https://web.archive.org/web/20191104080854/https://www.ibiblio.org/e-notes/Splines/lagrange.html`, 2001.

[4] Distill: Latest articles about machine learning. `https://distill.pub`. Accessed: 2019-11-01.

[5] Idyll example gallery. `https://idyll-lang.org/gallery`. Accessed: 2019-11-01.

[6] O. Karpenko, W. Li, N. Mitra, and M. Agrawala. Exploded view diagrams of mathematical surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1311–1318, 2010.

[7] W. Li, M. Agrawala, B. Curless, and D. Salesin. Automated generation of interactive 3D exploded view diagrams. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, pp. 101:1–101:7, 2008. doi: 10.1145/1399504.1360700

[8] Lior. General cubic hermite spline demo. `https://codepen.io/liorda/full/KrvBwr`. Accessed: 2019-11-03.

[9] Parametric Press. `https://parametric.press/issue-01/`. Accessed: 2019-12-10.

[10] A. Sheffer. Meshes - data structures. https://www.students.cs.ubc.ca/~cs-424/Slides/13_MesheDataStructures.pdf. Accessed: 2019-12-12.

[11] O. Shehata and M. Conlen. Unravelling the JPEG. *Parametric Press*, 2019.

[12] J. J. van Wijk. Unfolding the earth: Myriahedral projections. *The Cartographic Journal*, 45(1):32–42, 2008. doi: 10.1179/000870408X276594

[13] J. Yin. De Casteljau's algorithm. https://www.desmos.com/calculator/s78usaowv9. Accessed: 2019-11-03.