STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIX**, Special Issue 1, 2014
10th Joint Conference on Mathematics and Computer Science, Cluj-Napoca, May 21-25, 2014

FAST ALGORITHM TO SPLIT AND RECONSTRUCT TRIANGULAR MESHES

GÁBOR FÁBIÁN AND LAJOS GERGÓ

Abstract. In this paper we show a fast and efficient algorithm for cut and split a triangular mesh by a plane, and fully reconstruct the cutting surface. After the cut some of new triangular meshes will be created at the same format as the input mesh. Our approach is not to use complex data structures, just a vertex stream and an index stream keeping the algorithm simple, and ensuring the wide range of usability. We suggest a model for describe the boundary of a solid mesh to obtain advantages as a consequence of geometric topology. If we suppose the streams satisfy some reasonable restrictions, we find our algorithm has linear time complexity including the precomputation, splitting section, reconstruction and the decomposition.

1. Introduction

Triangular mesh operations play important rules in many applications of 3D graphics. Mesh cutting is often used in surgery simulation [10], physics-based simulation [3], or computer aided design [13]. Mesh slicing is a robust technique to take geometry information about the mesh [14], and to do topological modification [2]. Mesh cut and mesh slice algorithms generally do not contain reconstruction elements, because it is not needed to reuse the cut meshes in a new cut iteration. Mesh splitting algorithms are designed to do the reconstruction steps, but these algorithms usually work with approximation, and can be used only once for a mesh. Finding good-working mesh splitting algorithm is common topic on game developer forums. The goal of this paper to show a mesh splitting algorithm, that is fast enough to be used in real time applications, and topologically established in the sense that any newly created mesh has the same topological information as the original mesh before the cut.

Received by the editors: May 1, 2014.

2010 *Mathematics Subject Classification.* 68U05.

1998 *CR Categories and Descriptors.* I.3.5. [Computing Methodologies]: Computer Graphics – *Computational Geometry and Object Modeling*.

Key words and phrases. mesh splitting, mesh slicing, topological reconstruction.

2. Overview

There are some known algorithm to perform mesh slicing. The easiest way is the trivial slicing [4], if every triangle is tested for intersection. There are several similar methods working on STL files, more information can be found in [15]. The modern approach is using the triangle's z-coordinates to find the intersecting edges instead of triangles. Z-coordinates are used in sweep plane algorithms [9] and triangle grouping algorithms, see [12]. These algorithms are often use hashtables, complex data structures to minimize number of intersection tests. Mostly intersecting segments are computed first, after that the disjoint segments have to be merged to a contour (or edge loop), as can be seen in e.g. [7]. Our approach is different. Utilizing the topological properties of the mesh, we can get the edge loop in a single step. We will suppose that the mesh is a polygonal manifold (see [11] or [14]), and we use face-based representation [6]. The topological information obtained from an OBJ-like data structure, mesh is represented by a vertex stream, an index stream, and a face-adjacency stream. Our streams are arrays, we use low level operations to make the discussion and the implementation easier. After the cut, we obtain a set of new meshes, which have all corresponding streams completely recovered. This can be interpreted as the cut operator is closed respect to the mesh data structure. It means the new meshes have to be polygonal manifolds, therefore it is not enough to get the edge loop(s), we need to recover the split surface of the new meshes and decompose them into connected components. These steps are missing from a common mesh slicing algorithm. We attempted to design our algorithm enough fast for real time applications, thus the adjacency-streams are created with the new meshes together. We did not find this kind of approach in literature.

3. Definitions and Notations

Denote $[a..b]$ the discrete interval from a to b, i.e. $[a..b] := [a, b] \cap \mathbb{N}$, and let us define the vertex stream as $V : [1..m] \to \mathbb{R}^3$, and the index stream $F : [1..n] \to [1..m]^3$ for some $n, m \in \mathbb{N}$. Suppose that (V, F) altogether defines the boundary of a solid mesh in the sense that any (i, j, k) index triple in F defines a triangle plate $\triangle(V(i), V(j), V(k))$ on the boundary of the mesh. We would like to follow the convention, that vertex indices of a triangle in F are counter-clockwise (CCW) ordered, and the normal vector can be obtained by the right hand rule. Let us introduce the notation for a triangle defined by the face $F(i)$ as

$$\triangle F(i) := \triangle(V(F(i, 1)), V(F(i, 2)), V(F(i, 3)))$$

We assume the mesh is solid, so its boundary is a non-self-intersecting closed surface, consequently any edge belongs exactly two triangles [5]. On the other hand, we will suppose that a solid mesh does not contain cavities. It is not necessary, but helps to keep the discussion much more simple. If (V, F) satisfies all the conditions above, we will say (V, F) defines a solid mesh.

