DEVELOPMENT OF PHYSICAL CONNECTORS TO ASSEMBLE UNFOLDED PLANAR PANELS FOR THE CONSTRUCTION OF LARGE SHAPES

A Thesis

by

SHENYAO KE

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Ergun Akleman
Committee Members,	Ann McNamara
	Negar Kalantar Mehrjardi
Head of Department,	Tim McLaughlin

May 2017

Major Subject: Visualization

ABSTRACT

In this work, I propose to identify and construct physical counterparts of the links in geometric data structures that are used in shape modeling. These physical connections will be used to construct large shapes from unfolded polygonal meshes that are represented by geometric data structures. I will implement physical connectors as an extension to a unfolding system that has been developing in our laboratory at Texas A&M University. Using these connectors, any unfolded shape can be constructed by using laser-cut developable panels based on corresponding data structures. This approach is particularly suitable to construct complicated sculptural and architectural shapes from anisotropic materials that can only be bent in one direction.

ACKNOWLEDGMENTS

I would like to thank Dr. Ergun Akleman, my committee chair, for providing me continuous guidance and support while I was working on my thesis. I sincerely appreciate your generous offer of enormous knowledge during my time at Texas A&M University. I would also like to thank the rest of my committee, Dr. Ann McNamara and Prof. Negar Kalantar Mehrjardi for their expertise and wise guidance they have put into this thesis.

I would also like to thank my friends Yinan Xiong, Qiao Wang, Dustin Han, Yuxiao Du, Alyssa Pena, Kate Sackreiter, Fermi Perumal and etc. Thank you for your company whenever I was happy or stressed. I am so grateful to meet you in Department of Visualization and become your friend.

In the end, I would like to thank my parents for their endless support and encouragement through the years. They gave me a life to live on this world and taught me to discover the beauty of it. Their faith in me has always been guiding me to be a better man. I would never be able to achieve anything without their unconditional love.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Ergun Akleman and Ann McNamara of the Department of Visualization and Professor Negar Kalantar Mehrjardi of the Department of Architecture.

The software named "Unfolding" was developed by Peihong Guo, You Wu and Shenyao Ke, of which the connector module was fully implemented by Shenyao Ke. All physical objects in Chapter 5 and Chapter 6 were assembled by Shenyao Ke, except Figure 6.3 and Figure 6.4 by You Wu and Shenyao Ke, and Figure 6.5 by Negar Kalantar and her students.

All other work conducted for the thesis dissertation was completed by the student independently.

Funding Sources

Graduate study was supported by a fellowship from NSF Foundation.

NOMENCLATURE

2D	Two-dimensional
3D	Three-dimensional
CG	Computer Graphics
HDS	Half-Edge Data Structure
API	Application Program Interface

TABLE OF CONTENTS

Page

LIST OF FIGURES

1. INTRODUCTION AND MOTIVATION

1.1 Introduction

With the design and construction of more and more complex and unusually shaped buildings (see Figure 1.1), the computer graphics community has explored various of new methods to develop physical constructions for large shapes. To construct large shapes, several cutting and unfolding methods are developed to convert 2-manifold surfaces into either single or multiple developable panels that can be laser-cut from large planar sheets of materials, such as paper, wood and metal.

In order to reconstruct 3D shapes from those flattened panels, necessary connections, which correspond to the geometric links from the original designs, are strongly required while generating those panels. Both architects and computer scientist have been working on developing different connections. Many of them require external support to stitch counterparts of the links [2, 1] (see Figure 1.2 as example), while others can use internal structure to hold the shape against external force from breaking them down [11, 8]. The community keeps working on developing different connectors and treats them as different types.

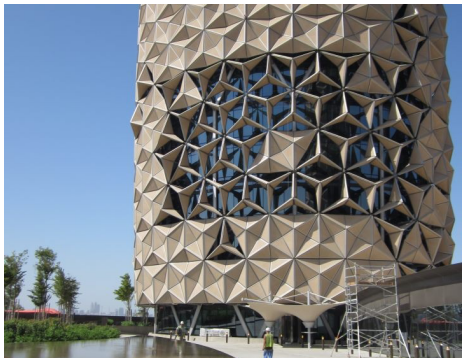
1.2 Motivation

It can be a lot of work to develop specific connectors when new designs come out. And people will be easily overwhelmed by huge amount of different connectors without clear classification when deciding which type to use in their own designs.

Instead of doing redundant work designing new types, connections can be classified as certain categories with certain features. Thus, designers can follow simplified design patterns without worrying about stability, expense and feasibility, and focusing on the shape and structure.

(a) Metropol Parasol (Photo by Rubendene)

(b) Heydar Aliyev Center (Photo by Iwan Baan)

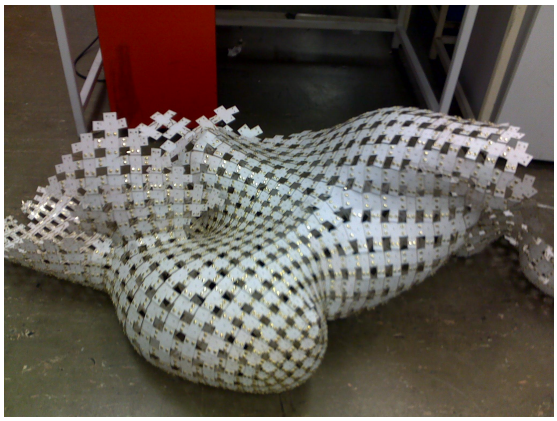


(c) Al Bahar Towers

Figure 1.1: Architectures over the world.

Our research group has presented our current progress for realization of complex 3D surfaces and structures that are assembled from 2D planar shapes. We have constructed physical structures that can be raised and formed in 3D-space using minimal amount of planar materials. However, with the growth of our surface processing methods, different connectors have been introduced into our system, which makes the system heavier over the time. Thus, it is urgent to classify connectors in order to simplify the process and make it clear to designers.

(a) Progress 1



(b) Progress 2

(c) Final

Figure 1.2: Construction of a large bunny using physical version of graphs rotation systems (GRS) by Xing et al. [1].

2. PREVIOUS WORK

With the development of computer aided design, we can now design more complicated shapes and structures. We often need to physically realize these virtual shapes and structures especially in fields such as engineering and architecture. 3D-printing has become very popular to convert virtual shapes into physical structures [12]. There are many inexpensive printers available in the market. Makerspaces are built everywhere from libraries to copy-centers to design and print interesting shapes [13]. Unfortunately, despite the hype, it is almost impossible to construct large shapes using 3D printers [14, 15].

Laser cutting provides an alternative to construct large shapes [16]. With laser cutting, we simply cut developable panels from paper or thin metal. We can then bend and connect them to construct large physical shapes [17]. Although this process doesn't sound difficult, in practice, it requires significant amount of technical expertise to construct interesting buildings, such as Walt Disney Concert Hall in Los Angeles [18]. for architectural firms. Therefore, there is a strong need for the development of processes for practical and economical construction of physical structures that can be raised and formed in 3-space using planar panels.

One of the critical issues related to construction with multiple panels is to assemble 3D shapes by connecting panels with each other. In other words, there is a need for the development of connectors and connection methods to join panels to construct 3D shapes. We observe there exist a wide variety of methods in the industry for joining planar materials such as crimping, welding, soldering, brazing, taping, gluing, cementing, using adhesives, using magnets, vacuuming such as suction cups, or even friction.

These joints, also called connectors, can be classified into two main categories: permanent and non-permanent. Permanent joints cannot be removed without damaging the

joining components. On the other hand, non-permanent joints can always be removed without any damage.

In this work, I am interested in non-permanent joints. I observe that many researches have been accomplished to create connections for physical structures using a wide variety of non-permanent joints. In this chapter, I review some of the previous works related to our research of physical connectors. I have identified types of non-permanent joints that are used to connect surface elements: (1) External and (2) Embedded. External joints includes only fastening, nuts and bolts and other non-permanent connectors. Embedded joints can be zippers, slit parts, plugs and holes.

2.1 External Joints

External joints are usually non-permanent metallic elements such as fasteners or screws-and-nuts. For the construction of large structures, we usually see fasteners since they are cheaper than screws-and-nuts. On the other hand, in the future, we can expect to see screws-and-nuts to build large shapes in a more precise manner.

2.1.1 Connecting with Fasteners

Fasteners are non-permanent metallic, such as brass, connectors. They are used to mechanically join two or more thin surfaces together. It seems that they are most widely used non-permanent joints to construct large shapes probably because of their availability and inexpensiveness.

Akleman et al. [2] used brass fasteners to connect multiple paper strips for construction paper strip sculptures . In their methods, fasteners play roles of vertices in physical data structures (see Figure 2.1).

Xing et al. [1] used brass fasteners to connect panels that represent vertices. In their case, the vertices are connected along the edges. Therefore, in this method each brass fastener connected only two vertex component panels (See Figure 2.2).

(a) Tetrahedron (b) Octahedron (c) Cube

Figure 2.1: Paper-Strip representations of platonic solids with brass fasteners connecting each vertex [2].

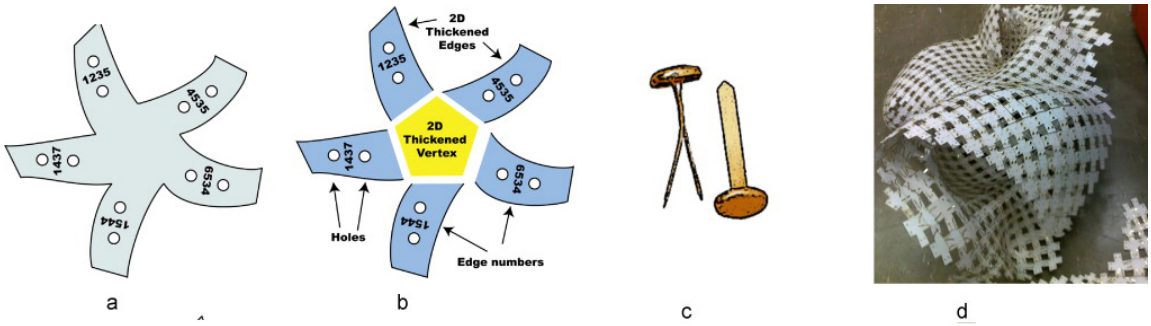


Figure 2.2: An example of assembling large number components from multi-panel pieces (a) into 3D objects in real world (b) from Xing's work[1]. Physical connections is placed in this example.

David Reiman [3] also used fasteners to connect rectangular panels to construct representations of regular and semi-regular polyhedra. In his case, each rectangular panel plays a role of edge and connections happen in a quadrilateral formed in face corners. Every quadrilateral is connected only with another quadrilateral (see Figure 2.3).

2.2 Embedded Joints

Embedded joints are integral part of the surface. Using embedded joints we can assmble two parts without a need an external joint such as fastener or We have identified em-

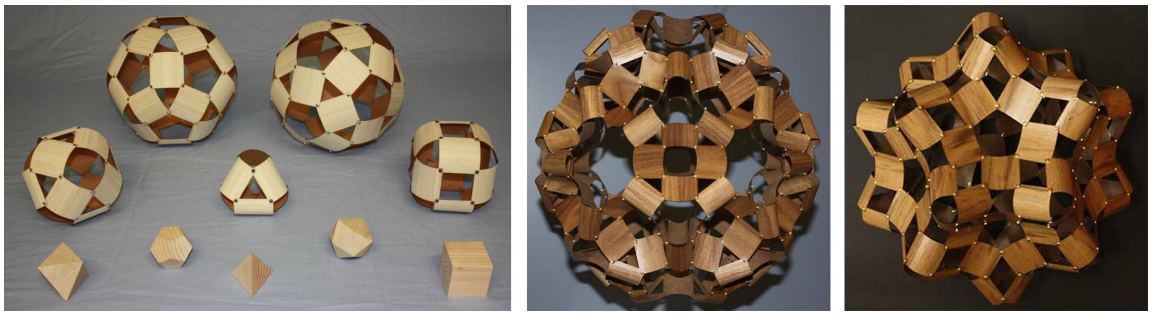


Figure 2.3: David Reiman's Phsyical Representations of Regular and Semi-regular Polyhedra [3]. Each rectangular panel is cut from veneer and connected by Split-pin Fasteners

bedded joints can be (1) Slitted parts; (2) Plugs and holes, (3) Teeth connectors and (4) Zippers.

2.2.1 Connection with Embedded Slits

Embedded slit structure is first introduced by George Hart [19] in 2004 as slide-together structure for platonic solids. Hart developed a wide variety of interlocked planar pieces for construction of slide-together sculpture [19] and symmetric modular sculptures [20, 21]. Miller et al [4] independently discovered a general version of embedded slit structure. Their work has shown that these slit polygons can be used to construct any physical shape as interlocking-slit structure (See Figure 2.4).

The concept of embedded slit structure is also used in toy manufacturing. Figure 2.5 shows two types of recently discovered "slit" geometric toys corresponding to half-edge data structure. Architect Dick Esterle [5] introduced *"Space Chips"* to the public in 2011. Each space chip in this design has rigid circular structure with slits. *"ITSUPHUN" Puzzles* is another slit geometric toy introduced in 2015 Bridges Art and Math conference [6]. In both geometric toy cases, slits again play the role of half-edges that allow to form edges when correctly connected.

(a) (b) (c)

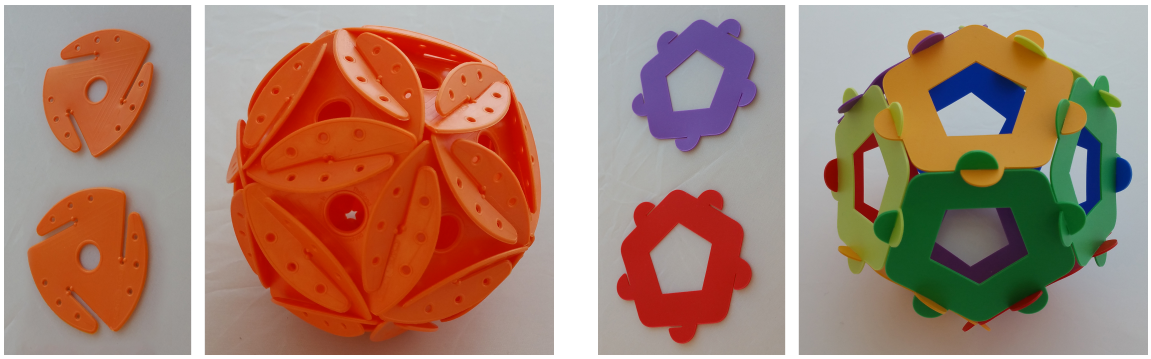
Figure 2.4: Examples of models corresponding to half-edge data structures [4], which represents (a) a shape made from five cubes, (b) a dodecahedron, and (c) an octahedron. In this case, slits are physical connectors that correspond to half-edges that link faces to each other.

2.2.2 Connection with Embedded Plugs and Holes

Flexeez (also known as "Wammy" in Japan) is a geometric construction toy that lets you create any object when twisted, bent and wrapped. It uses embedded plugs and holes to construct shapes corresponding to quad-edge data structure. Each single piece in Figure 2.6 represents a quad-edge.

