A State Machine for Real-Time Cutting of Tetrahedral Meshes

Daniel Bielser, Pascal Glardon, Matthias Teschner, and Markus Gross

Computer Graphics Laboratory
ETH Zurich, Switzerland
e-mail: bielser@inf.ethz.ch

Abstract

We introduce an algorithm that consistently and accurately processes arbitrary intersections in tetrahedral meshes in real-time. The intersection surfaces are modeled up to the current cut tool position at every point in time. Tetrahedra are subdivided by using a progressive method, which inserts the required sub-structures step by step. A state machine tracks the topology of each tetrahedron and controls the progressive subdivision. In order to keep the state machine as small and clear as possible, each topological pattern of a tetrahedral intersection appears only once. These topological patterns are mapped onto the actual case of a tetrahedral intersection by some given transformation operations. The state transitions, which contain the specific subdivision operations, are described in a predefined lookup table, which is written in a simple script language. The handling of reverse movements and possible trembling of the users hand, as well as a recursive continuation of the state machine concept, complement the proposed algorithm. In three examples, covering free form modeling, volume visualization, and surgery simulation, we indicate the large field of applications in which our algorithm can be utilized.

Keywords: Tetrahedral Meshes, Real-Time Cutting, Virtual Sculpting, Volume Visualization, Surgery Simulation

1. Introduction

In modern interactive simulation and modeling environments the ability to cut 3-dimensional geometry in real time is of fundamental importance. This creates the need for efficient cutting algorithms that process the underlying representation. Such methods can be utilized in a wide spectrum of applications including surgical interventions, free form modeling, or scientific visualization. In surgery simulation, for instance, interactive cutting algorithms enable the dynamic simulation of scalpel intersections that open immediately behind the scalpel. In the case of free-form modeling or sculpting, dynamic cutting supports a precise positioning and guidance of a cutting tool. In scientific visualization, real-time cutting algorithms create new opportunities for the interactive analysis of volume data sets. Seismic data sets, as an example, can be cut arbitrarily along interesting strata.

The cut of a 3-dimensional solid, however, changes the topology of the underlying data structure and thus poses a great technical challenge. The complexity of a cut algorithm largely depends on the underlying discretization. Very often, 3-dimensional material is represented by an unstructured tetrahedral mesh. As opposed to regular voxel cells, tetrahedral meshes are much more flexible allowing to represent complex data with less primitives. In addition, such meshes frequently serve as a basis for physically-based modeling methods, since they are much more efficient when calculating realistic deformations. The dynamic modeling of intersections and topological changes in such meshes, however, is non-trivial.

In this paper, we present a novel algorithm which accurately represents and tracks arbitrary cuts of tetrahedral meshes in real-time. The algorithm tracks topological changes and inserts new intersection faces dynamically and on-the-fly. Central to our approach is a state machine model to control the topological patterns, where each change induces a state transition. In order to keep the number of possible states small, we exploit the symmetry of cut patterns. Our method is robust and can cope with reverse movements and tremor of the user's hand.

2. Previous Work

Due to the complexity of volume based algorithms, cutting algorithms were first applied to surface meshes. There already exist some concepts for dynamically updating surface meshes that handle topological changes. [1] uses a strategy to determine and duplicate the vertices of the polyhedra that are close to the collision points along the line of cut. Recently, [5] introduced a face subdivision scheme that enables a more accurate representation of surface cuts. Additionally, this work supports the simultaneous cutting of multiple surface layers. Obviously, surface-based methods do not allow cutting of volumetric models.

An approach to cutting voxel-based representations can be found in the Chain Mail algorithm [10]. During cutting, intersected connections between neighboring voxels are split up. Unfortunately, bumpy intersection faces are unavoidable when using voxel representations.