Since any cross-section of a closed surface is a closed curve on the cutting plane, we have

Remark 1. *If (V, F) defines a solid mesh, then any of its cross-section defined by an arbitrary plane is a set of simple polygons.*

Consider a plane given by a normal $\mathbf{n}$ and a vertex $\mathbf{p}$. Preserving the uniformity of discussion let us suppose, that there are no vertices in V lying on the cutting plane, i.e.

$$\forall j \in [1..m] \ : \quad s(j) := \langle V(j) - \mathbf{p},\ \mathbf{n} \rangle \neq 0$$

where $\langle ., . \rangle$ denotes the dot product in $\mathbb{R}^3$. We will say, a plane is allowed, if it satisfies the condition above. In the practice, if we have a not allowed plane, we will translate it for a small $\varepsilon > 0$ in the normal direction. For example ε can be defined as below:

$$\varepsilon \ := \min \left\{ \varepsilon_0, \quad \frac{1}{2} \min\{|s(j)| \ : \ s(j) \neq 0 \ j \in [1..m]\} \right\}$$

where ϵ_0 is a small fixed positive, the maximal admissible tolerance. The new allowed plane is given by the normal $\mathbf{n}$, and a new vertex $\mathbf{p}' := \mathbf{p} + \varepsilon\mathbf{n}$. The triangles in F can be sorted to three sets, triangles completely above or below the plane, and the intersecting triangles.

We obtain two important remarks by using allowed planes.

Remark 2. *If P is allowed cutting plane, then every intersecting triangle has exactly two edges, that have intersection with P.*

Since (V, F) defines a solid mesh, the following is true.

Remark 3. *If P is allowed plane, then every intersecting triangle has two another intersecting adjacent lying on the intersecting edges. The 3rd adjacent might be intersecting.*

4. Generating adjacencies

To take advantages of our conditions we need to know the edge-adjacent triangles for any triangle. We will define the adjacency stream, but first we introduce the triadic addition operator to perform operations on triangles easier.

$$\dot{+} : \{1, 2, 3\} \times \mathbb{N} \to \{1, 2, 3\} \ : \quad j \dot{+} k := (j + k - 1 \bmod 3) + 1$$

For example if $j = 2$ we have $j \dot{+} 0 = 2,\ j \dot{+} 1 = 3,\ j \dot{+} 2 = 1,\ j \dot{+} 3 = 2$, and so on. Because our triangles are CCW, $F(i, j \dot{+} 1)$ is the first vertex of the ith triangle after the jth vertex in CCW order.

Suppose that (V, F) defines a solid mesh, then we can assign to (V, F) the $A : [1..n]^3 \to [1..n]$ adjacency stream, where $A(i, j) := k$ if and only if $\triangle F(i)$ adjacent to $\triangle F(k)$ along the edge $\overline{F(i, j)F(i, j \dot{+} 1)}$. The assignment is unique as a consequence of definition of solid mesh.

Our first task is to generate the A adjacency stream. It can be done offline once and for all before the algorithm starts. We suggest to store the mesh as (V, F, A), completed by the adjacencies. Now we show a linear time method, to calculate A.