2.2.3 Connection with Teeth

I observe that there also exists joints are repeated pattern that can connect with each other, which I call teeth connectors. This type of teeth connectors are used to construct D-forms by the London designer Tony Wills [7, 22, 23] (see Figure 2.7). On the other hand, Gönen et al [24] did not use such an non-permanent connection for the construction of their alternative D-form construction.



(a) Space Chips (b) "ITSPHUN" Puzzles

Figure 2.5: Examples of geometric toys that uses embedded slits for connection. The particular structure in *"Space Chips"* [5] represents an icosahedron, and a regular dodecahedron in *"ITSPHUN"* [6].

2.2.4 Connecting with Zippers

Zippers are external parts. However, once they are stitched, we can consider them as embedded pieces. Although they conceptually resemble teeth connections, connection mechanism is different. In 2010, Demaine et al [8] used zippers to construct pita-forms (see Figure 2.8c). A pita-form is a 3D concept introduced by Demaine and O'Rourke obtained by glueing the boundary of a single 2D planar shape to itself [25, 26] (see Figures 2.8a and 2.8b). By replacing glue with zipper, Demain constructed non-permanent pita-forms that can be folded and unfolded without any damage. Demain's method can also be used to fold D-forms that consists of two planar panels [22, 7, 27, 24].

2.3 Classification

In this thesis, I will classify non-permanent connections that are used to join planar panels using the concept of 2-manifold. My mathematical classification provides a generalized model, which can be universal solution to any geometry processing algorithm.

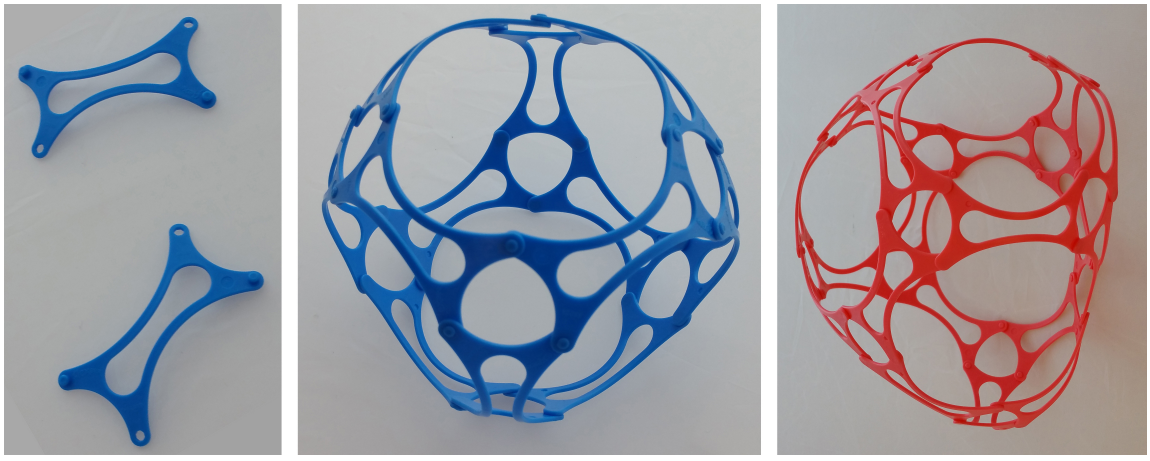
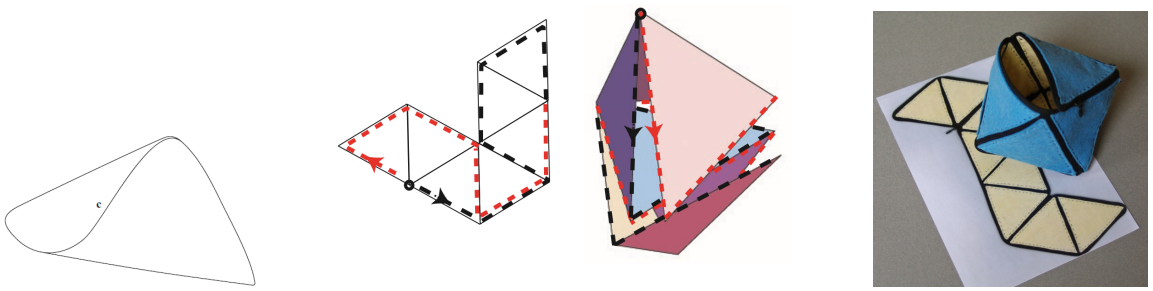


Figure 2.6: Examples of *Flexeez* (also known as "Wammy" in Japan) pieces and examples of 3D structures build by Flexeez pieces. Note that this plastic pieces have embedded plugs and holes to assemble the pieces together to construct 3D structures. KOKUYO Co., Ltd. owns the copyright of this toy.

Figure 2.7: Examples of Tony Willis' D-Form structures built from sheet metal [7]. As shown in details these D-Forms are constructed by using teeth connectors along the perimeter.



(a) A pita-form (b) Another example of pita-form (c) A zipped object

Figure 2.8: Examples of pita forms and application of zipper connectors on a physical octahedron [8].

3. THEORETICAL FRAMEWORK

In this chapter, I will present a theoretical framework for the development of joints. Since we are interested in physical realization of 2-manifolds, which are also called surfaces, I will first provide important concepts related to surfaces. These concepts will be useful for the development of theoretical concept of joint.

3.1 Preliminaries

I give a brief review of the background necessary for this thesis. A *2-manifold* or *surface* (or more precisely, a *2-dimensional manifold*) is a topological space where every point has a neighborhood topologically equivalent, i.e., *homeomorphic* to an open disk [28]. In other words, a shape is called a 2-manifold (or surface) if and only if every point on the shape has a neighborhood that is homeomorphic to the 2D Euclidean space [29].

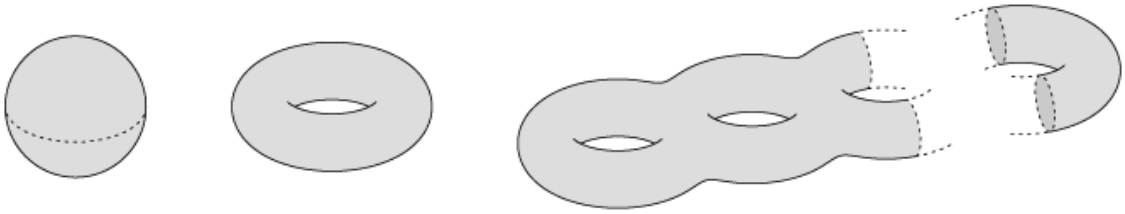


Figure 3.1: Examples of orientable surfaces. From left to right, genus-0, genus-1 and genus-k surfaces. As shown in the figure, genus-k surface is obtained by adding a handle to genus-(k-1) surface. Image courtesy of manifold atlas project [9].

A 2-manifold is *orientable* if it does not contain a Möbius band [9] (See Figure 3.1). In this work, our goal is to construct physical versions of orientable 2-manifold surfaces, which we simply call surfaces.

For the phsyical construction, concept of mathematical charts and mathematical atlas is

useful [29] (see Figure 3.2 for an example). A surface can be described using mathematical maps, called charts. We can view charts as the maps in the pages of an atlas. If the atlas provides all local maps of the world, the collection of all charts gives us overall world. Borrowing from this idea, collection of of charts provides us an atlas of 2-manifold surface [30]. In fact, Grimm and Hughes used concepts of the charts and atlas to parametrize genus-k surfaces [31] for Computer Graphics applications. In other words, a surface can be constructed from multiple overlapping charts where they overlap carry information essential to understanding the global structure.

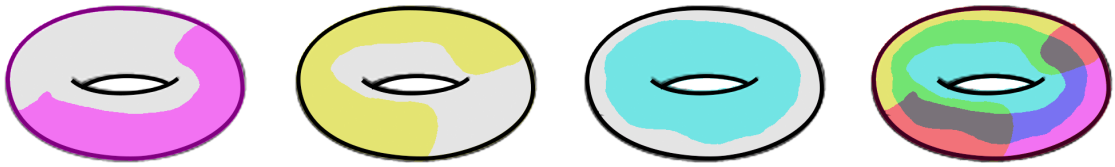


Figure 3.2: Examples of charts on a genus-1 surface. The first three images shows three charts on a toroid surface and the last image shows the atlas of these charts. Note that there are different types of overlapping regions over the atlas. For our purposes of construction of connections, it is essential to identification and classify these overlapping regions.

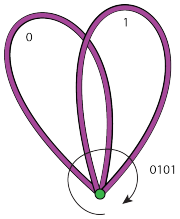
In this thesis, our goal is really to develop strategies for connecting physical versions of charts in overlapping regions to obtain 3D atlas that corresponds the original surface. Note that physical versions of charts are called panels and they are obtained by laser cutting planar materials such as paper, veneer or thin metal.

From the perspective of this thesis, the key is the identification and classification of overlapping regions of charts. There is, therefore, a need for a more clear definition to differentiate between different types of charts and to identify different types of overlapping regions. Fortunately, it turned our that the classical theory of graph rotation systems provides a theoretical framework to define and classify charts.

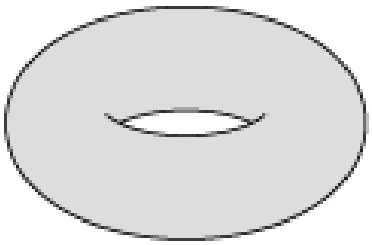
3.1.1 Graph Rotation Systems

The graph rotation system is a powerful tool for guaranteeing topological consistency. In this part, we introduce historical background and some mathematical fundamentals (see [9] for more detailed discussion). The concept of rotation systems of a graph originated from the study of graph embeddings due to Heffter [32]. Edmonds [33] was the first to explicitly study the rotation systems of a graph.

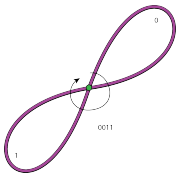
Let G be a graph. A *rotation* at a vertex v of G is a cyclic permutation of the edge-ends incident on v [33]. A *rotation system* of G, which is a list of rotations, one for each vertex of G, corresponds to embedding the graph to an orientable surface. An *embedding* of a graph G on the 2-manifold surface S, or more formally, a one-to-one continuous mapping ρ from the graph G to the surface S specifies a *mesh* on a 2-manifold surface S [28].



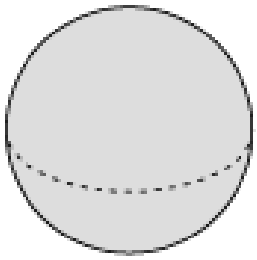
(a) A rotation system of a graph.



(b) A genus-1 surface.



(c) Another rotation system of the same graph.



(d) A genus-0 surface.

Figure 3.3: Two examples of rotation systems of a graph. Figure 3.3b shows an example of genus-1 surface where the graph in Figure 3.3a can be embedded. Figure 3.3d shows an example of genus-0 surface where the graph in Figure 3.3c can be embedded.

The vertices and edges of the mesh are the corresponding vertices and edges of the graph G, and each connected component of $S - \rho(G)$ is called a *face* of the mesh [34, 28]. The mesh is *cellular* if each connected component of $S - \rho(G)$ is homeomorphic to an

open disk and this graph embedding is called *cellular decomposition*. In this work, we only consider cellular decompositions.

We observe that the cellular decompositions constructed using graph embeddings can provide a way to define and classify charts. To define charts, we will use the concept of 2D thickening, again an idea borrowed from topological graph theory [9].

3.1.2 2D Thickening

The combinatorial structures around the embedded graph can be used to represent charts in 2-manifold surfaces. These charts can be described by 2-thickening of parts of the embedded rotation system of the graph. In this work, we will generalize the standard concept of 2D-thickening that can only defined on vertices and edges [35, 36].

Let R be a cellular region described by mesh as a union of a connected set of vertices, edges, faces and also let $T(R)$ denote the 2-thickening of R. We, then, define $T(R)$ as the open set that is union of all open sets defined by a geodesic metric around each point $p \in R$ [37]. The only requirement with the the geodesic metric is that $T(R)$ must stay cellular [1].

Using this simple definition, we can now construct charts. For instance, we can define a *vertex-chart* as a thickened-vertex. Note that every m-valent vertex is 2D-thickened into an m-sided chart with curved edges. Similarly, we can define an *edge-chart* as a thickened-vertex. Also note that in this process every 2D-thickened edge slightly larger version of itself that includes its two boundary edge-regions, which is different than classical definition of 2D-thickened edge [35]. Moreover, we can now define an *face-chart* as a thickened-face, which becomes slightly larger version of itself that includes its boundary edge- and vertex-regions. Figure 3.4 provides planar examples.

2D-thickened regions of vertices, edges and faces can help to classify overlapping

[1] Cellular requirement is not really necessary. However, it is better to keep it to keep exposure simple.

regions and existing connection methodologies. For instance, we can now precisely can say that the overlapping region of two thickened-faces that share one edge is the thickened version of that edge (see Figure 3.5).

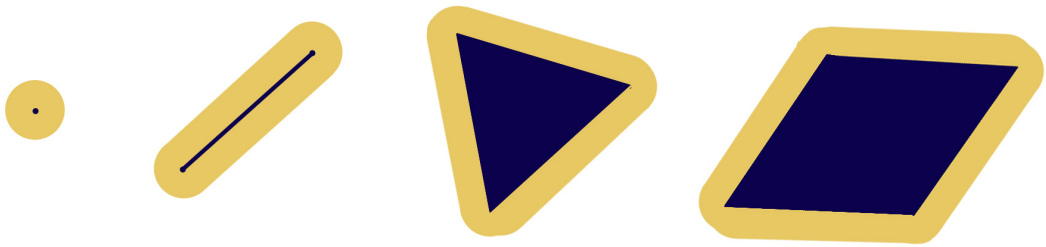


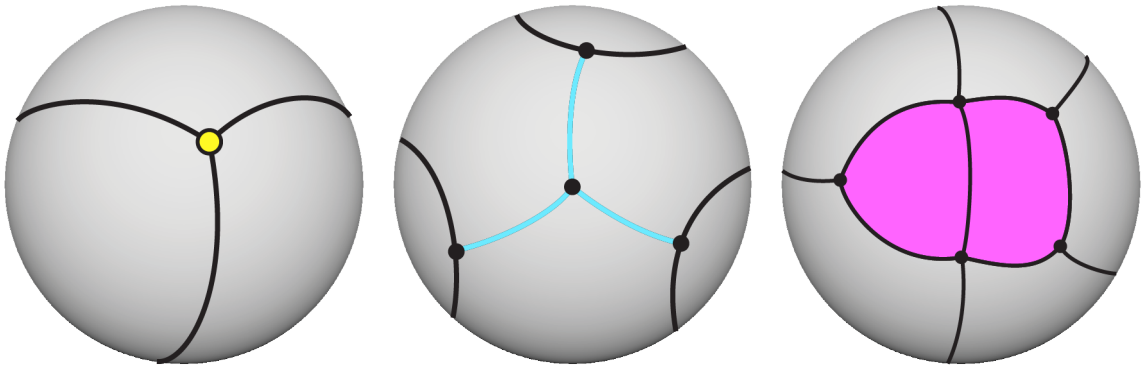
Figure 3.4: Examples of thickenings in plane and using Euclidean distance. Blue regions shows original entities, a vertex, an edge, a triangular and a quadrilateral faces. Yellow shows thickened regions. As it can be clear from this image, thickened-edge includes thickened versions of its two end-vertices and thickened-face includes its thickened boundary edges. Note that the thickened regions also corresponds to Minkowski sums, which are hard to compute [10]. This is not a problem for us since we are using this only for conceptual device to formalize our process.