In regards to volume meshes, all previous research has designed cutting algorithms which work on tetrahedral mesh decompositions. Considering the number of possible intersections, tetrahedra are topologically simpler than all other volumetric primitives. A simple realization of cuts in tetrahedral meshes is the removal of entire tetrahedra, examplified in [7]. It is real-time, but generates very uneven surfaces and a large gap between the two intersection faces. The procedure introduced in [13] does not create this gap. It spans the cut surface along existing mesh nodes. However, the generated surfaces exhibit uneven surfaces which are similar to [7] where entire tetrahedra are removed. Subsequent work [14] and [12] allows existing nodes to be shifted into the actual intersection surface and thus improves the previous approach. Nonetheless, the given degrees of freedom do not allow for representations of general intersection surfaces.

To our knowledge, the geometrical subdivision algorithm presented in [4] was the first algorithm that allowed real-time cutting of volume meshes. The infinite number of possible cut topologies of a tetrahedron is reduced by discretizing permitted intersections on a tetrahedron. The proposed discretization processes only one intersection per tetrahedral edge and per tetrahedral face. The evolving tetrahedral subdivisions are established by using a lookup table approach. The algorithm is effective for arbitrary, irregular tetrahedral meshes and captivates with a high level of accuracy and topological freedom. However, the universal subdivision scheme proposed in this work may lead to a rapidly increasing number of tetrahedra for large cuts and may produce cracks in a physical representation of the model.

An improved algorithm [2] solves these problems by applying individual subdivision patterns for each cut topology. The consistency of the mesh subdivision is achieved by restricting the allowed subdivisions of tetrahedral faces to the three types displayed in Fig. 1. An arbitrary combination of these face types results in the six subdivision patterns presented in Fig. 2. The algorithm demonstrated in [2] not only remeshes intersections, but also consistently inserts the entire substructure of the mesh, and provides correct junctions of neighboring elements. As a result, the simulation enables topological changes such as intersecting incisions or the complete severance of a model.

Only recently, [6] integrated the aforementioned cutting algorithm in their surgical simulator. Additionally, the approach was extended to multi-resolution grids by [8] and [9].

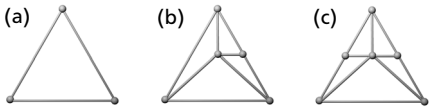

Figure 1: *Allowed subdivisions of tetrahedral faces*

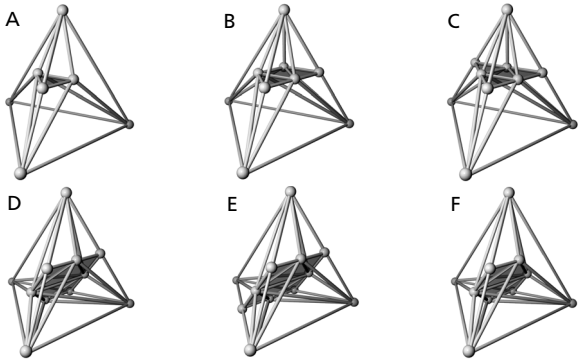

Figure 2: *Individual subdivision patterns for each subdivision topology of a tetrahedron*

The cutting algorithms described thus far process the individual tetrahedral elements only after they have been completely passed and all intersection positions have been determined. As a result, the representation of the intersection surface is not continuous, but contains gaps along the cut tool. In modeling and visualization applications, the incomplete intersection surface is unacceptable and complicates the precise guidance of the cutting tool. In regards to surgery simulation, the intersections are opened too late. At this point, a progressive construction of the tetrahedral subdivisions is desirable. [11] enhanced the original algorithm described in [4] by such a progressive representation of the tetrahedral intersections in a simple way. Each intermediate topology of a tetrahedron in the process of being subdivided is represented by the corresponding subdivision topology. For example, the sequence of patterns A, B, C in Fig. 2 is applied to model the subdivision depicted in pattern C. After each edge or face intersection, the entire data structure of the subdivision is removed, only to be replaced by the next subdivision pattern. Due to the large number of removal operations, this procedure is computationally expensive. Furthermore, the algorithm presented in [11] does not elaborate on successive incisions into a model.