Construct an $m \times m$ matrix N containing zeros. If there is an edge between the vertices $V(i)$ and $V(j)$, then exactly two triangles join to this edge, e.g. $\triangle F(k)$ and $\triangle F(l)$. Then let $N_{ij} := k$ and $N_{ji} := l$ or inversely $N_{ij} := l$ and $N_{ji} := k$. Then $N_{ij} \neq 0$ if and only if there exists an edge $\overline{V(i)V(j)}$, moreover $N_{ij} \neq 0$ if and only if $N_{ji} \neq 0$. Now from N we can construct A easily. Consider the jth vertex of the ith face, and its edge $\overline{F(i, j)F(i, j \dot{+} 1)}$. There are exactly two triangles joining to this edge, $\triangle F(i)$ and one another. Consequently either $i' := N_{F(i,j),F(i,j\dot{+}1)}$ or $i'' := N_{F(i,j\dot{+}1),F(i,j)}$ is the adjacent triangle's index that we are looking for, and obviously the other element is i. Then let

$$A(i, j) := \begin{cases} i' & i' \neq i \\ i'' & i'' \neq i \end{cases}$$

It can be checked, this construction of A corresponds to the definition of adjacency stream.

5. Sorting vertices and faces

Let us suppose that a (V, F, A) solid mesh is given with its face adjacencies, and given an allowed plane $P := \{x \in \mathbb{R}^3 \mid \langle x - \mathbf{p},\ \mathbf{n}\rangle = 0\}$. As P is allowed $s(j) \neq 0 \quad (j \in [1..m])$, where $s(j) = \langle V(j) - \mathbf{p}, \mathbf{n}\rangle$. We need to construct (V_1, F_1, A_1) and (V_2, F_2, A_2) collections of vertex, index and adjacency streams, that are obtained below and above the cutting plane, respectively. First we sort the vertices into two disjoint sets. Put $V(j)$ to V_1 if it is below the plane, else put it to V_2. Besides we define the π_V permutation. π_V assigns the new indices in V_1 or V_2 to old indices in V.

Consequently

$$\begin{aligned} \pi_V(j) = k \quad \Leftrightarrow \quad & (s(j) < 0 \text{ and } V_1(k) = j) \text{ or} \\ & (s(j) > 0 \text{ and } V_2(k) = j) \end{aligned}$$

Next we need to sort the faces. Notice, that a face completely below or above the cutting plane if and only if its all three vertices below or above the plane. In all other cases the face is intersecting, this case will be discussed later. Let us introduce the notation $f_{i,j} := F(i,j)$. We sort the faces just like the vertices, and we construct the permutation function π_F like before, except that its value is 0 for an intersecting face. So π_F assigns the new face indices in F_1 or F_2 to old indices in F. So we have

$$\pi_F(i) = \begin{cases} 0 & s(f_{i,j}) = 0 \\ k & (s(f_{i,j}) < 0 \text{ and } F_1(k) = i) \text{ or} \\ & \quad (s(f_{i,j}) > 0 \text{ and } F_2(k) = i) \end{cases}$$

Notice, that π_F can be used to decide if a triangle is intersecting or not.

Remark 4. *$\pi_F(i) = 0$ if and only if $\triangle F(i) \cap P \neq \emptyset$.*

Finally we need to sort the A adjacency stream ensuring that the newly formed meshes can be cut again. Consider an arbitrary triangle from F_1 or F_2. If every three adjacent triangles completely below or above the cutting plane, our task is easily reindexing the faces by π_F. Remark 4 follows that if there is an intersecting triangle between the adjacencies, then A_1 or A_2 will contains a 0. We obtain the following result.

Remark 5. *$A_\alpha(i,k) = 0$ $(\alpha = 1,2)$ if and only if the ith triangle of F_α adjacent to an intersecting triangle along the edge $\overline{V_\alpha(k)V_\alpha(k \dot{+} 1)}$.*

6. Reconstruction of cutting surface

We need to construct new triangles from the intersecting triangles. As we have an allowed plane, any triangle has two intersecting edges, consequently after the cut we obtain one new triangle and one quadrilateral. The quadrilateral need to be divided into two triangles. Resulting three triangles are not intersecting, one triangle has the unique non-intersecting edge, two triangles has a cutting edge (edge lying on the cutting plane).