3.2 Classification of Types of Charts

Existing methods discussed in Chaper 2 can be classified using 2D-thickening methodologies by identifying the types of charts that are used to construct 3D structures. This results are also consistent with our earlier classification using mesh data structures [38]. We observe that there are only three cases: (1) Methods use Thickened Edges as Charts; (2) Methods use Star-Shaped Charts; and (2) Method uses Quad-Edge Charts.

3.2.1 Thickened Edges as Charts

Paper strip structures [2] use thickened edges as shown in Figure 3.6. One disadvantage of this method, overlapping regions are vertex regions and the number of regions depends on the valence of the vertex. This makes it hard to use every type of joints in



(a) Vertex before thickened (b) Edges before thickened (c) Faces before thickened

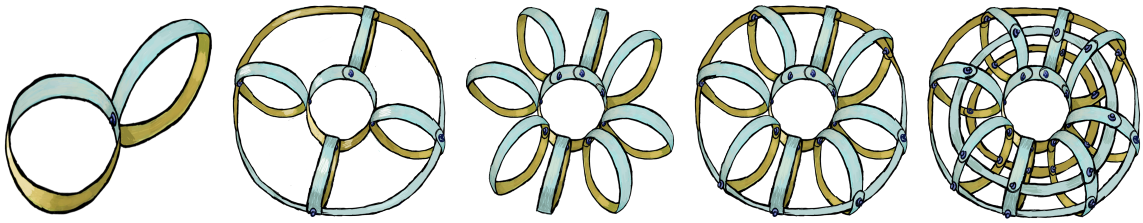
(d) Thickened vertex (e) Thickened edges (f) Thickened faces

Figure 3.5: Examples of thickening vertices, edges and faces on 3D shapes. (d) demonstrates a vertex thickened on a 3D surface from (a), where yellow represents the thickened area from original vertex. (e) demonstrates three thickened edges in cyan, which originally touch each other at the same vertex in (b). Their overlapping area is highlighted in dark blue in (e). (c) shows two face with different valences sharing the same edge. Once thickened, theire overlapping region turns out to be a thicked edge, which is colored red in (f).

these overlapping regions. An advantage is the method is that it is easy to construct shapes using planar developable panels since we only deal with thin and long shapes.

3.2.2 Star-Shaped Charts

In this case, charts are obtained by removing its thickened vertex regions from thickened face region. The results are what is called dual-vertex component shown in Figure 2.2. All embedded slit structures such as George Hart's slide-together sculptures [19],



(a) A toroid with single face (b) A toroid with eight faces. (c) Another toroid with eight faces. (d) A toroid with 16 faces. (e) A toroid with 32 faces.

Figure 3.6: These edge-charts are constructed by 2D-thickening edges of embedding different graphs to a toroidal, i.e. genus-1 surface. As shown here, embedding a different graph to the surface results in different set of edge-charts. Note that n number of edge-charts are overlapping in vertex-regions where n is the valence of the vertex. Also note that edge-charts alone does not form an atlas; in other words, do not cover the whole surface.

Miller and Akleman's interlocking-slit structures [4], Dick Esterle's Space-Chips [5] and ITSPHUN toys [6] also uses the same type of Face-Vertex Regions. Moreover, Demain's pita-forms and Willis's D-forms also in this category. In their case, embedded graphs is very simple. Namely, a single edge and two vertices. Therefore, those cases, can only be used for construction of genus-0 shapes.

The advantage of this method is that overlapping regions are edge-vertex regions and in every overlapping regions only two charts overlap. Another advantage is that the only empty region comes from vertices, which is usually very small. On the other hand, it is hard to use planar panels for faces that consist of double curved regions.

3.2.3 Quad-Edges Charts

In this case, we take the set difference of edge-charts and its vertex-charts. The results becomes quarilaterals with curved edges, which corresponds to quad-edge structures. Then we apply one more thickening operations to obtain overlapping regions. This thickening operation creates overlapping regions in the corners of the quadrilaterals. Methods such as David Reiman's physical representations of regular and semi-regular polyhedra

[3], Flexeez and our work on the construction with physical version of quad-edge data structures [17] uses this type of charts.

There are several advantages of this method. First, only two charts overlap in every overlapping region. Second, quad-edge charts can approximate the original shapes using developable panels. On the other hand, this approach also leaves a significant amount of empty space not covered with charts.

3.3 Constraints for Connections in Overlapping Regions

Now, we are ready to formalize connections in overlapping regions. Without loss of generalization, let $\mathcal{O}$ denote an overlapping region and let $T(R_i)$ denote chart i that overlap other charts in overlapping region $\mathcal{O}$ where $i = 1, 2, \ldots, n$ then the following equality holds:

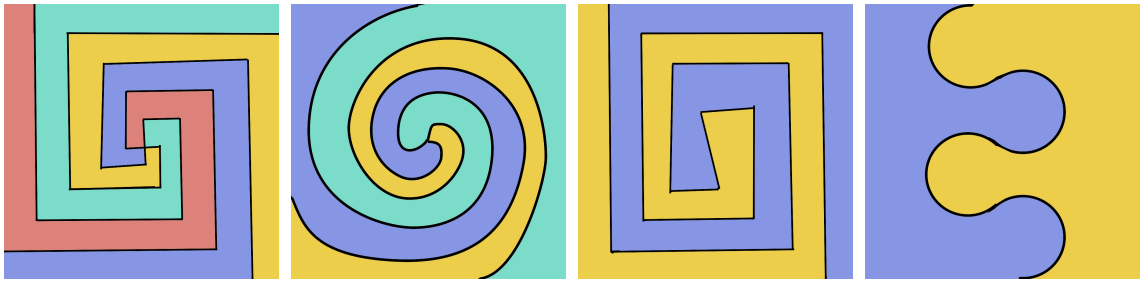
$$\bigcap_{i=1}^{n} T(R_i) = \mathcal{O};$$

as discussed earlier, there are only only two following cases:

1. n is any positive integer larger than 2. In this case, R_i's are edges or $T(R_i)$'s are thickened edges.

2. $n = 2$. In this case, $T(R_i)$'s are either star shaped or quad-edge charts.

In overlapping regions, we have a strict constraint: two physical objects can be in the same positions. In other words, if these charts are turned into physical objects, they cannot fill the same 3D space in the same time. Therefore, we need to "shape" overlapping regions in such a way that in every point in overlapping region, there will be only one chart. Let M_i denote the reshaping operation over chart i that can provide the following constraints



(a) An overlapping region with four spirally interlocked charts.

(b) An overlapping region with three spirally interlocked charts.

(c) An overlapping region with two spirally interlocked charts.

(d) An overlapping region with two teeth-interlocked charts.

Figure 3.7: Examples of strategies to obtain interlocked charts in overlapping regions.

for all overlapping regions

1. $M_i(T(R_i))$ is homeomorphic to an open disk (3.1)
2. $\bigcap_{i=1}^{n} M_i(T(R_i)) = \emptyset$ where $\emptyset$ is the empty set, and (3.2)
3. $\bigcup_{i=1}^{n} (\mathcal{O} \cap M_i(T(R_i))) = \mathcal{O}$ (3.3)
4. $M_i(T(R_i)) - \bigcap_{k=1}^{m} (\mathcal{O}_k \cap M_i(T(R_i))) = T(R_i) - \bigcap_{k=1}^{m} (\mathcal{O}_k \cap T(R_i))$ (3.4)

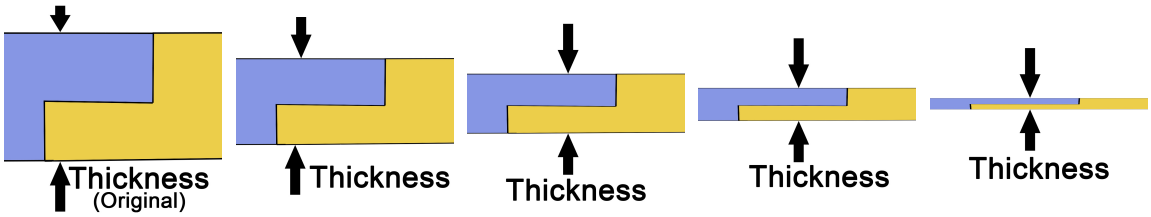
where $\mathcal{O}_k$ is all the overlapping regions of $T(R_i)$. Then, shape transformation M_i can be used to construct embedded connectors since the constraint 3.1 guarantees that chart i will continue to be cellular with no hole; the constraint 3.2 guarantees that there will be no self-intersection in overlapping region; the constraint 3.3 guarantees that charts will completely fill the overlapping region; and the constraint 3.4 guarantees that operation does not change non-overlapping regions. To extend this for external joints, we need to relax the constraint 3.1 to allow holes in overlapping regions. In that case, the charts will not completely fill the overlapping region. We, therefore, need to add external joints in the

constraint 3.3 to fill overalpping regions.

In addition to these constraints, parts in overlapping regions must be interlocked to keep the pieces together. We have identified two types of interlocking strategies as shown in Figure 3.7. Figures 3.7a, 3.7b and 3.7c shows interlocking charts using spiral forms. Note that spiral forms can be used for any number of charts. Figure 3.7b shows teeth structure that is used both zippers and teeth joints. This type of interlocking structures can be used for interlocking only two charts.

3.4 Considering Thickness in Overlapping Regions

Definition of 2-manifold does not assume that the surface has a thickness. However, in physical realizations we always have some thickness even if it is very small. This property can also allow us to assume two or more charts can share a location without intersecting as shown in Figure 3.8. Using this property, we can obtain joints that can provide uniform thickness. In the next chapter (Chapter 4), we will provide details for physical connections.



(a) Thickness = 1 (b) Thickness = $\frac{1}{2}$ (c) Thickness = $\frac{1}{4}$ (d) Thickness = $\frac{1}{8}$ (e) Thickness = $\frac{1}{16}$

Figure 3.8: When the thickness goes to zero, the intersection of the physical panels that corresponds charts continue to be empty. Therefore, we can assume that the two charts can share the location even in the limit case.

4. PHYSICAL CONNECTIONS

In this chapter, I will present my observations of existing physical connectors and its generating procedure in manufacturing and architecture. Although different productions and research declared that they developed unique types of connectors, they can actually be classified based on our theory framework. I will propose a generalized procedure to generate all possible connectors on 2-manifold surfaces, which can be produced easily from the geometric structure.

4.1 Observations

4.1.1 Overlapping Regions

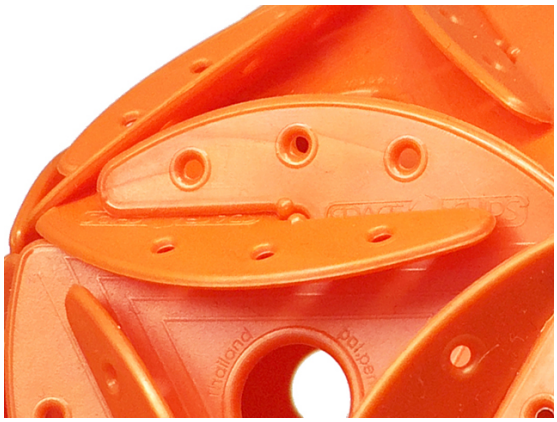
All joints for 2-manifold designs appears only in limited regions, which are called overlapping regions (see Figure 4.1). Connected joints for a 2-manifold design are embedded in a cellular piece without holes, which is homeomorphic to an open disk, regardless of geometry processing algorithms applied to the shapes (see Figure 4.1a and Figure 4.1b). If there are holes in overlapping regions, as long as we can use some external joints to fill the holes, it remains cellular (see Figure 4.1c and Figure 4.1d).

Cases like Zometool (see Figure 4.2) seem to break this rule. However, they form 3-manifolds, where the neighborhood of every point is not always equivalent to an open disk. Therefore, we will not take them into considerations since they do not meet the requirement of 2-manifolds.

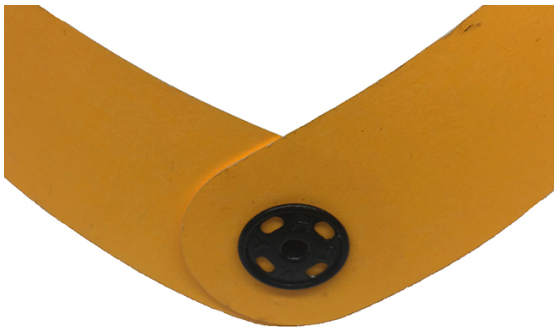
4.1.2 Chart Orientations

Since our goal in this thesis is to construct physical version of orientable 2-manifold surfaces, graph rotation system should be able to represent all shapes we are dealing with. Based on the nature of graph rotation system, it is obvious that we can get the orientation

(a) Conneceted *Flexeez* toy pieces



(b) Connected *Space Chips*



(c) Connected paper strips

(d) Connected quad-edge charts

Figure 4.1: Examples of connected physical panels using physical joints. (a) and (b) are joints embedded in cellular pieces without holes when they are connected to corresponding pieces. (c) and (d), on the other hand, have holes in overlapping regions, but can still form cellular pieces with external joints.

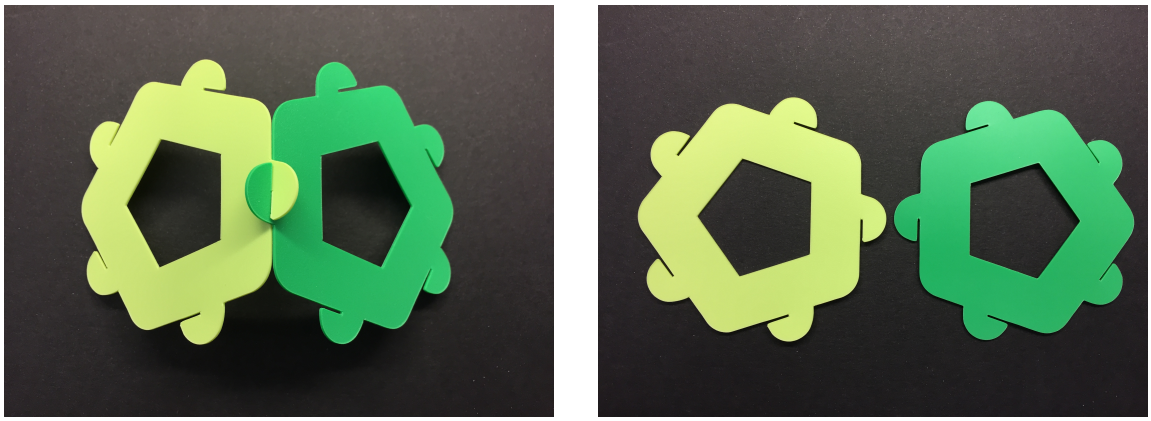
information to synchronize connection orders in all panels.