3. Concept

In this paper, we suggest a progressive continuation of the tetrahedral subdivision algorithm applied in [2]. In contrast to [11] our subdivision of a tetrahedron is built incrementally from an undivided tetrahedron up to a complex subdivision with minimal removal operations. A state machine thereby tracks the topological pattern and controls necessary updates of each tetrahedron. This results in a fast algorithm for the dynamic simulation of very accurate volumetric trajectories in real-time. At every point in time, it presents the intersection surface up to the current position of the cut tool and allows intersecting trajectories.

In order to handle all possible cut trajectories, the progressive algorithm is continued in a recursive way. It subdivides a tetrahedron and continues the subdivision on the next lower subdivision level whenever the subdivision rules are not general enough to handle a particular tetrahedral

incision. Thereby, a correct continuation of intercepted intersection surfaces on the sub tetrahedra level has to be guaranteed.

The pseudo code below sketches our intersection algorithm. Before any geometric modifications can be processed, the intersections between a cut tool and the tetrahedral mesh have to be registered by a collision detection algorithm. The algorithm described in [2] and [3] consistently determines all edge and face intersections, even for dynamically deforming tetrahedral meshes. As long as no new edges or faces are intersected for a tetrahedron, the existing face intersection points of the tetrahedron are adjusted to the current intersection positions. When a new edge or face intersection occurs, the new subdivision state is checked for validity (only one intersection per edge and face). For valid states, the subdivision of the tetrahedron is performed by calling the procedure `SubdivideTetrahedron()`. This procedure first determines the transformation between the new topological pattern of the tetrahedron and its actual geometric representation, and then applies the state transition as predefined in the state machine. In the rare cases of invalid subdivision states, the tetrahedron gets completely subdivided according to its last valid subdivision state. Then, a repeated collision detection step marks all intersections between the cut tool and the sub tetrahedra.

Each sub tetrahedron affected by the intersection also gets subdivided after the transition from its parent tetrahedron is completed.

```
mark all tetrahedra currently affected by the cut;
for (all these tetrahedra)
    if (tetrahedron has new intersections)
        determine new subdivision state;
        if (new subdivision state is valid)
            SubdivideTetrahedron();
        else
            determine last valid state;
            subdivide tetrahedron completely;
            mark all sub tetrahedra affected by the cut;
            for (all these sub tetrahedra)
                handle transition from parent to
                                    subtetahedron;
            SubdivideTetrahedron();
    else
        adjust face intersection positions;

procedure SubdivideTetrahedron()
    determine transformation of topological pattern;
    apply state transition;
```

The following sections describe the algorithm in detail. Section 4 explains the functionality of the state machine, which is the core of our algorithm. Section 5 describes some extensions that handle reverse movements of a cut tool, and Section 6 introduces the recursive continuation of the state machine concept. Finally, Section 7 presents three applications of the introduced intersection algorithm.