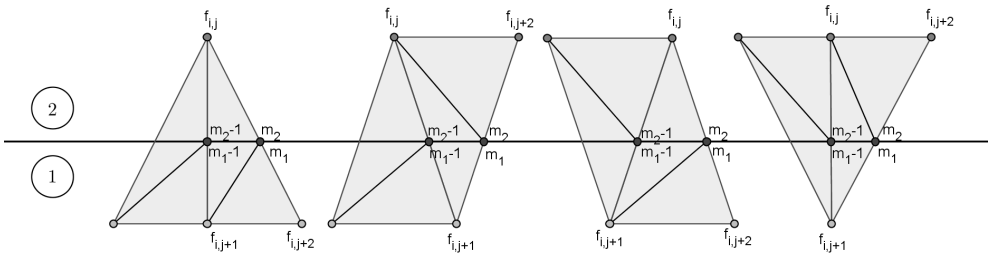
Due to Remark 1 we know, the cross-section is set of simple polygons. Consequently if we found an intersecting triangle $\triangle F(i_1)$ then there exist $\triangle F(i_2), \dots \triangle F(i_L)$ triangles such $\triangle F(i_j)$ is adjacent to $\triangle F(i_{j+1})$ for $j = 1, \dots, L-1$, and $\triangle F(i_L)$ adjacent to $\triangle F(i_1)$. Then we say $\{\triangle F(i_j)\}_{j=1}^{L}$ triangles define a face loop. Consequently, the cross-section $\{\triangle F(i_j) \cap P\}_{j=1}^{L}$ is an edge loop on the plane P, i.e. an L-vertex polygon. We give an iterative method to correctly finish the $V_\alpha, F_\alpha, A_\alpha$ streams ($\alpha \in \{1,2\}$) to obtain sets of solid meshes with adjacencies. In every iteration one new vertex, three new triangles will be created, and some of adjacencies will be set.

We will follow some conventions, to give a comprehensive discussion about the possible cases. Let us denote the points below and above the cutting plane with domain 1 and domain 2, respectively. Now we working only with intersecting triangles, therefore at least one vertex lies in domain 1, and at least one another in domain 2. If the third vertex is in domain 1 then we say the triangle faces upwards, else we say it faces downwards. We need to examine two adjacent triangles at the same time to set its adjacencies correctly, consequently there are four possible cases (Up-Up, Up-Down, Down-Up, Down-Down).

Let us suppose that we are currently processing $\triangle F(i)$, and the previous triangle is processed aside from setting its adjacencies with the current triangle, the non-intersecting edge and the cutting edges. Furthermore, we assume, that preceding triangles in two steps or more distant from current one now completely processed aside from the non-intersecting edge and the cutting edges, except the first one, where the face loop starts. We will follow the next three rules.

- rule 1: The intersecting edge between vertices $f_{i,j}$ and $f_{i,j+1}$ processed in the previous step, and $f_{i,j}$ always lying in the domain 2.
- rule 2: All of vertices $f_{i,j},\ f_{i,j+1},\ f_{i,j+2}$ are the first vertex of exactly one newly created triangle. The new triangle's vertices are CCW ordered.
- rule 3: The last added triangle contains the unique non-intersecting edge.

FIGURE 1. Rearrange vertices



6.1. **Rearrange vertices.** Figure 1 shows how can we label the vertices. Notice, rule 1 and rule 2 ensures that there are no way to label the vertices differently. If the current triangle faces upwards, then the new vertex lies on edge $\overline{V(f_{i,j})V(f_{i,j+2})}$. Obviously we need to add this new vertex to V_1 and V_2 as well. V_1 and V_2 contains m_1 and m_2 vertices up to know, in this step we

increase the counters, and add the new vertex.

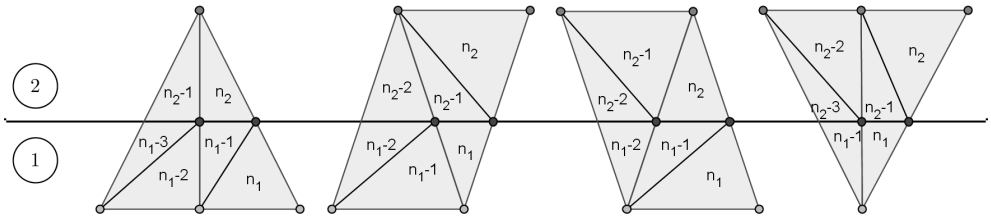
$$x := \overline{V(f_{i,j})V(f_{i,j+2})} \cap P$$
$$m_2 := m_2 + 1$$
$$V_2(m_2) := x$$
$$m_1 := m_1 + 1$$
$$V_1(m_1) := x$$

Notice, that the previous triangle's intersecting vertex already added to vertex streams. These are $V_1(m_1 - 1)$ and $V_2(m_2 - 1)$, it can be seen in Figure 1.