Correct orientations also help people to understand the structure, and form 3D shapes. The internal links becomes confusing and will make it impossible to build the shape when panels are not properly oriented.

An example of connectors on star-shaped charts indicates the importance of connector orientations. In Figure 4.3a, different panels are able to connect to each other with correct connector orientation. The union of correctly oriented panels forms only one chart in

Figure 4.2: An example of connectors for 3-manifolds from zometool, which is excluded from this thesis. Points over the shape are not always equivalent to an open disk.

overlapping region. However, self-intersection occurs in overlapping region and it fails when connector orientation is flipped (see Figure 4.3b).



(a) Correct Connector Orientation (b) Wrong Connector Orientation

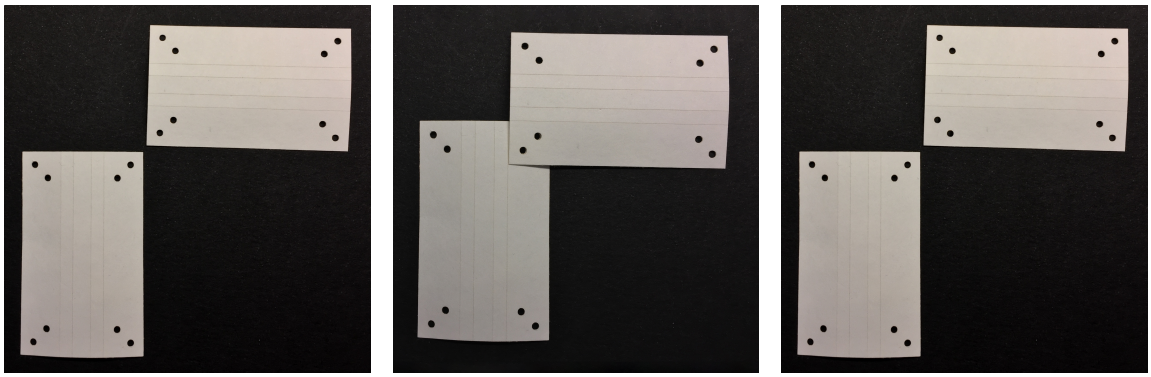
Figure 4.3: An example of different connector orientations on edges in *Space Chips*.

Another example of connectors on quad-edge charts also proves the significance of connection orientation. When the holes are placed following the internal orientation (see Figure 4.4a), it can sufficiently match the current piece with corresponding piece, and external connectors can fill in the holes to form on an open disk(see Figure 4.4b). while flipped orientation cannot connect two pieces together (see Figure 4.4c).

4.1.3 Affected Regions

An interesting fact is that non-overlapping areas will not be changed no matter how we add joints in overlapping regions or what kind of joints we embedded. In other words, connectors only influence the overlapping regions. Within the overlapping regions, we can apply different types of joints and different amount.

Figure 4.5 shows two examples of quad-edge charts, which are generated from cubes. Figure 4.5a and Figure 4.5b are created by different research groups with different amount



(a) Correct Connector Orientation at Corner

(b) Correct Connector Orientation When Merged

(c) Wrong Connector Orientation

Figure 4.4: Different connector orientations at corner.

of joints in each overlapping region. Moreover, their joint layouts are also different from each other. But their non-overlapping regions share the same topologies, quadrilateral charts, which are the set difference of edge-charts and vertex-charts. Another example of quad-edge charts applied to tetrahedron can be found in Figure 4.5c. It also has the same chart topology, since it's generated following the pattern of quad-edge charts.

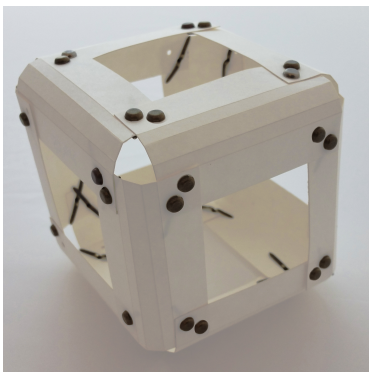
4.1.4 Interlocks

In order to connect counterparts in the same overlapping regions, connectors should be able to stick to each other in some way. In other words, interlocks form in the overlapping regions, which makes the local structure stable enough from distortion.

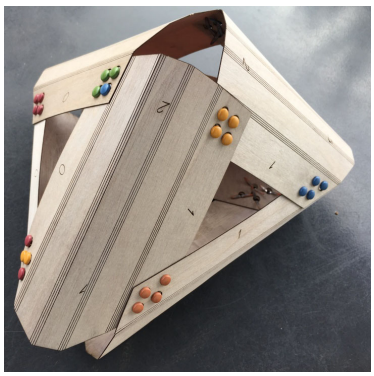
After we designed a set of shape transformation M_i, where $i = 1, 2, \ldots, n$, we apply the transformations to each chart in the overlapping region. Once we put them together, they can hook to each other without extra support.

An example of charts hooking to each other is presented in Figure 4.6. The overlapping region is a thickened edge, and the embedded slit on each side forms two spirally interlocked charts. A 3D version of this kind of spirally interlocked charts can be found

(a) A cube shaped quad-edge charts with 1 pinhole for external fasteners in each overlapping region generated by David Deiman [3]



(b) A cube shaped quad-edge charts with 2 pinholes for external fasteners in each overlapping region created by Akleman et al. [38]



(c) A tetrahedron shaped quad-edge charts with 4 pinholes for external fasteners in each overlapping region created by Akleman et al. [38]

Figure 4.5: Examples of cube shaped quad-edge charts with different numbers of pinholes for external fasteners. The pinhole layouts in the overlapping regions are also different.

in Figure 4.7, where the cellular chart forms a rectangular prism because of the material thickness.

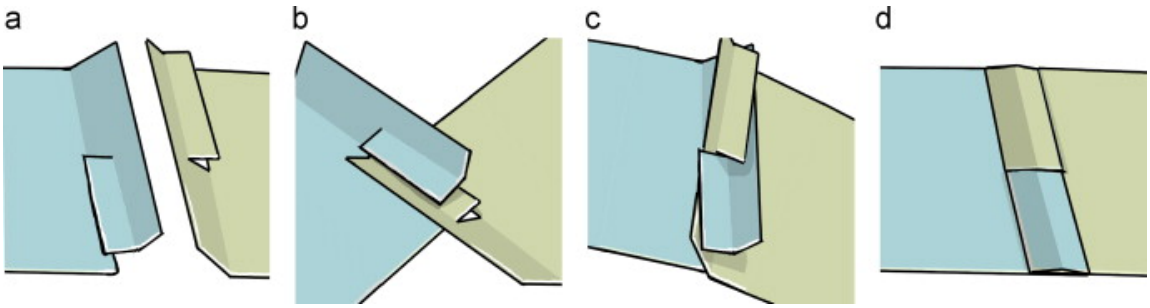
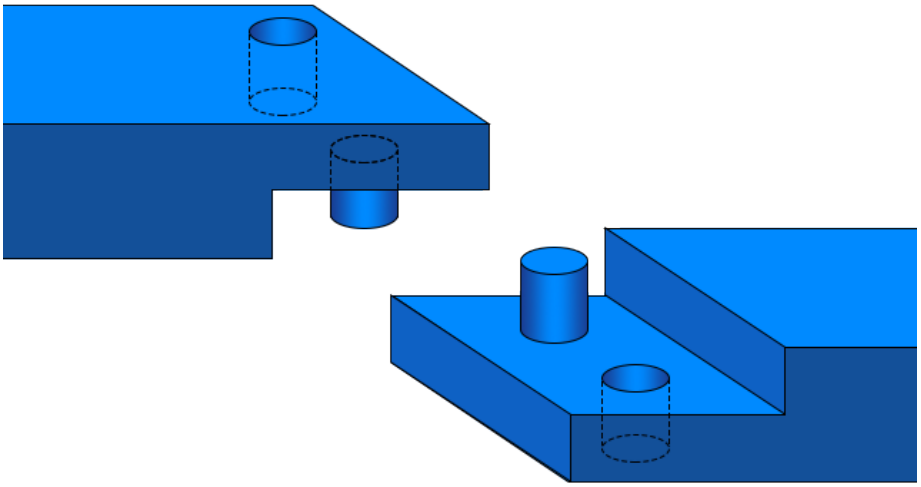
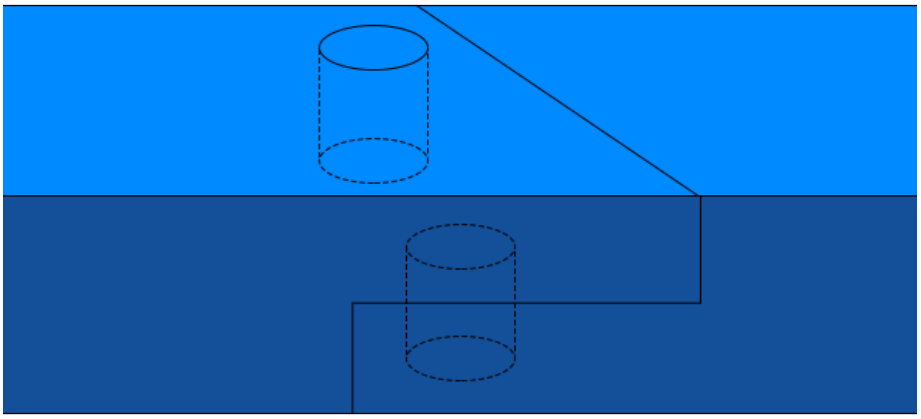


Figure 4.6: An example of spirally interlocked connections on edge charts.

Joints like zippers, however, only have two charts overlapping at each overlapping region. We can use teeth-interlocked charts to connect different panels (see Figure 4.8).



(a) Connectors in reality with thickness



(b) Rectangular prism formed from two spirally interlocked charts

Figure 4.7: An example of stable structure formed by spirally interlocked connectors.

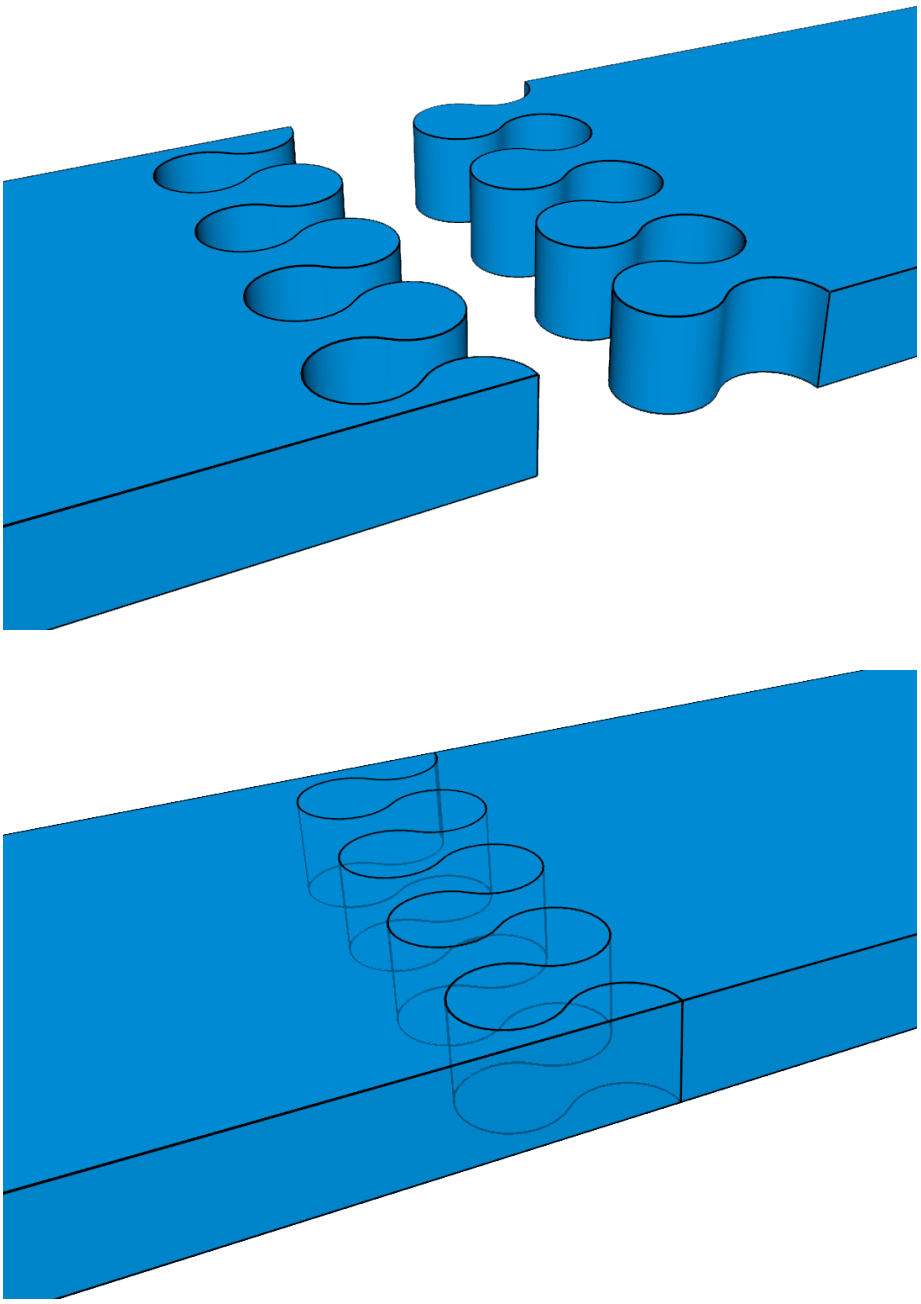


Figure 4.8: Another example of zipper shape connector forming rectangular prism.

4.2 Generalized Procedure

As we have discussed about the theory framework about joints development and the properties of physical connectors in previous chapters and sections, we are able to generalize the procedure of creating connectors from arbitrary 2-manifold designs in three steps as:

1. Geometry Processing
2. Overlapping Region Identification and Generation
3. Charts Transformation

4.2.1 Geometry Processing

A general design for manufacturing or architecture usually store polygonal mesh to store the shape and structure. However, polygonal meshes are not always developable panel, which can be unfolded into 2D plane without distortion. And they don't have overlapping regions for connectors to assemble multiple panels or construct 3D shapes. Therefore, we need to apply certain geometry processing algorithm to create charts.

As explained in Chapter 3, we will apply 2D-thickening methodologies to the input polygonal mesh to create different types of charts for constructing 3D structures. Although the implementation detail can be different, all methodologies are categorized into three cases: (1) Methods use Thickened Edges as Charts;(2) Methods use Star-Shaped Charts; and (2) Method uses Quad-Edge Charts.

4.2.2 Overlapping Region Identification and Generation

After processing the input geometry, we will need to identify overlapping regions from the generated charts for connectors. For charts from thickened edges and star-shaped charts, overlapping regions are automatically generated during geometry processing stage.