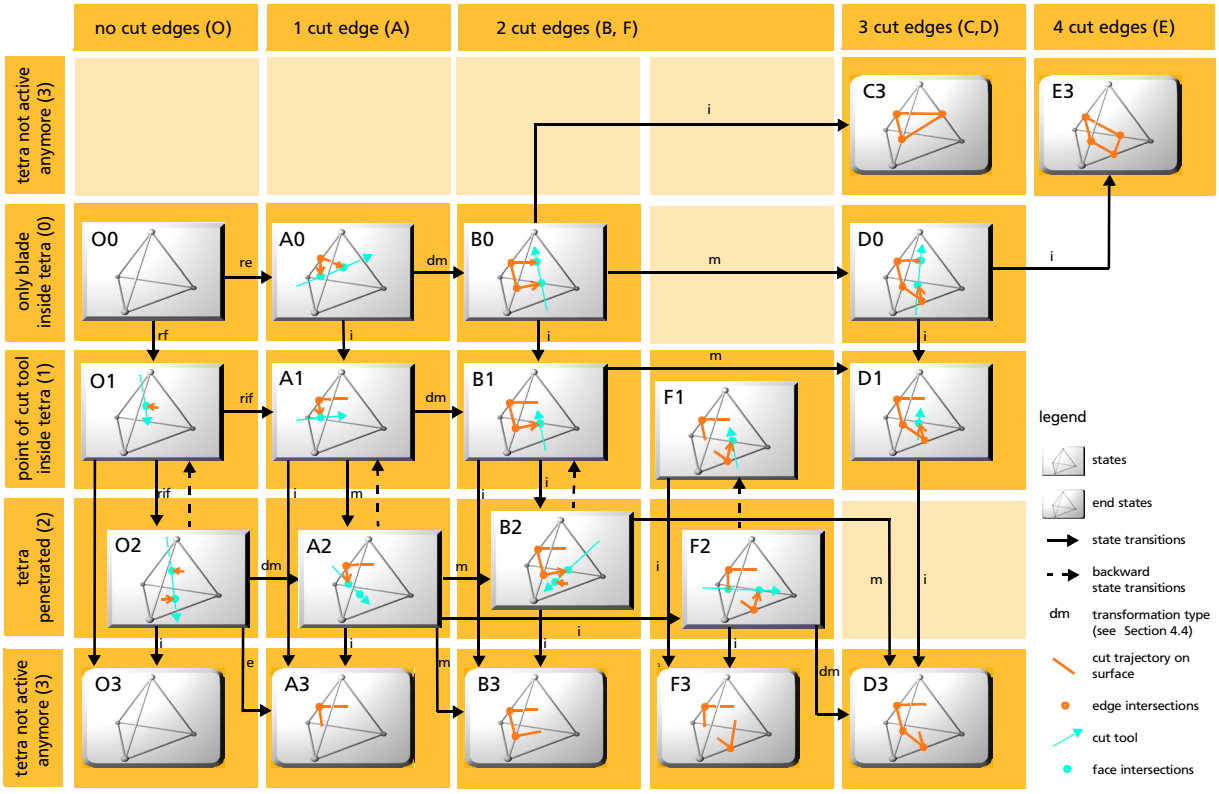

Figure 3: *State machine diagram*

4. State Machine Based Tetrahedral Subdivision

The key of our approach is a state machine which tracks the modification of a tetrahedron from an undivided state to a particular subdivision. This state machine clearly and efficiently handles the complex problem of arbitrary progressive subdivisions. In the state machine of Fig. 3, each state depicts a combination of edge and face intersections, defining a particular topology of an intersected tetrahedron. The states are labeled uniquely with a letter fixing the topology of the edge intersections and a number representing the topology of the face intersections. In the depicted state machine the states are sorted by the number of edge intersections from the left to the right and by the number and type of face intersections from top to bottom. Each tetrahedron stores its current subdivision state. The state O0 stands for the initial state of a tetrahedron. Whenever an additional edge or face is cut, the tetrahedron changes its state. The corresponding state transition describes the modification of the current subdivision. After some state transitions each tetrahedron arrives at one of the end states labeled with "3" where the cut tool has left the tetrahedron. The subdivisions of the states A3 to E3 correspond to the subdivision patterns already depicted in Fig. 2.

The consideration of all combinations of edge and face intersections would result in a state machine of more than 3000 states. Thus, we implement a state machine exclusively for the topological pattern of the subdivision. This approach results in a much smaller state machine, but requires the correct handling of all transformations from the general pattern to the particular subdivision. In Fig. 3, the transformation type that has to be considered for each state transition is denoted. The indicated transformations are explained in more detail in Section 4.4.