Next, if the current triangle faces downwards, the only difference is that the other edge is intersecting.

$$x := \overline{V(f_{i,j+1})V(f_{i,j+2})} \cap P$$
$$m_2 := m_2 + 1$$
$$V_2(m_2) := x$$
$$m_1 := m_1 + 1$$
$$V_1(m_1) := x$$

FIGURE 2. Rearrange faces



6.2. **Rearrange faces.** First let us suppose, that the current triangle faces upwards. Then F_2 gets one new triangle, F_1 gets two triangles. For example consider the top triangle. As shown in Figure 1, the triangle's top vertex is $f_{i,j}$ in the original F stream. Consequently in F_2 its index is $\pi_V(f_{i,j})$, see definition of π_V. The other two vertices of the triangle is the last added, and the previous one. These indices in V_2 are m_2 and $m_2 - 1$. So we need to add the triangle $(\pi_V(f_{i,j}), m_2 - 1, m_2)$, because it is CCW ordered, and starts with one of the original triangle's vertices corresponding to rule 2. The other two triangles' vertex indices can be read from Figure 1.

$$n_2 := n_2 + 1$$
$$F_2(n_2) \ := (\pi_V(f_{i,j}),\ m_2 - 1,\ m_2)$$

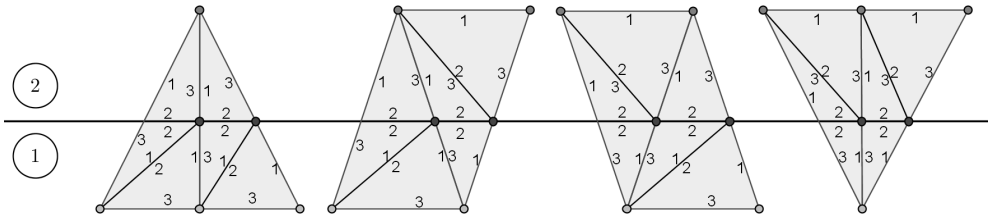
$$n_1 := n_1 + 2$$
$$F_1(n_1 - 1) := (\pi_V(f_{i,j+1}),\ m_1,\ m_1 - 1)$$
$$F_1(n_1) := (\pi_V(f_{i,j+2}),\ m_1,\ \pi_V(f_{i,j+1}))$$

Notice triangles are added to F_1 and F_2 in order as rule 3 requires, i.e. the last added triangle has the non-intersecting edge.

If the current triangle faces downwards, we have

$$n_2 := n_2 + 2$$
$$F_2(n_2 - 1) := (\pi_V(f_{i,j}),\ m_2 - 1,\ m_2)$$
$$F_2(n_2) := (\pi_V(f_{i,j+2}),\ \pi_V(f_{i,j}),\ m_2)$$
$$n_1 := n_1 + 1$$
$$F_1(n_1) := (\pi_V(f_{i,j+1}),\ m_1,\ m_1 - 1)$$

FIGURE 3. Rearrange adjacencies



6.3. **Rearrange adjacencies.** Our next task is adjusting the adjacencies for the newly created faces. Figure 2 shows the indices of the faces created in the previous and the current step. In Figure 3 can we see the order of the edges, rule 2 guarantees the uniqueness. The order of the vertices determines the order of edges, that is important to know creating adjacency stream correctly, see definition.

Suppose that the current and the preceding triangle faces upwards. Then the top triangle's 1st edge is adajcent to the previous top triangle's 3rd edge. Set the following adjacencies.

$$A_2(n_2, 1) := n_2 - 1$$
$$A_2(n_2 - 1, 3) := n_2$$

The 2nd edge of current top triangle is cutting edge, it will be discussed later. The adjacency on 3rd edge is not defined yet, easy to verify, it will be set in the next step. The bottom triangle's adjacencies can be specified using Figure 2 and Figure 3.

$$A_1(n_1 - 1, 1) := n_1$$

$$A_1(n_1, 2) := n_1 - 1$$
$$A_1(n_1 - 1, 3) := n_1 - 2$$
$$A_1(n_1 - 2, 1) := n_1 - 1$$