All we need to do is to mark those charts, which overlap in the same region, and pass it along to next stage.

For quad-edge charts method, we need to apply one more thickening operations on the quadrilaterals from 2D thickening to obtain overlapping regions. The new created overlapping regions lie in the corners of the quadrilaterals.

With overlapping regions generated and identified, we are able to locate where to create joints and figure out the proper size and amount of joints.

4.2.3 Charts Transformation

The last step is to locate the overlapping regions and find all charts covering the corresponding regions. For each chart overlapping the same region, we need to apply a set of shape transformations $M_1, M_2, \ldots, M_n$, where $n = number \quad of \quad charts$ in the same overlapping region. All transformations should follow from Constraint 3.1 to Constraint 3.4. The generated connectors should also form interlocked charts with either spiral or teeth structure.

After this step, we can deliver the geometric information to machines, like laser cutter, to cut planar materials. The cut panels will then be assembled and construct the 3D shapes.

5. IMPLEMENTATION AND PROCESS

5.1 Platform Development

As in the current system, we are pursuing a general solution to standardize workflow to make it eligible to use graph rotation system to represent 2-manifolds and handle multiple different geometry processing algorithm to create overlapping regions for connectors. Each geometry processing algorithm should be independent from each other. The processing stage should be extensible for further development.

Therefore, a standardized workflow is highly recommended for this system to be delivered to developers. It can also benefit the users to understand how it works to create desired designs.

5.1.1 Basic Workflow

The current system (see Figure 5.1) takes in geometry data from either compiled files or local files on drive as inputs. Geometry data will be stored in certain geometric data structure after read into the system.

When geometry data is properly stored, the system will then apply different geometry processing algorithms to the geometry to create new topology. Regardless of the practical shape of the input geometry, a well-defined geometry processing algorithm should handle the input geometry properly to deliver it to the user.

Then, the system will unfold the processed geometry onto a single plain. So the shape is available to send to laser cutter or other machines for manufacturing.

Before the final output, the system will generate connectors on the unfolded mesh based on the generalized connector model we defined in Section 4.2, and export it as machine-friendly format.

Figure 5.1: Our current system for multiple topological modeling algorithms.

5.1.2 Modern Graphics Pipeline

Based on the requirement, the system need to be designed with independent stages, and standardized input and output. The modern graphics pipeline (see Figure 5.2) has these features, so I introduced the concept into the system we built for our research team.

The data passed through the modern pipeline is restricted into certain type of structure, following some standard.

Each stage of the pipeline doesn't have to know how previous stage is implemented as long as the input and output follows the rules defined in the standard. It is stage independent in this sense.

Figure 5.2: Workflow of modern graphics pipeline.

5.1.3 Mocking The Workflow

As mentioned before, I borrowed the concept of modern graphics pipeline into the system and mocked the workflow to follow a pattern which is widely used in the industry (see Figure 5.3).

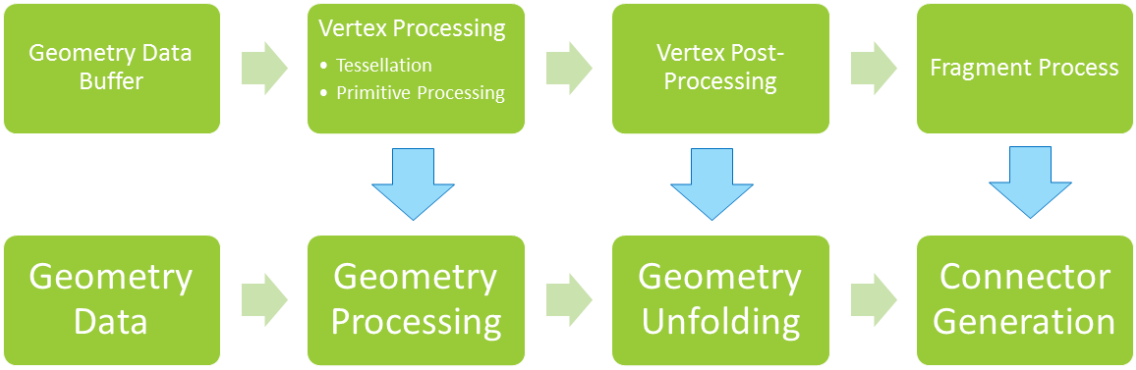


Figure 5.3: Example of how our system mocks the concept of modern graphics pipeline.

The input geometry data in our system can be similar to the geometry data buffer in graphics pipeline. We need certain data structure to store the geometry information as much as we need.

Geometry processing algorithms can be treated as the vertex processing in the graphics pipeline. In the graphics pipeline, vertex processing can be written in different types of shaders, which can be written or extended by different developers. Our system can certainly support extensible geometry processing algorithms developed by others. As long as we expose API to developers, any topological geometry processing algorithms can be added to the system.

The vertex post-processing stage is hidden from users, because it has a fixed algorithm applied to the geometry in order to deliver standardized data stream to the fragment process. The geometry unfolding algorithm in the system is a general algorithm applied to all geometry, which has similar functionality to vertex post-processing.

In my thesis, I am not focusing on the actual geometry processing algorithm or the geometry unfolding process. I will combine these two stages as a black box in this thesis. And the workflow in my thesis will be simplified (see Figure 5.4). The input and output will be my main concern in this thesis.

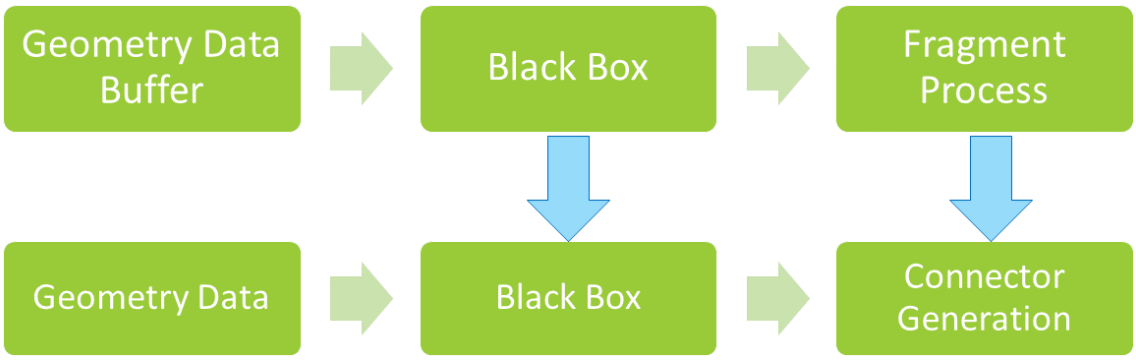


Figure 5.4: Treat geometry processing and unfolding stages as black box in this thesis.

5.2 Geometric Data Structures

With the restriction of 2-manifold surface in the design, an efficient underlying data structure is required to represent the surface to maintain its structure and internal links. Polygon mesh has always been the mainly used geometry representation in computer graphics and turns out to be our first choice in this thesis because of its simplicity of data structure and ability to represent complex shapes. Objects created with polygon mesh can be stored as combination of different objects, which also helps the design to be able to vary from simple to extremely complex. However, there exists several different geometric data structures in computer graphics to represent polygonal mesh and each maintains different information and offers various opportunities in geometric process.

In this thesis, it is required to efficiently represent the internal links between vertices, edges and faces to construct physical connectors. Thus, I will focus on the connection feature of each geometric data structure listed below.

5.2.1 Element List

With basic components of polygonal mesh, an element list (see Figure 5.5) stores the mesh data in several array buffers as underlying data structure. It can efficiently represent

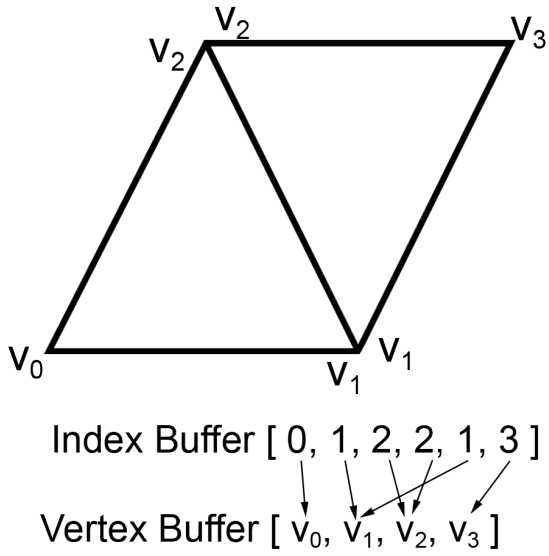


Figure 5.5: An example of how element list stores geometry data.

complex geometry with compact storage.

From the perspective view of hardware, element list is a memory efficient format with great speed performance, which is highly required in real-time computation, such as game development and scientific visualization. Element list implemented by index array can be packed in a compact way, which saves great amount of computer memory. It stores the least information to keep the consistency of geometry.

The downside of element list is also obvious. It is difficult to get adjacent information or accomplish higher end geometry property evaluation, since we only have links from face to vertex. It is not sufficient to represent other links between vertex and edge, and face and edge.

5.2.2 Winged-Edge Data Structure

With limitation of element list, Baumgart first described winged-edge data structure (see Figure 5.6) in his technical report in 1972[39] and refined in 1975[40]. Winged-edge data structure explicitly describes the geometry and topology of faces, edges, and vertices.

To implement wingededge data structure, it requires to store a table for each edge to keep track of adjacent vertices, edges, and faces. The edge table allows for quick traversal from edge to faces, edges, and vertices due to the explicit linked structure. In another word, with winged edge data structure, it is possible to get all touching vertices and all adjacent faces from a given face, and all edge-adjacent vertices from a given vertex.

The winged-edge representation is a little bit more complicated than element list representation and is easy to mess up during mesh processing. What's even worse, the edge itself is bidirectional but the winged edge is oriented. If we want to traversal all edges from a face, we need to take care of the edge orientation. Thus, it is not sufficient to represent a graph rotation system.

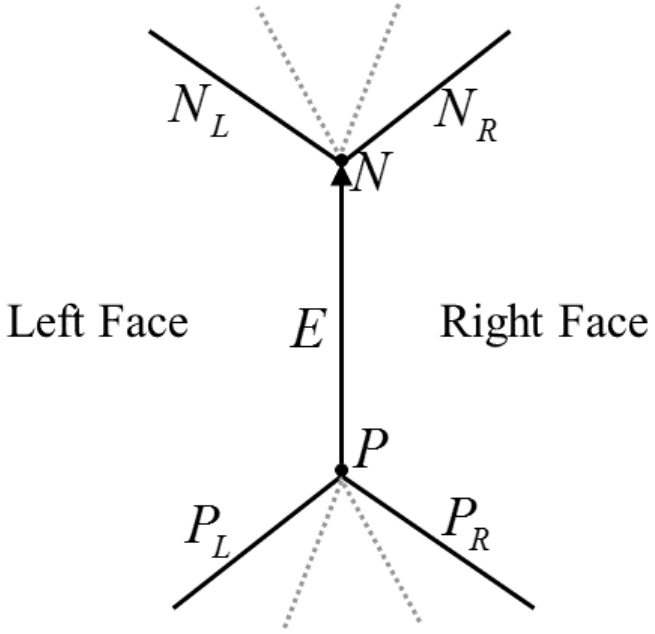


Figure 5.6: An example of internal links represented by winged-edge data structure.

5.2.3 Half-Edge Data Structure

In order to perform all queries from component to component, the half-edge data structure[41] (see Figure 5.7) was invented as one of the more sophisticated boundary representations.

It can sufficiently represent 2-manifold and is widely accepted in computer graphics. It becomes very popular because of its simplicity of constructing internal connection between primitives, and flexibility in traversal the topology.

Compare to element list and winged-edge data structure, half-edge data structure contains full topology information with easy access to traversal the entire mesh. It doesn't take too much extra storage to keep track of the links, which is still memory efficient. Hence, graph rotation system can be easily implemented using half-edge data structure.

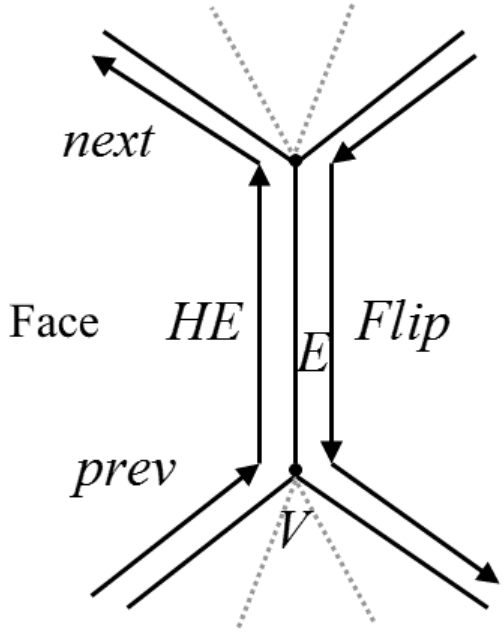


Figure 5.7: An example of internal links represented by half-edge data structure.

5.2.4 Quad-Edge Data Structure

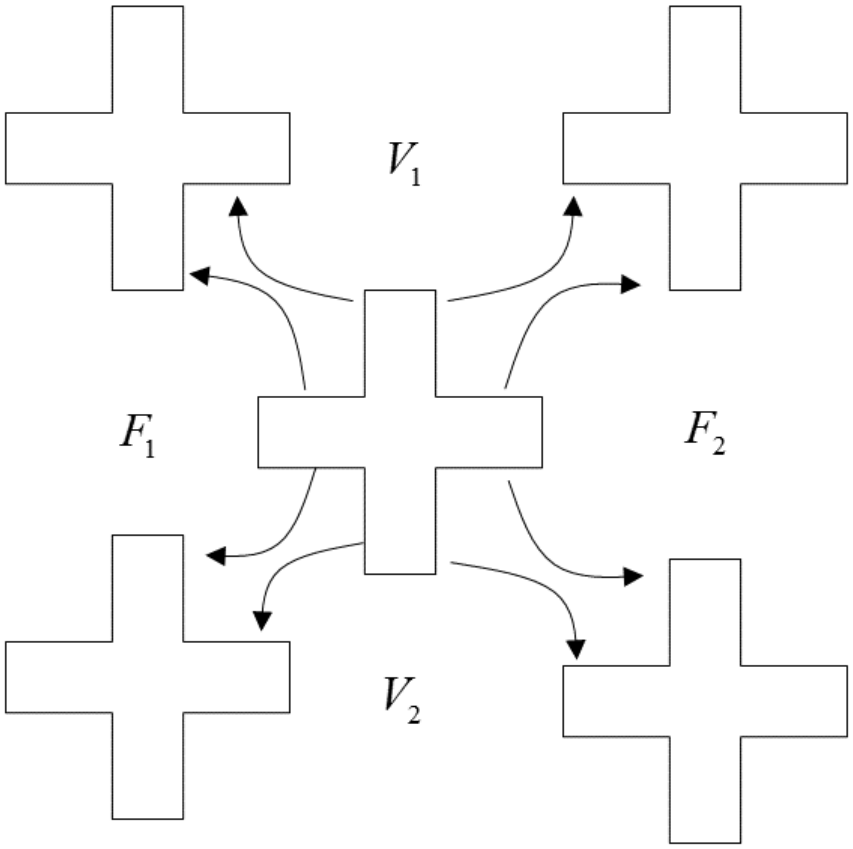


Figure 5.8: An example of internal links represented by quad-edge data structure.