4.1. Basic Cut Functionalities

In a first step, the transformations are ignored and the operational sequence is explained with the help of the generic subdivision depicted in Fig. 3. The following example exemplifies the functionality of the progressive subdivision.

example 1: *state transition without transformations*

As mentioned, the final subdivision of a tetrahedron is constructed incrementally, where in each intermediate state the tetrahedron has a consistent mesh representation. Remove operations in a state transition are minimized for the sake of efficiency. In the example shown in Fig. 4 an intersection starts with an untouched tetrahedron (a), whose state is O0*. When the first collision between the cut tool and one of the tetrahedra's faces occurs in (b), the state of the tetrahedron is changed to* O1 *and the first new elements are inserted in the tetrahedron. At this position one node has to be inserted to store the entry point of the cut tool and another new node (named* ENP *as an abbreviation for entry point) will move with the current intersection position of the cut tool inside the face. Both nodes are strut to the surrounding face nodes. As long as no further edge collision is registered the* ENP *moves along the face (c).*

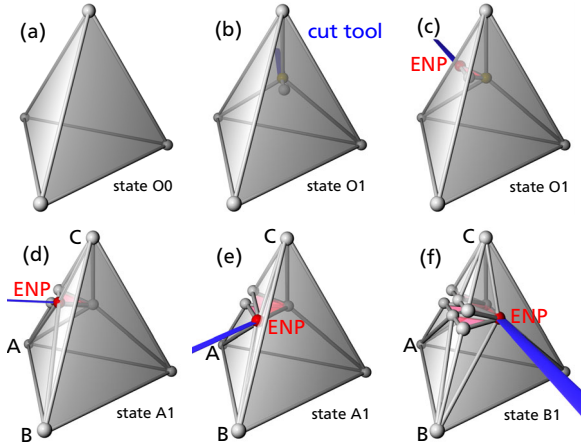

Figure 4: *Generic example for state transitions (without transformations)*

When the cut tool crosses one of the edges of the tetrahedral face, as depicted in (d), the corresponding edge has to be cut and the tetrahedral face is split in order to model the cut. The tetrahedron's state changes to the next state A1*. The* ENP *moves over to the next face and some connecting edges, faces, and tetrahedra have to be inserted. Then the cut tool continues moving inside the new face (e).*

In (f) a second edge has been cut. Now, the previous tetrahedral face has to be split completely. The processing of the tetrahedron continues in this way until the cut tool leaves the tetrahedron entirely.

The described procedure presents one major complication. For a partially intersected face it is not known in advance, if the cut may be continued and which edge will be affected. Since a consistent face subdivision has to be guaranteed at all times, the face has to be triangulated before its exact end-state is known. In order to illustrate this, a partially intersected face (A,B,C) is depicted in Fig. 5 (b). The current cut tool intersection is marked by the node n1 in the center of the face. The cut tool has entered the face over the edge (A,C). The cut tool has two means for leaving the face, it can either leave it without cutting any further edge, over the edge (B,C) as depicted in Fig. 5 (c), or it can leave the face over the edge (A,B) (Fig. 5 (d)). If in an implementation the two faces (B,C,n1) and (A,B,n1) were created by using the straight forward method of storing their vertices as references to the tetrahedral nodes, then complicated retrieval functions would be required in order to remove one of these faces shortly after. It would be computationally expensive to decide which of the two faces has to be removed when one of the tetrahedron's faces is changing from the state depicted in Fig. 5 (b) to the state in Fig. 5 (c) or (d). Additionally, it would be difficult to decide which of the already inserted faces, edges and tetrahedra have to be reassigned to the new face node n2 in Fig 5 (c) and (d).