The 2nd edge of $(n_1 - 1)$th triangle is cutting edge. The 1st edge of n_1th triangle will be set in the next step. The 3rd edge of the n_1th triangle is the non-intersecting edge. It can be checked, the adjacencies discussed above will not change even the preceding triangle faces downwards, thanks to our rules. It is important to notice, every adjacencies that not defined yet will be set in the next iteration, except the cutting edges. Let us define the cutting edge adjacencies to 0.

$$A_2(n_2, 2),\ A_1(n_1 - 1, 2) := 0$$

In this case the non-intersecting edge is $\overline{V(i, j \dot{+} 1)V(i, j \dot{+} 2)}$, so the adjacent triangle's index is

$$A_1(n_1, 3) := \pi_F(A(i, j \dot{+} 1))$$

Notice, if the non-intersecting edge-adjacent is intersecting triangle, then $\pi_F(A(i, j \dot{+} 1)) = 0$. We would like to preserve adjacency information, so we set

$$\pi_F(i) := n_1$$

It also means, further the ith triangle is considered non-intersecting. Later, when we will process the $A(i, j \dot{+} 1)$th triangle, we will set the inverse of the adjacency. We only do this change only if $\pi_F(A(i, j \dot{+} 1)) \neq 0$.

$$\pi_F(A(i, j \dot{+} 1)) \neq 0 \ \Rightarrow\ A_1(\pi_F(A(i, j \dot{+} 1)), 3) := n_1$$

If the current triangle faces downwards regardless to the previous triangle's direction we have the following adjacencies.

$$A_1(n_1, 3) := n_1 - 1$$
$$A_1(n_1 - 1, 1) := n_1$$
$$A_2(n_2 - 1, 3) := n_2$$
$$A_2(n_2, 2) := n_2 - 1$$
$$A_2(n_2 - 1, 1)\ := n_2 - 2$$
$$A_1(n_1, 2),\ A_2(n_2 - 1, 2) := 0$$
$$A_2(n_2, 1) := \pi_F(A(i, j \dot{+} 2))$$
$$\pi_F(A(i, j \dot{+} 2)) \neq 0 \ \Rightarrow\ A_2(\pi_F(A(i, j \dot{+} 2)), 1) := n_2$$
$$\pi_F(i) := n_2$$

In terms of all possible cases we obtain the following remark.

Remark 6. *$A_\alpha(i, 2) = 0$ if and only if the ith triangle in F_α has cutting edge.*

6.4. **Processing a face loop.** Let us take attention only one face loop for present our consideration. Typically we have more face loops after the cut, this method can easily adapted for this case using the stream indices carefully. Let us suppose, that we found the first intersecting triangle of a face loop, $\triangle F(s)$. At this point we need to save the start index and the current size of vertex and index streams, so let us save $s, m_1^{(s)}, m_2^{(s)}, n_1^{(s)}, n_2^{(s)}$. First we find the single intersecting edge going from domain 2 to domain 1. It will define the first vertex that need to be added to V_1 and V_2. Now we need to find the adjacent triangle along the unique intersecting edge, and start the vertex, face and adjacency rearranging discussed in the preceding subsections. After processing the ith triangle $\pi_F(i) \neq 0$, therefore the next triangle is clearly defined, as one of the intersecting edges now adjacent to a non-intersecting triangle. Repeat rearranging until we find that the next adjacent triangle's index is s. Then we need to close the face loop. The last processed triangle is $\triangle F(s)$, because its preceding adjacencies are undefined. The only difference is, that we do not need to add a new vertex to vertex streams, because the start points $V_1(m_1^{(s)}+1), V_2(m_2^{(s)}+1)$ has been contained. For setting the adjacencies we can use $n_1^{(s)}+1,\ n_1^{(s)}+2,\ n_2^{(s)}+1,\ n_1^{(s)}+2$ indices depending on the triangle's orientation. After the last step every vertex and face will contained in V_α, F_α, and all adjacencies are properly defined in A_α, except the cutting edges and non-intersecting edges.