The quad-edge data structure (see Figure 5.8) was first described in the paper by Guibas and Stolfi [42]. Much like winged-edge and half-edge data structure, quad-edge data structure is a minimal upgrade based on previous structures. Edges in this representation is directed and each edge contains links to the four neighboring edges. Vertices and faces are represented by circular linked lists of edges. Thus, it is easier to iterating through the topology.

It can represent graph rotation system, however, quad-edge data structure takes a lot of work to build and update links and requires more memory to fully represent internal links in the geometry.

5.3 Half-Edge Data Structure Implementation

In this thesis, the data structure should efficiently represent graph rotation system in order to build internal links and create charts for connectors. Half-edge data structure offers full traversal functionality over the entire geometry to developers, and its performance is in a good condition compare to other data structures. Therefore, it is obvious to use half-edge data structure as the underlying data structure for the system we build.

5.3.1 Links in Half-Edge Data Structure

In the definition of half-edge data structure (see Figure 5.7), links in between primitives, such as vertex, edge and face, should be established to represent the internal relation among those primitives.

Each edge in half-edge data structure consists of a pair of two half-edges. All the half-edges are directed edge with a beginning point and an end point.

Half-edges in a pair are opposite to each other, whose direction is flipped from the other. In other words, the beginning vertex of a half-edge is the end vertex of its flipped half-edge.

Links between primitives are listed as:

- Each half-edge has a link to its flipped half-edge;
- Each half-edge has a link to its previous half-edge, and a link to its next half-edge;
- Each half-edge has a link to a vertex at its end;
- Each half-edge has a link to a face where it belongs;

- Each vertex has a link to an adjacent half-edge;
- Each face has a link to a half-edge on its boundary.

It is sufficient to traversal the entire geometry from any primitive with these links listed above.

5.3.2 Classical Implementation

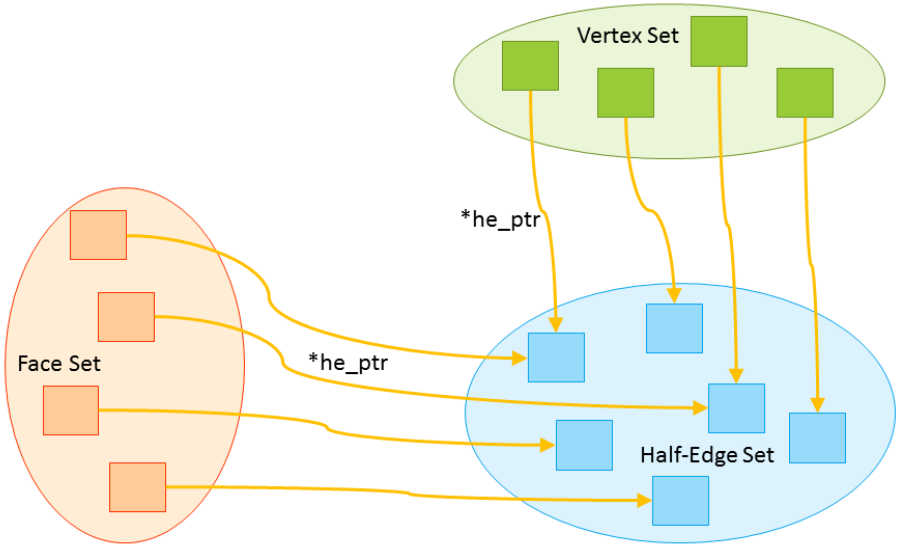
In classical implementation of half-edge data structure, different types of primitive are stored in separate data sets (see Figure 5.9a). The developer can query primitives easily with underlying data structure, such as tree or hash table.

Links between primitives are implemented using explicit pointers, which are the address of data storage on memory. Developers can easily access to different primitives with pointers without additional query method (see Figure 5.9b).

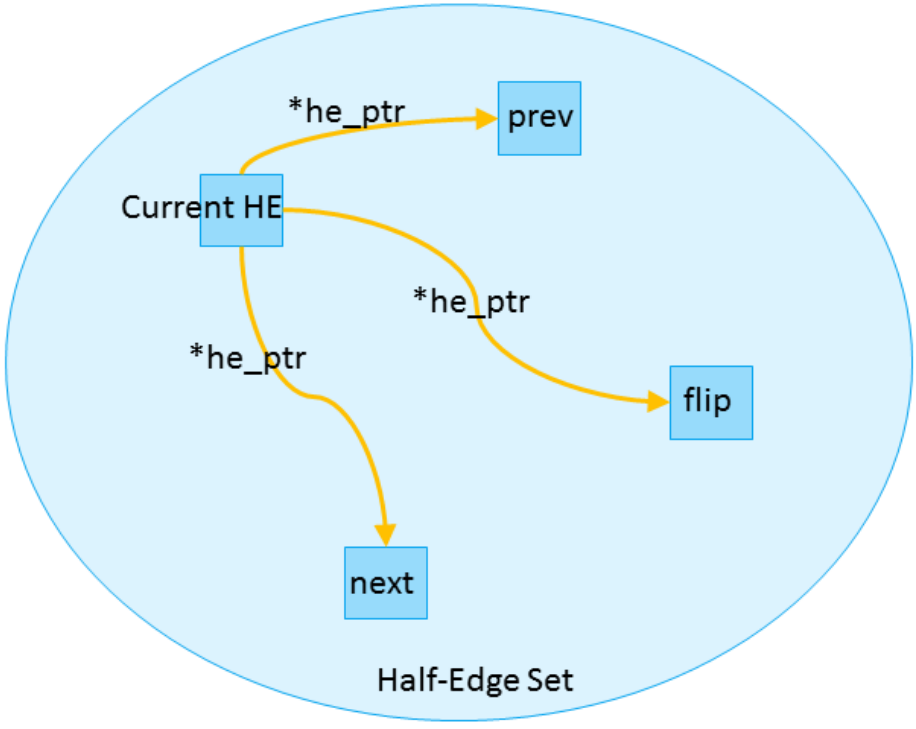
However, problems with using classical implementation are obvious in our system. Since we create new mesh based on reference mesh, we constantly duplicate mesh as reference or output in many operations.

Unfolding operation, for instance, doesn't change the topology of the input mesh. It only changes the vertex position to generate new mesh. Links should keep the same structure in this case. However, Explicit pointers stay the same after duplication. In other words, the links in the duplicated mesh still point to the original mesh (see Figure 5.10). Developers have to take care of the links and update it carefully every time, which can easily leads to fatal errors by mistake in software development and is hard to debug.

Moreover, this implementation may not be memory efficient, since it doesn't require primitives to be tightly packed in storage. It can results in performance issue if the input or output mesh turn out to be complex and in high details.



(a) Links between different types of primitives.



(b) Links between half-edges.

Figure 5.9: Classical implementation of half-edge data structure. Links between different types of primitives are represented as arrow curves.

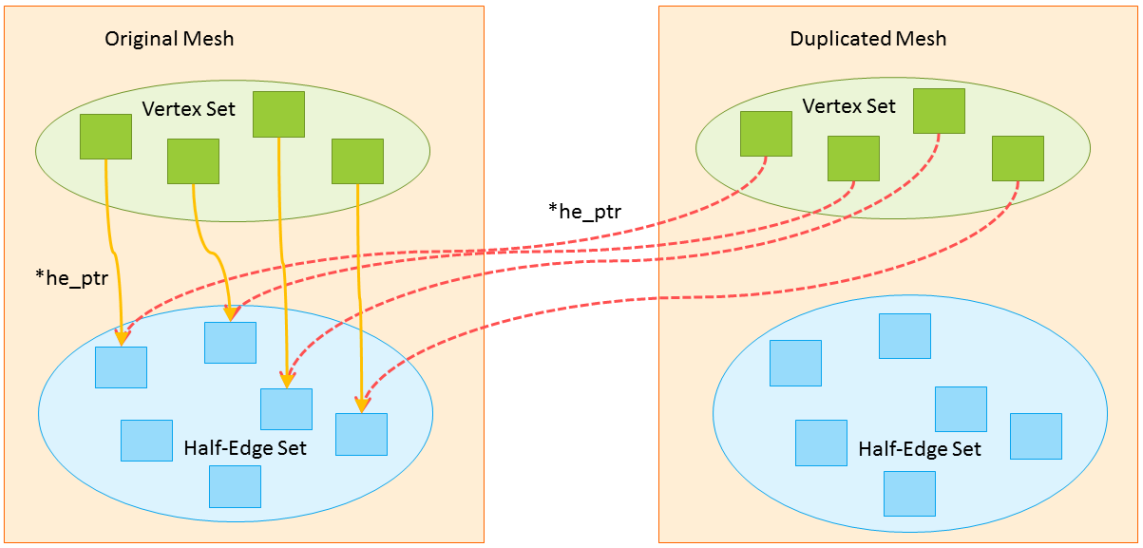


Figure 5.10: Links point to original mesh after duplication in classical implementation of half-edge data structure.

5.3.3 Buffer Array Implementation

To make the data storage memory efficient, we can use a compact array to store primitives instead of storing pointers in sets. It is similar to the technical specification of modern graphics pipeline where data is stored in a buffer array. Based on this idea, my implementation of half-edge data structure will use buffer array as the underlying structure.

It benefits not only the memory efficiency but also make other stages easier. Rendering for example, requires primitives to be stored in continuous memory chunk before it is sent to the graphic card. It improves performance of geometry processing as well, since buffer array can be used directly as an hash table, whose query speed is constant time.

I use primitive unique index rather than the address of data storage to represent links between different types of primitive, and use mesh object to manage the access between them. From mesh level, we have access between half-edge and vertex, and half-edge and face (see Figure 5.11a).

For primitives in same buffer array, half-edges specifically, relative offset is more efficient to represent links (see Figure 5.11b). Connected half-edges can be access directly from current half-edge, since half-edges are strictly aligned in memory by using compact buffer array. The actual implementation can be written as:

$$\boldsymbol{HE_{flip} = HE_{cur} + Offset_{flip}} \tag{5.1a}$$

$$\boldsymbol{HE_{prev} = HE_{cur} + Offset_{prev}} \tag{5.1b}$$

$$\boldsymbol{HE_{next} = HE_{cur} + Offset_{next}} \tag{5.1c}$$

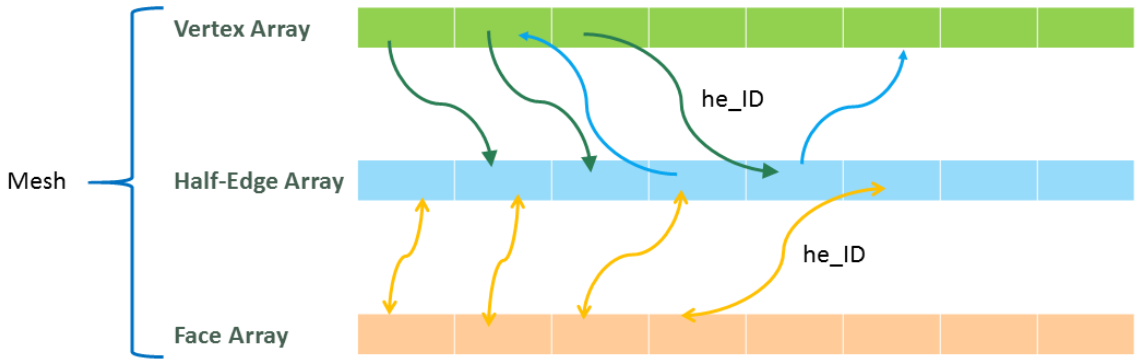
where $\mathbf{HE_{cur}}$ stands for the current half-edge, $\mathbf{HE_{flip}}$ stands for the flipped half-edge with opposite direction from the current one, $\mathbf{HE_{prev}}$ and $\mathbf{HE_{next}}$ stand for the previous and next half-edge connected to the current one in the same face. $\mathbf{Offset_{flip}}$, $\mathbf{Offset_{prev}}$ and $\mathbf{Offset_{next}}$ mean the unique index offset from flipped half-edge, previous, and next half-edge to the current half-edge.

By using this buffer array implementation, the advantages are obvious despite some downsides like indirect access between different primitive types. The data storage is compact and thus memory efficient. Performance of allocating geometry either when loading from local file or duplicating geometry is reduced significantly. Primitive traversal method maintains speed and simplicity. It is safer in development and debugging process without triggering access violation from explicit pointers.

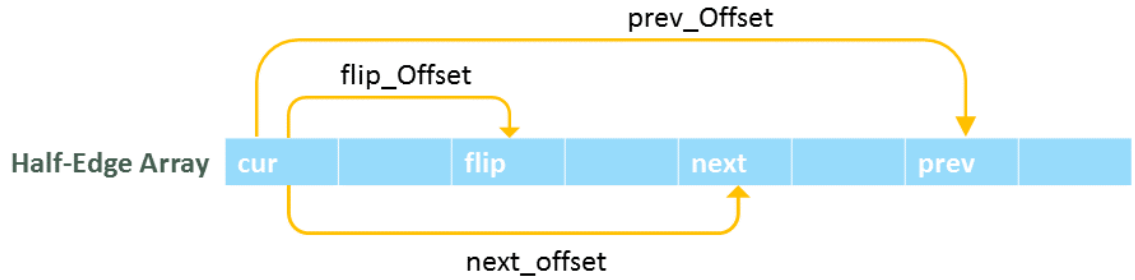
5.4 Additional Information in the Data Structure

5.4.1 Primitives Flags

By thickening different primitives, we can create different charts and tessellate smooth surfaces into developable panels. In order to deliver information from input geometry to

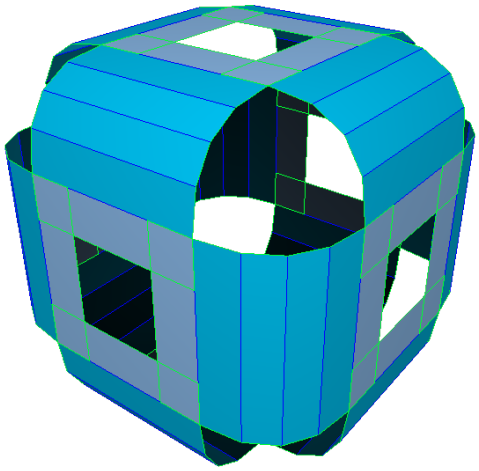


(a) A mesh contains these buffer arrays of primitives and manages accesses between half-edge and vertex, and half-edge and face.

