Chapter 3

Filling - Filling Holes

There are many ways to generate polygonal surfaces by computer software. 3D scan technique and CAD software are both good tools that can be used to produce polygonal mesh models. Although we can produce polygonal mesh model by the above mentioned tools, due to the improper operation or limitation of these tools, it is possible that the generated models contain holes. When a generated model contains holes on it, guilt a number of existing mesh-based algorithms that are designed for processing 3-D meshes may fail due to this kind of defects. On the other hand, a mesh model containing holes looks visually imperfect. Therefore, it seems to have strong reasons to fill the holes of a mesh model if they do exist.

The types of holes that exist in a 3D object may be variant. For the purpose of filling different types of holes, different strategies need to be applied []. Among different types of holes, the simplest hole is the hole having cyclic form. Facing this kind of hole, we can simply use a patch or apply a surface approximation algorithm to fill it. However, the hole filling issue is not a must for many real world applications. Generally speaking, the hole-filling problem can be categorized as a geometry problem in the field of mathematics. In what follows, we shall provide a brief survey on related problems.

3.1 Related Work

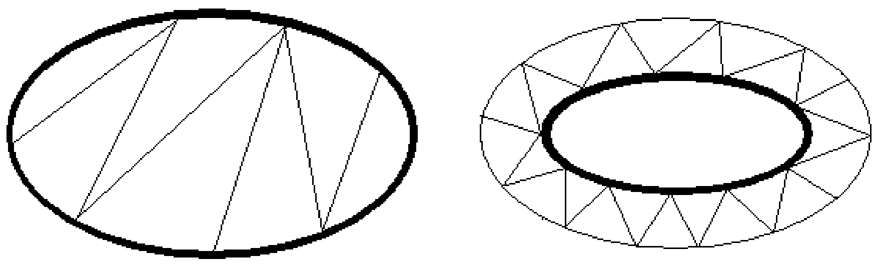
Since a 3D model may contain holes and these defects will make a 3D model reconstruction process incomplete, many researchers have devoted themselves to solve this critical issue. In [5], Sharf et al. propose the ”Context-based surface completion” method to tackle this problem. They transformed the input into a set of points and then derived an approximated surface by calculating the cloud formed by these points. The above mentioned approximation process is conduced through a coarse-to-fine manner. After getting the best patches of holes, they iteratively apply the patches on the holes with rigid and non-rigid transformation. In contrast to filling holes with smooth patches, the context-based method locates the neighboring regions of holes in the input model and then completes the filling process. Although this method can make holes of an object patched, it doest not address geometry-related issues. In other words, it is possible for this method to find similar skins from the surrounding area, but may not find meaningful patches for holes.

Mesh model usually contains many fragmentized pieces of a polygonal surface. Facing this kind of problem, Ju [3] uses the concept of dual graph to transform the original mesh data into a new domain. In the transformation of dual graph, the original faces become vertices and if two faces have a neighboring relation, there will be a corresponding edge between two vertices that are transformed from the above two faces. They execute a model-repairing process using scan-conversion, sign generation and surface reconstruction step by step.

Barequet and Sharir [1] proposed to fix the gaps of a mesh model along the boundary of polyhedron. They find the matched boundaries by a voting technique and stitch these matched boundaries. However, this technique cannot guarantee that all gaps are fixed. Therefore, they apply a triangulation algorithm to fix the remaining parts. The triangulation algorithm that they adopted uses dynamic programming to locate some patches with minimum areas to fill holes. However, when the algorithm meets the effect of crenellation, the resultant outcome is not satisfactory. Under these circumstances, Liepa [4] added an angle parameter to the original weighting function to improve the performance of the minimum area triangulation algorithm. In this work, we shall adopt the approach proposed by Liepa [4] to generate a hole-free mesh model. Liepa's approach includes four steps. They are hole identification, hole triangulation, refinement and finally fairing. Before describing the process of hole-filling, we shall briefly introduce some definitions that will be used in the subsequent sections.