7. Completing adjacencies, capping holes, dividing components

Finishing the adjacency streams we need to adjust the missing adjacencies. Suppose that the length of a face loop started with $\triangle F(s)$ is L. Since Remark 1, its intersection with P is an L-vertex simple polygon, that can be defined by a sequence of its vertices.

$$V_\alpha(m_\alpha^{(s)}), \ldots, V_\alpha(m_\alpha^{(s)}+L)$$

If we rotate these points to the P plane, we obtain a common polygon triangulation problem. Rotation is can be done e.g. by a translation and change of basis. So let us define $\mathbf{e} := V(s+1) - V(s)$, and the matrix R

$$R := (\mathbf{n}, -\mathbf{n} \times \mathbf{e}, \mathbf{e})$$

where $\times$ denotes the cross product. Then $\mathbf{v}'$, the image on the plane of vertex $\mathbf{v}$ can be computed by

$$\mathbf{v}' = R^{-1}(\mathbf{v} - \mathbf{e})$$

Simple polygons can be triangulated easily, many algorithms are known to solve this problem, see e.g. [1]. After the triangulation is done, we get an index stream like $T : [1..L-2] \to [1..L]^3$, containing vertex triplets that

define the triangles. First we sort T, such that $T(j)$ triangle contains the jth edge of the polygon. We need to set the normals correctly, therefore we calculate the normal of the triangle defined by $T(j)$. If the triangle's normal has the same direction as the plane's normal, then let $T_1(j) := T(j)$ and $T_2(j) := (T(j,2), T(j,1), T(j,3))$. If the normals have opposite directions then let $T_1(j) = (T(j,2), T(j,1), T(j,3))$ and $T_2(j) = T(j)$. The acyclic permutation of vertex indices reverses the normal vector of the triangle, so after this transformation the normal of any triangle in T_1 has the same direction as the plane's normal, and the opposite in T_2. Thereafter we only need to shift the triangulation indices, and add it to the index stream.

$$F_\alpha(n_\alpha + j) := T_\alpha(j) + m_\alpha^{(s)} \quad j = [1..L-2]$$

Defining the corresponding adjacencies is quite simple. Notice, that any triangle in T_α is adjacent to two other triangles in T_α, and one from F_α. Therefore if we generate the adjacencies of T_α (see Section 4), as result we get the A_{T_α} adjacency stream, this indices need to be shifted just like above. Furthermore $A_{T_\alpha}(j,k) = 0$ if and only if the kth edge of jth triangle has an adjacent from F_α, and it is can not be other than the jth triangle after the $n_\alpha^{(s)}$th triangle of F_α, which has cutting edge, since T_α is sorted by cutting edges. We only need to set these triangle pair adjacencies correctly and we are done with reconstruction.

At this point for a given (V, F, A) solid mesh and a P plane we constructed two other sets of solid meshes $(V_1, F_1, A_1), (V_2, F_2, A_2)$ below and above the plane, respectively. Finally we need to find the connected components. It can be done easily knowing the adjacency streams. Consider a triangle and mark it, thereafter go to an unmarked edge adjacent triangle, and mark it. This must be repeated, while exists unmarked triangle. If there are no more unmarked triangles, we found a connected component of a solid mesh set. If every triangle has been marked in the set, we have found all connected components.

8. Summary and Further Works

Our algorithm input is a solid mesh with adjacencies (V, F, A), and its output a collection of solid meshes (V_i, F_i, A_i) at the same format. In the previous sections we showed, that all new (V_i, F_i, A_i) triplet defines a solid mesh indeed with all of its adjacencies. Our algorithm is designed for real time applications, therefore let us deal with the time complexity. Let us suppose, that $|V| = m$, $|F| = n$, and the any cross-section of the (V, F) solid mesh contains p polygons at most. Le us assume that the ith polygon has L_i number of edges. We will examine the time complexity respect to the

face count, because $n \neq m$ for boundary of an arbitrary solid mesh. Sorting vertices, faces, adjacencies takes linear time. In rearrange section we need to find a face where a face loop starts, it can be done in linear time. Processing a face loop depends on its edge count, consequently the ith face loop can be processed in L_i time. Capping this face loop with a common triangulation algorithm takes $L_i \log L_i$ steps, sorting by cutting edges takes the same amount of time. If all p face loops are processed, we need to look for connected components. Decomposition is obviously linear, since we marked the loop starter faces. The following table summarize the time complexity of different parts of the algorithm.