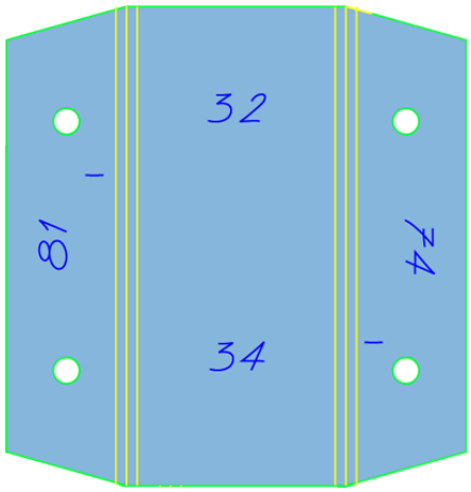


(b) An example of the access from current half-edge to connected half-edges.

Figure 5.11: Half-edge data structure implementation with primitives are stored in compact buffer arrays.



(a) A highlighted model with different colors based on different primitive flags.



(b) An exported planar panel for laser cutter with highlighted colors from flags and labels from reference IDs.

Figure 5.12: Examples of primitive flags and reference IDs in our software to help user with design and manufacturing.

later process, we added flags to primitives to record its type and visualize primitives with different properties or types in an intuitive way. Flags also help to mark the overlapping regions for later operations. In laser cutting stage, it is easier to differentiate the strength and speed of laser based on primitive types, which is marked by flags.

In Figure 5.12a, an example of quad-edge charts is generated from a cube [17] and highlighted with different colors based on the flags assigned to primitives.

Charts boundaries are highlighted as green while other internal edges are shaded dark blue. Overlapping regions lie in gray faces. Curved bridge faces, which are difference set of thickened edges and thickened vertices, are colored in light blue. In connector generation stage, we can easily find the overlapping region and add connectors, place extra lines around internal edges for bending, export the boundary, and etc (see Figure 5.12b).

5.4.2 Reference ID

In order to reconstruct 3D shape from 2D panels, we need to keep track of connections from original mesh, even after several different geometry operations. We introduced reference ID, the primitive ID from input mesh, to each primitive to represent its origin (see Figure 5.12b).

Reference IDs represent links in the original mesh to help user to rebuild the shape with intuitive information. It can also help to calculate overlapping regions and record orientations with other information like primitive flags.

Based on reference ID, the overlapping region, Figure 5.13a for example, can be calculated based on the information. Then we apply shape transformations to the two charts in the corner and it generates connectors with in the overlapping region (see the white circles in Figure 5.13b). Number labels, which represent the primitive ID of the original charts, will be placed in corresponding area for users to rebuild the shape. For instance, number "4" in Figure 5.13b means that the overlapping region was originally part of a face with ID number 4 before edge thickening.

Since we are using graph rotation system in all of our geometry processing algorithms, the orientation of origin primitives can be determined easily. In other words, the processed geometry is also represented by the graph rotation system. We can either trace back to find the orientation of original mesh and figure it out based on reference ID (see Figure 5.13c). As in the case of quad-edge charts, one of the chart in a overlapping region overlays on top of the other. For material with thickness, we mark the bottom chart with "-" in the output panel to specify the order. This information can be used to carve out redundant material in overlapping regions to avoid self-intersection (see Figure 5.13d).

Moreover, we can easily trace back to get the internal link from the original mesh and marked it with reference ID. Quad-edge charts discard vertex charts overlapping with

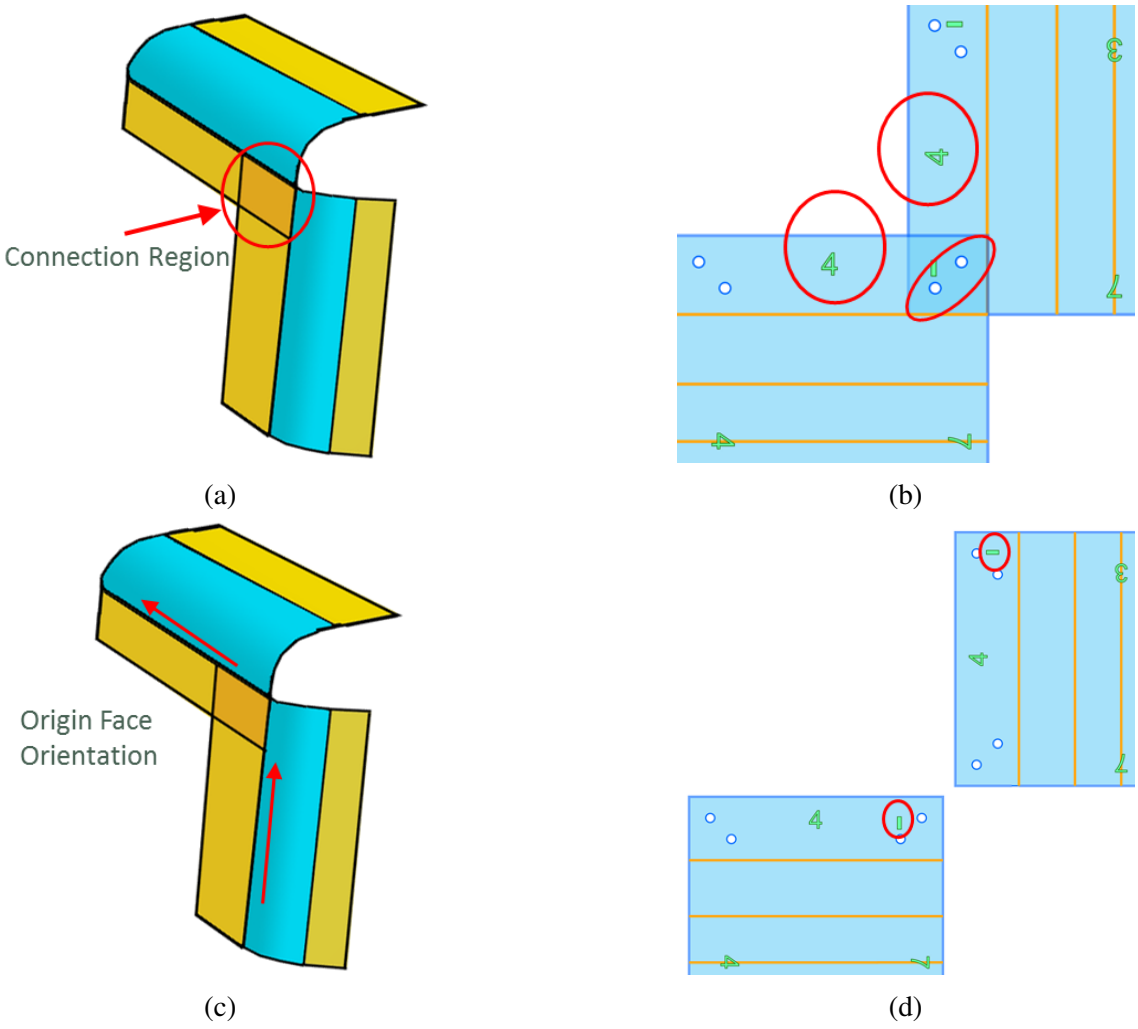
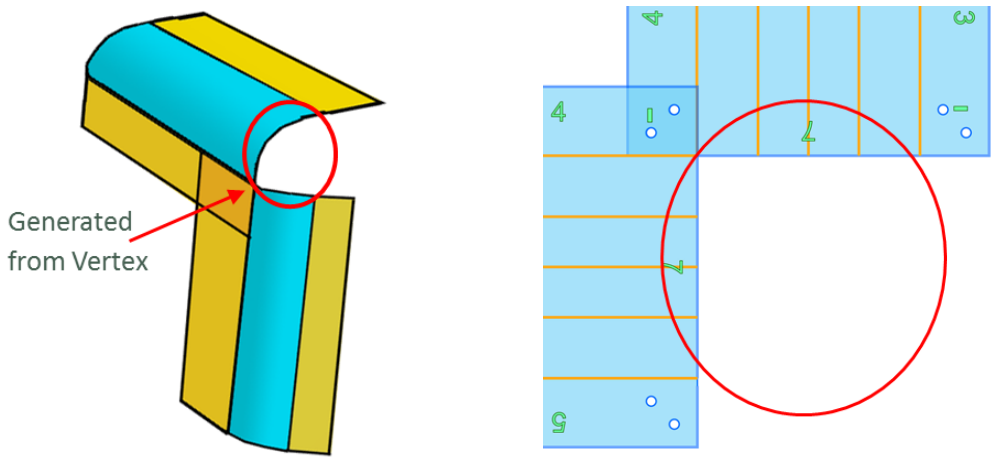


Figure 5.13: Examples of functionalities of reference IDs in our system. On the left are shapes in 3D structure and on the right are unfolded panels with connectors and labels.

thickened edge charts to create multiple developable panels (see Figure 5.14a). Reference ID helps to record this opperation and we can label the vertex chart boundaries with vertex ID from input vertex (see the number "7"s near the boundaries Figure 5.14b).

5.5 Output Geometry Data Format

We have been explaining and developing the geometry and links in virtual world we built. To bring the design from computer to real world, we need to export the geometric



(a) (b)

Figure 5.14: An example of how reference IDs represent links in the original mesh. (a) are quad-edge charts in 3D structure, and (b) are unfolded panels with connectors and labels.

data to some laser cut supported format.

Adobe Illustrator and AutoCAD file format are well supported in current manufacturing industry as input data formats. However, to support exporting to these formats, we need to add third party libraries to the system.

We used scalable vector graphics, known as SVG for short, as our output format. It is an XML based plain text format and maintains all geometry information without data loss. It is very flexible to develop in our current system without any third party reliance library, which makes it platform free. It is well supported by web browsers, Adobe Illustrator, AutoCAD and etc. We can easily import it into other software for further modification with minor or even no changes.

5.6 Connector Generator

To make the unfolding and exporting process user friendly, a graphic user interface is included in the system. Options like shape transformation operation (we call "component type" in the interface), label size and font, component scales are available to users.

Some additional features like adding extra etch lines to make bending easier and changing etch line types are implemented in this system as well to improve user experience (see Figure 5.15).

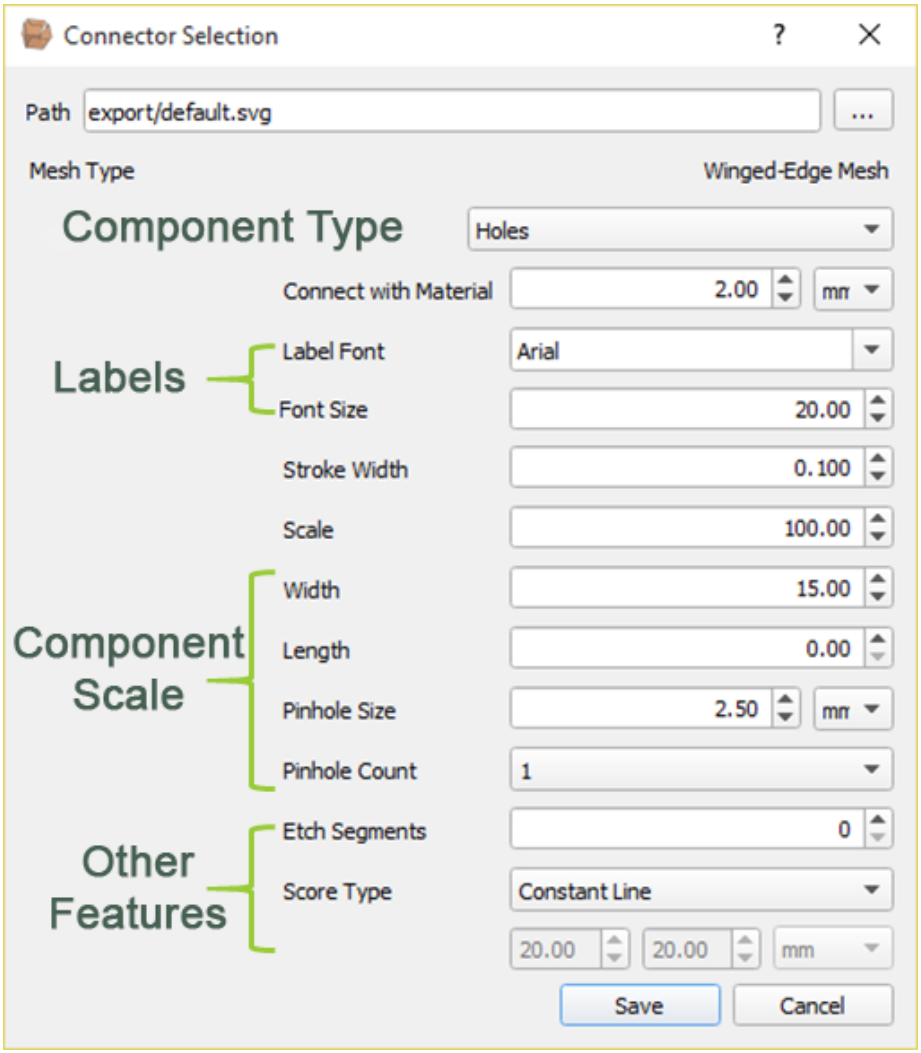


Figure 5.15: Links point to original mesh after duplication in classical implementation of half-edge data structure.

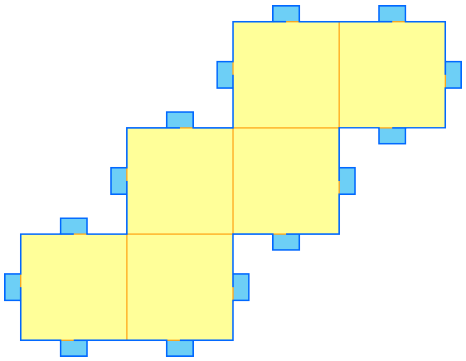
6. RESULTS

The unfolding system we've been developing can create and export developable single or multiple panels. By using the classification, we can add physical connectors to each panel for constructing 3D shapes.

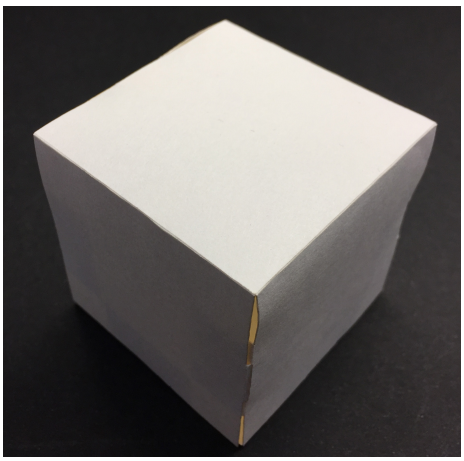
A simple example of exported files from our system can be found in Figure 6.1a. After we send the exported vector graph to laser cutter, we will get physical panels with connectors on edge-side. The final 3D shape is constructed easily from the panel (see Figure 6.1b).

More examples of basic shapes with various processing algorithms are listed in Figure 6.2. We used simple shapes for early test on the theory. After that, we built more complex shapes, such as a 3-genus shape in Figure 6.3, to enhance the theory. We also built a human-sized Stanford Bunny (see Figure 6.4).

We delivered this research to architects with our software. It inspired their creativity. They built some amazing works with complexity in shape and stability in structure. The system is well accepted among those architects. Their work approves that this classification could successfully create physical connectors to assemble planar panels to large 3D shapes (see Figure 6.5).