3.2 Definition and Discussion

Our input is a triangular surface. A triangular surface is defined as a set of vertices and a set of faces that form this surface. If two triangular faces share a common edge, we call they are adjacent faces. Two adjacent triangular faces are said to be consistently oriented if



(a) Outer hole (b) Inner hole

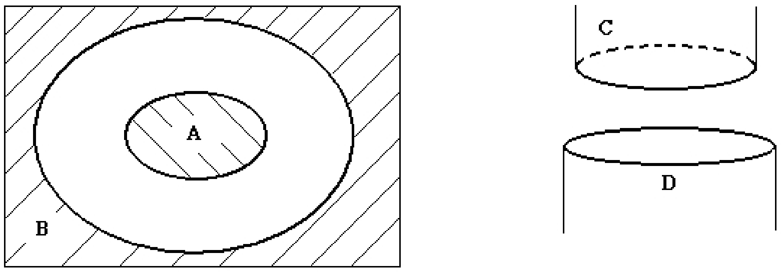
Figure 3.1: Hole Example (1)

their borders traverse their common edge in opposite direction. The dihedral angle of two oriented faces is the angle of their face normals.

In Chapter 2, we make the input of a triangular surface become a triangular surface that does not contain singular vertices and isolated singular vertices. The reason why we have to do this is that the hole-filling algorithm does not accept an input that contains topological singularities. Despite of singular vertices or isolated singular vertices, their existence will make the hole-filling method crashed. On the other hand, a boundary edge is defined as an edge that is adjacent to exactly one face. A hole is a closed cycle which is composed of a set of neighboring boundary edges.

After executing the hole-filling process, we can say that the result is hole-free from the view point of topology if we do not care whether the input is consistently oriented and manifold. But from the view point of geometry, the hole does exist. For this reason, we limit the input of a triangular surface to be consistently oriented. Under the circumstance, we can guarantee to generate better experiment results.

The definition of a hole is entirely topological. Due to this definition, when executing triangulation algorithm, the self-intersection effect may occur. Additionally, while using this definition, we usually do not know whether the hole is the right one to be filled. The strategy of hole-filling is to apply the triangulation algorithm to all the holes that are iden-



(a) Isolated island (b) broken pipe

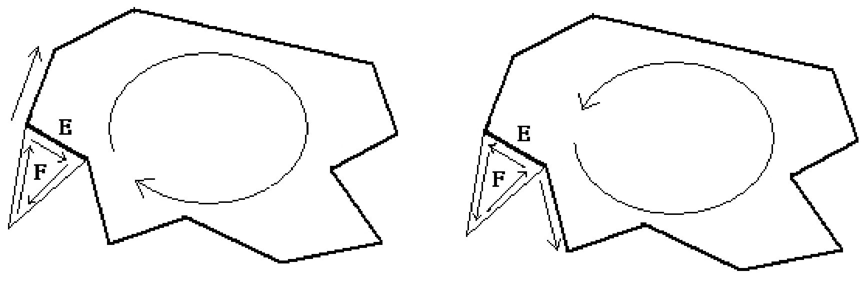
Figure 3.2: Hole Example (2)

tified. It cannot automatically judge which one should be filled and which one should not be filled. In Fig. 3.1(a), the thick cycle is an outside hole, and sometimes we do not need to fill it. However, for the care of Fig. 3.1(b), the inside hole is definitely needed to be filled.

When we only handle one hole, it seems that a triangulation algorithm is a good choice for hole-filling. However, if two or more holes are geometrically related, we should take the following situations into consideration. As shown in Fig. 3.2(a), there is an isolated island A inside the hole of B. According to the strategy of filling one hole, we need to execute the triangulation algorithm twice on this mesh model: one for the hole A, and one for the hole B. After executing the triangulation algorithm, we will have two hole-free connected components. As a matter of fact, it is worthy of a discussion on this situation. Fig. 3.2(b) shows a broken pipe having two holes (C and D). The best solution to the hole-filling issue is to connect them. However, this kind of case is rare and can be considered as a special case. In this work, we shall put our emphasis on filling a single hole.

3.3 Hole-Identification

In order to identify every hole of a mesh model, we must start the process from the boundary edges. First of all, we label every boundary edge that has only one adjacent face. From the pool that keeps all the boundary edges, we select a boundary edge as the pivot



(a) Clockwise direction (b) Counterclockwise direction

Figure 3.3: Hole Identification

and then find the next neighboring boundary edge.