Generating adjacencies	n
Sorting vertices	m
Sorting faces	n
Sorting adjacencies	n
Rearrange (V,F,A)	$pn + \sum_{i=1}^{p} L_i$
Capping holes	$2\sum_{i=1}^{p} L_i \log L_i$
Decomposition	pn

The characteristic number of loops in a cross-section p depends only on the topological properties of the mesh, not on n, m. Furthermore if we consider a common model, not some special counterexample, then we find that any polygon of any cross section has relevantly lesser edges, than n. Hence $L_i << n$ implies $L_i \log L_i << n$ for all $i \in [1..p]$, so we can suppose that $\sum_{i=1}^{p} L_i \log L_i < n$. On the other hand, it is known, a simple polygon theoretically can be triangulated in linear time [5], but the common triangular meshes used in 3D graphics satisfy our last condition. Consequently rearrange and capping holes as well can be done less then $2pn$ steps, i.e. $O(n)$ time. In this sense our algorithm's time complexity is linear respect to n, so it is an asymptotically optimal solution of the problem [12].

The algorithm implemented and tested in MATLAB, we plan high-level implementation in C# getting information about limitations of real time application respect to n and p. We are motivated to modify our algorithm, empower it to cut meshes with half-planes or triangles, and test it in physics-based or medical simulations. In the future we would like to handle not only vertices and indices, but texture coordinates as well, serving the requirements of real time graphics software developers.

References

[1] *M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry - Algorithms and Applications*, Springer, 2008.
[2] M. Botsch, L. P. Kobbelt *A Robust Procedure to Eliminate Degenerate Faces from Triangle Meshes* VMV 2001, Stuttgart, Germany, 2001.
[3] C. D. Bruyns, S. Senger, *Interactive cutting of 3D surface meshes*, Computers & Graphics, 2001.
[4] K. Chalasani, B. Grogan, *An algorithm to slice 3D shapes for reconstruction in prototyping systems*, ASME Computers in Engineering Conference, 1991.
[5] B. Chazelle, *Triangulate a Simple Polygon in Linear Time* Discrete & Computational Geometry, 1991.
[6] S. Ghali. *Introduction to Geometric Computing*, Springer, 2008.
[7] X. Huang, Y. Yao, Q. Hu, *Research on the Rapid Slicing Algorithm for NC Milling Based on STL Model* AsiaSim 2012 Communications in Computer and Information Science, 2012.
[8] J. M. Lee, *Introduction to Topological Manifolds*, Springer, 2011.
[9] S. McMains, C. Sequin, *A coherent sweep plane slicer for layered manufacturing*, Proceedings of the 5th ACM symposium on Solid modeling and applications - SMA, 1999.
[10] C. Mendoza, C. Laugier, *Simulating Cutting in Surgery Applications using Haptics and Finite Element Models*, Proceedings of the IEEE Virtual Reality, 2003.
[11] R. Mukundan, *Advanced Methods in Computer Graphics - With examples in OpenGL*, Springer, 2012.
[12] G. Rodrigo, V. Neri, M. Rodrigo, S. Murilo *Slicing Triangle Meshes: An Asymptotically Optimal Algorithm*, Proceedings of the 14th International Conference on Computational Science and Applications, 2014.
[13] E. Sifakis, K. G. Der, R. Fedkiw, *Arbitrary Cutting of Deformable Tetrahedralized Objects*, Eurographics/ ACM SIGGRAPH Symposium on Computer Animation, 2007.
[14] M. Szilvási-Nagy, I. Szabó, *A Slicing Algorithm for Triangular Meshes*, 6th International Conference on Applied Informatics, Eger, Hungary, 2004.
[15] K. Tata, G. Fadel, A. Bagchi, N. Aziz *Efficient Slicing for Layered Manufacturing* Rapid Prototyping Journal vol. 4, no. 4, 1998.

Eötvös Loránd University, Faculty of Informatics, Department of Numerical Analysis, 1117 Budapest, Pázmány Péter sétány 1/C
E-mail address: robagnaibaf@gmail.com, gergo@inf.elte.hu