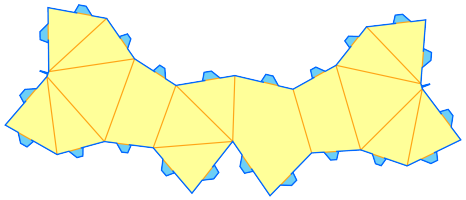


(a)

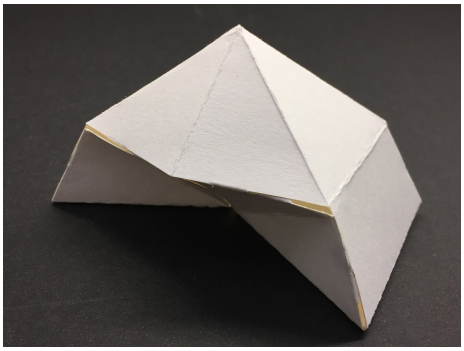


(b)

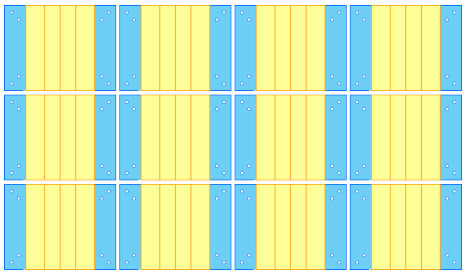
Figure 6.1: A simple example of exported panel using rotation graph system.



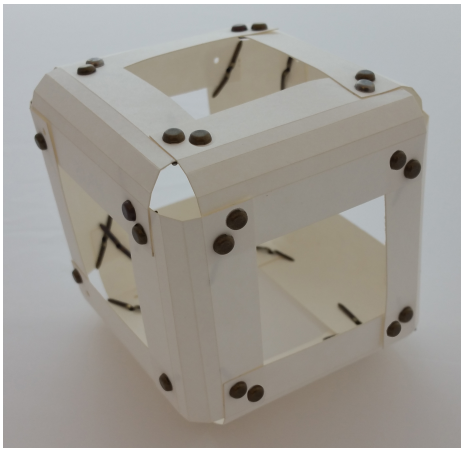
(a)



(b)



(c)



(d)

Figure 6.2: Examples of exported panels (on the left) and physical 3D shapes. Physical connectors are highlighted as blue in the exported files.

Figure 6.3: A 3-genus shape example using our classification.

Figure 6.4: A large scale 3D Stanford Bunny built using quad-edge charts.

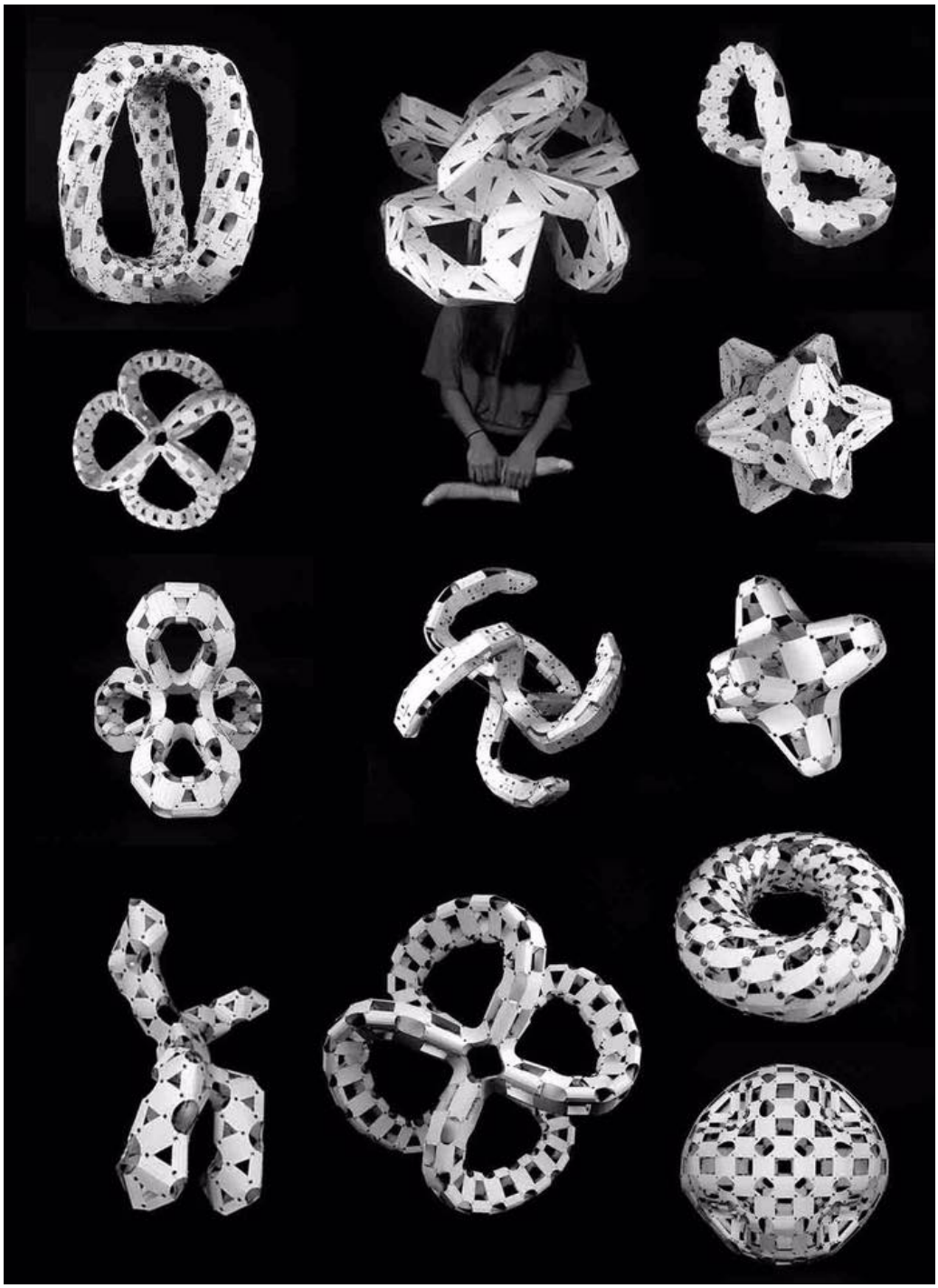


Figure 6.5: Large 3D shapes built by students from College of Architecture at Texas A&M University advised by Professor Negar Kalantar using our system. Photo by Negar Kalantar, August 2016.

7. CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this research, my goal is to identify non-permanent physical connectors to represent the internal links in geometric data structure that are used in shape modeling. We can use these physical connectors to assemble unfolded planar panels for constructing large shapes.

By using our classification of physical connectors, quantities of large shapes were successfully constructed. The variety of the design proved that our method is robust as a general solution to connector generation process.

Physical connectors generated by our method can assemble large shapes easily with the help of reference ID and panel orientations from the graph rotation system. Pieces stay tight with each other, and maintain local stability with our accurate overlapping region location mechanism. Self-intersection and distortion can be avoided efficiently since we strictly follow the constraints for generating connectors in overlapping regions.

Our software is user-friendly and extensible for more geometry processing algorithms. Developers can keep coming up with new geometry processing algorithms under our classification and create connectors with minor change in code.

7.2 Further Works

Opportunities for future works include developing more geometry processing algorithms to create various of topologies. We successfully added physical connectors to construct orientable 2-manifold shapes. We will work on constructing physical connectors for non-orientable 2-manifold surfaces, such as woven structure. Connection construction for origami-shape objects will be another direction for us to work on.

We will improve the software by doing user study and collecting feedback from users.

It is very important in this research to collaborate with designers, architects and people from other disciplines to get constant inspiration. We will improve the information display module to make intermediate process information available to user if they request.

REFERENCES

[1] Q. Xing, G. Esquivel, E. Akleman, J. Chen, and J. Gross, "Band decomposition of 2-manifold meshes for physical construction of large structures," in *ACM SIGGRAPH 2011 Posters and Talks*, p. 58, ACM, 2011.

[2] E. Akleman, J. Chen, and J. L. Gross, "Paper-strip sculptures," in *Shape Modeling International Conference (SMI), 2010*, pp. 236–240, IEEE, 2010.

[3] D. A. Reimann, "Snub polyhedral forms constructed from flexible 60-120 degree rhombic tiles," in *Proceedings of Bridges 2016: Mathematics, Music, Art, Architecture, Education, Culture* (C. S. D. M. K. F. Eve Torrence, Bruce Torrence and R. Sarhangi, eds.), (Phoenix, Arizona), pp. 443–444, Tessellations Publishing, 2016. Available online at `http://archive.bridgesmathart.org/2016/bridges2016-443.html`.

[4] J. Miller and E. Akleman, "Edge-based intersected polyhedral paper sculptures constructed by interlocking slitted planar pieces," in *Bridges Leeuwarden: Mathematics, Music, Art, Architecture, Culture*, pp. 259–264, Tarquin Publications, 2008.

[5] D. Esterle, "Space chips, [polyhe]dron construction kit," 2011.

[6] "Itsphun, geometeric construction kit."

[7] T. Wills and J. Sharp, "Dforms: 3d forms from two 2d sheets," *Bridges: Mathematical Connections in Art, Music, and Science, Reza Sarhangi and John Sharp, eds, London*, pp. 503–510, 2006.

[8] E. D. Demaine, M. L. Demaine, A. Lubiw, A. Shallit, and J. L. Shallit, "Zipper unfoldings of polyhedral complexes," *MIT Open Access Article*, 2010.

[9] J. L. Gross and T. W. Tucker, *Topological graph theory*. Courier Corporation, 1987.

[10] P. K. Agarwal, E. Flato, and D. Halperin, "Polygon decomposition for efficient construction of minkowski sums," *Computational Geometry*, vol. 21, no. 1-2, pp. 39–61, 2002.

[11] E. A. P. Hernandez, S. Hu, H. W. Kung, D. Hartl, and E. Akleman, "Towards building smart self-folding structures," *Computers & Graphics*, vol. 37, no. 6, pp. 730–742, 2013.

[12] H. Q. Dinh, F. Gelman, S. Lefebvre, and F. Claux, "Modeling and toolpath generation for consumer-level 3d printing," in *ACM SIGGRAPH 2015 Courses*, p. 17, ACM, 2015.

[13] H. Michele Moorefield-Lang, "Makers in the library: case studies of 3d printers and maker spaces in library settings," *Library Hi Tech*, vol. 32, no. 4, pp. 583–593, 2014.

[14] H. Yoshida, T. Igarashi, Y. Obuchi, Y. Takami, J. Sato, M. Araki, M. Miki, K. Nagata, K. Sakai, and S. Igarashi, "Architecture-scale human-assisted additive manufacturing," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 88, 2015.

[15] S. Lim, R. A. Buswell, T. T. Le, R. Wackrow, S. A. Austin, A. G. Gibb, and T. Thorpe, "Development of a viable concrete printing process," *International Association for Automation and Robotics in Construction*, 2011.

[16] C. L. Caristan, *Laser cutting guide for manufacturing*. Society of manufacturing engineers, 2004.

[17] E. Akleman, S. Ke, Y. Wu, N. Kalantar, A. Borhani, and J. Chen, "Construction with physical version of quad-edge data structures," *Computers & Graphics*, vol. 58, pp. 172–183, 2016.

[18] D. R. Shelden, *Digital surface representation and the constructibility of Gehry's architecture*. PhD thesis, Massachusetts Institute of Technology, 2002.

[19] G. Hart, "'slide-together' geometric paper constructions," in *Teachers' workshop at Bridges Conference*, 2004.

[20] G. Hart, "Symmetric sculpture," *Journal of Mathematics and the Arts*, vol. 1, no. 1, pp. 21–28, 2007.

[21] G. Hart *et al.*, "Modular kirigami," *Proceedings of Bridges Donostia*, pp. 1–8, 2007.

[22] J. Sharp and T. Wills, "D-forms and developable surfaces," *Bridges: Mathematical Connections between Art, Music and Science*, pp. 121–128, 2005.

[23] E. D. Demaine and G. N. Price, "Generalized d-forms have no spurious creases," *Discrete & Computational Geometry*, vol. 43, no. 1, pp. 179–186, 2010.

[24] Ö. Gönen, E. Akleman, V. Srinivasan, *et al.*, "Modelling d-forms," *Bridges: Mathematical Connections between Art, Music and Science*, pp. 209–216, 2007.

[25] K. A. Seaton, "Sphericons and d-forms: a crocheted connection," *arXiv preprint arXiv:1603.08409*, 2016.

[26] E. D. Demaine and J. O'Rourke, *Geometric folding algorithms*. Cambridge university press Cambridge, 2007.

[27] J. Sharp, *D-Forms: Surprising New 3-D Forms from Flat Curved Shapes*. Tarquin Publications St. Albans, 2009.

[28] J. Chen and E. Akleman, "Topologically robust mesh modeling: concepts, data structures, and operations," *Int. J. Shape Model*, vol. 5, no. 2, pp. 149–177, 2000.

[29] P. A. Firby and C. F. Gardiner, *Surface topology*. Elsevier, 2001.

[30] C. M. Grimm and J. F. Hughes, "Modeling surfaces of arbitrary topology using manifolds," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 359–368, ACM, 1995.

[31] C. Grimm and J. Hughes, "Parameterizing n-holed tori," in *Mathematics of Surfaces*, pp. 14–29, Springer, 2003.

[32] L. Heffter, "Über das problem der nachbargebiete," *Mathematische Annalen*, vol. 38, no. 4, pp. 477–508, 1891.

[33] J. Edmonds, "A combinatorial representation of polyhedral surfaces," *Notices of the American Mathematical Society*, vol. 7, 1960.

[34] E. Akleman and J. Chen, "Guaranteeing the 2-manifold property for meshes with doubly linked face list," *International Journal of Shape Modeling*, vol. 5, no. 02, pp. 159–177, 1999.

[35] E. Akleman, J. Chen, and J. L. Gross, "Block meshes: Topologically robust shape modeling with graphs embedded on 3-manifolds," *Computers & Graphics*, vol. 46, pp. 306–326, 2015.

[36] J. L. Gross and J. Yellen, *Graph theory and its applications*. CRC press, 2005.

[37] D. Burago, Y. Burago, and S. Ivanov, *A course in metric geometry*, vol. 33. American Mathematical Society Providence, 2001.

[38] E. Akleman, S. Ke, and Y. Wu, "Physical mesh data structures," in *ACM SIGGRAPH 2016 Talks*, p. 9, ACM, 2016.

[39] B. G. Baumgart, "Winged edge polyhedron representation," tech. rep., DTIC Document, 1972.

[40] B. G. Baumgart, "A polyhedron representation for computer vision," in *Proceedings of the May 19-22, 1975, national computer conference and exposition*, pp. 589–596, ACM, 1975.

[41] K. Weiler, "The radial-edge structure: A topological representation for non-manifold geometric boundary representations," *Geometric Modelling for CAD Applications*, pp. 3–36, 1988.

[42] L. J. Guibas and J. Stolfi, "Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams," *ACM Transactions on Graphics (TOG)*, vol. 4, no. 2, pp. 75–123, 1985.