However, a boundary edge in a hole always has two neighboring boundary edges. We have to make decision which one to choice. Choosing the right direction, we can get the correct normals of faces. To solve this problem, we have to make use of the information of the face which is adjacent to the boundary edge. One key to the success of this section is that the input model must be consistently oriented.

For example, Fig. 3.3 illustrates, the boundary edge E is the pivot edge that we choose, the face F is the adjacent face of E, and the hole is inside the cycle. Fig. 3.3(a) and Fig. 3.3(b) show two different directions of face F, thus resulting in different rotating manners, e.g., counter-clockwise or clockwise. The output mesh model will be consistently oriented.

A most important thing that should be concerned about is that there can be no topological singularities in an input mesh model. Under the circumstances, the hole-identification process will face a serious problem if the above mentioned problem exists. In general, the definition of a hole is a closed cycle which is composed of boundary edges. If one wants to traverse the constituent boundary edges of a hole, he/she will encounter difficulties when there exists an isolated singular vertex in the hole. In order to guarantee the success of the hole-identification process, we shall apply the method proposed in chapter 2 to remove all topological problems.

3.4 Hole Triangulation

After completing hole-identification, the next step is to find a triangulation method to fill these identified holes. For a given 3D closed polygonal curve, we can always find a minimum-area triangular surface to fill it by using the dynamic programming technique. It is well-known that the adjustment of weight functions plays an important role in a dynamic programming process. For perfectly filling a hole, a dynamic programming process usually requires to balance the weight functions between two triangular faces. The most commonly adopted parameter that can be used to adjust the weight functions of two neighboring triangular faces is their areas. There are two ways to calculate the area of a triangular face. The first is to utilize the two vectors formed by three vertices of a triangular face. One can take the cross product of these two vectors and measure its length. The magnitude of this length can be used as a reference to determine the weight. However, the computation time consumed in this process is extensive. The second way to determine the weight is to use the total length of the three edges. However, this method is not as accurate as the first method. In addition to the area feature, it is also possible to use the dihedral angle between two triangular faces as a judging criterion. Usually, the minimum area triangulation method is suitable for holes that do not have crenellations. However, if the above mentioned conditions exist, the algorithm is not feasible. Hence, we combine the area and the dihedral angle together to determine the coefficient of the weight function.

The new weights on the triangles are defined as an ordered pair (angle, area). The following equation is a test function for new weight

$$L_{new} = [0, \pi] * [0, \infty) \text{ ,with } 0_{Lnew} = (0, 0),$$

where ”*” links the two range of angle and area, respectively.

Once we have the values of weight, an important thing is to pre-determine which one of the two weights is more important. In this work, we consider the angle is more important than the area. Therefore, the comparison criteria should be as follows:

$$(a, b) < (c, d) \text{ if and only if } a < c \text{ or } (a = c \text{ and } b < d)$$

Furthermore, the addition operator sums the areas but retains the largest dihedral angle:

$$(a, b) + (c, d) := (\max(a, c), b + d)$$

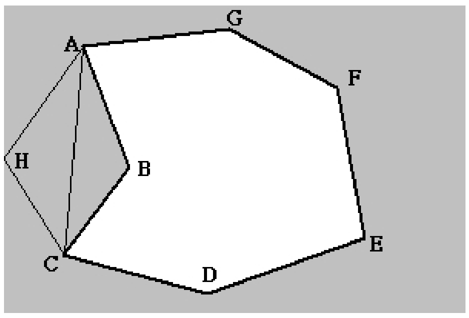
Usually, a triangulation algorithm operates on every known hole that is composed of a sequence of vertices v_1, v_2, v_3, ..., v_{n-1}. Define a weight function Ω $V^3 \rightarrow$ L, where L is a weight set and Ω maps a triangular face into a weight value. For $0 < i < j < n$, let $W_{i,j}$ be the weight of the minimum-weight triangulation. The triangulation algorithm is elaborated in details as follows:

Triangulation Algorithm

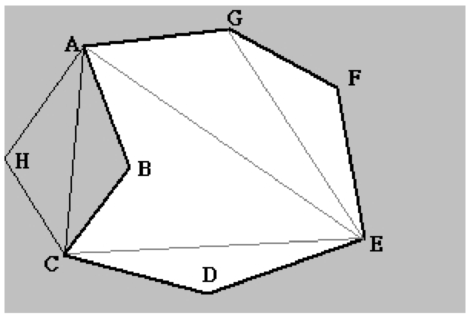
```
[1]  For i = 0, 1, ..., n-2, let $W_{i,j+1} = 0$.
     For i = 0, 1, ..., n-3, let $W_{i,i+2} = \Omega(v_i, v_{i+1}, v_{i+2})$.
     Set j = 2.
[2]  Set j = j + 1 For i = 0, 1, ..., n-j-1 and k = i + j,
     Let $W_{i,k} = \min_{i<m<k} [W_{i,m} + W_{m,k} + \Omega(v_i, v_m, v_k)]$,
     Save index m into $T_{i,k}$ when the minimum is achieved.
[3]  If $j < n - 1$ then go to step 2; otherwise the weight of the minimum-weight
     triangulation is $W_{0,n-1}$.
[4]  Trace back the path from the matrix T, and then we add the result triangles into the
     face list.
```

The triangulation algorithm acts like performing a convex hull finding problem even facing the crenellation situation. However, the triangulation algorithm will face the topological singularity problem if an edge is not a boundary edge but its two endpoints are boundary vertices. We will provide an example to show how topological singularities emerged when the above situation occurred.

As illustrated in Fig. 3.4, there exists a hole A, B, C, D, E, F, G in the surface shown in Fig. 3.4(a). In the triangulation algorithms, they didn't consider a special case. That is,

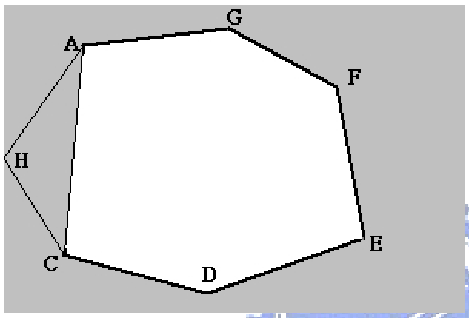


(a) Before filling process

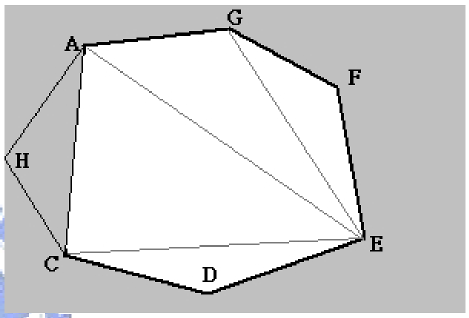


(b) After filling process

Figure 3.4: Potential problem of triangulation algorithm (1)



(a) Before filling process



(b) After filling process

Figure 3.5: Potential problem of triangulation algorithm (2)

when an edge is shared by two faces, it will ignore the case when another face would like to share the same edge. However, according to the case shown in Fig. 3.4(a), this situation does exist. Therefore, we proposed a revised version which deals with this problem at the cutting stage (which has been described in Chapter 2). If we apply the original triangulation algorithm, the newly formed patch may contain a set of face {f {A, B, C}, f{A, E, C}, f{C, E, D}, f{A, G, E}, f{G, F, E}} as shown in Fig. 3.4(b). The edge A, C is now adjacent to three faces and make it become singular. However, it is incorrect. Therefore, we revise the existing method by the following processing method.

If a face that is adjacent to the boundary vertices is adjacent to zero or one boundary edge, our triangulation algorithm will never produce a surface with topological singularities (for the convenience of explanation, we name the face adjacent to boundary vertices a boundary face). Our triangulation algorithm applied the dynamic programming technique to find all possible vertex pairs (v1, v2) and then form a triangular patch. When there exists zero or only one boundary edges on a boundary face, all the vertex pairs that can be used to make patches will always be the boundary edges. Fig. 3.5 shows that all the boundary faces are adjacent to one or zero boundary edge. And the triangulation algorithm will find a patch to fill the hole without generating topological singularities.