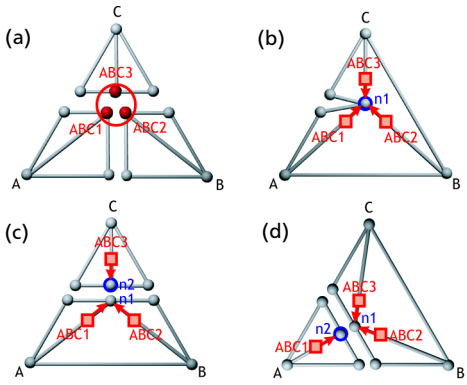

Figure 5: *Virtual Nodes for a progressive face subdivision*

This problem can be solved by introducing an indirection at data-structure level. For this purpose one can think of a tetrahedral face to be virtually presplit (Fig. 5 (a)). In such a virtually presplit face the three potentially new mid nodes ABC1, ABC2, and ABC3 (virtual nodes) are included in the data structure. When entering the face there is still only one real face mid node n1 created. The three virtual nodes point to n1 as depicted in Fig. 5 (b). All elements that are constructed at this stage reference the virtual nodes and do not reference the real node n1. At this time, if a mid node is split, a second real node n2 is inserted and referenced by the corresponding virtual node. As a result, all references of objects attached to the virtual node are reassigned automatically.

4.2. Transformation Operations

Since the state machine depicted in Fig. 3 only describes the topological pattern of the tetrahedral intersections for one given example, one has to incrementally determine the

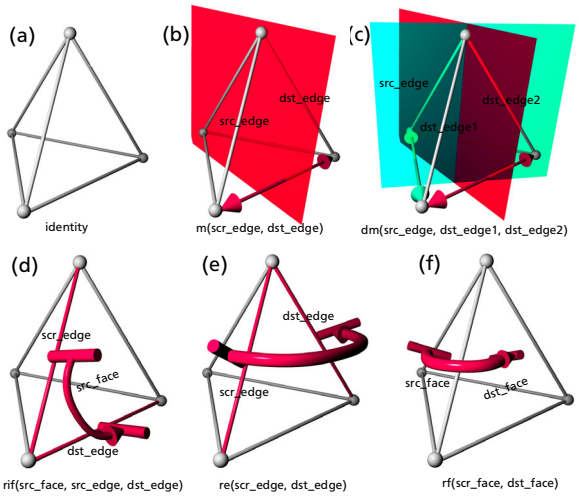

Figure 6: *Transformations:* (a) *identity,* (b) *mirror,* (c) *double mirror,* (d) *rotate inside face,* (e) *rotate over edges, and* (f) *rotate over faces*

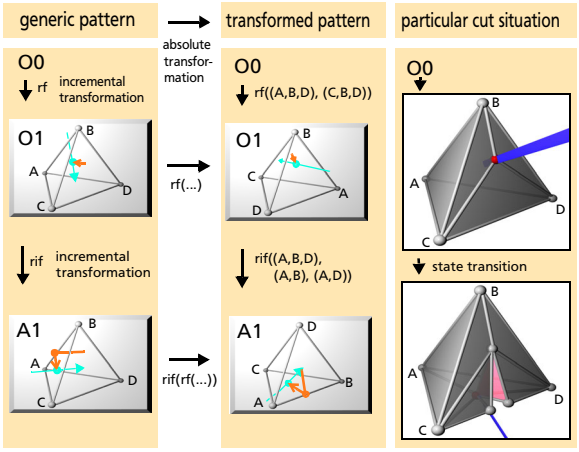

Figure 7: *Simple example for transformations*

transformation between this generic pattern and a particular case of intersection. The transformations, that have to be performed before applying the state transitions from the generic pattern to a particular intersection case, are restricted to specific types of rotations and mirror operations. Fig. 6 lists all types of transformations as they occur in the state machine of Fig. 3. All transformation types are described by the transformation from a *source object* to its *destination object*. The *source object* is the edge or face that is cut next according to the topological pattern of the state machine, whereas the *destination object* is the most recently intersected edge or face of the current tetrahedral representation. Thus, the transformation realizes the mapping of the object, for which the intersection is described in the generic state machine, to the object that has been actually intersected.

If multiple state transitions are performed for a tetrahedron, a sequence of transformations is applied to the tetrahedron. The current transformation state of a tetrahedron is built incrementally and has to be stored. For this reason, correspondences between the original and transformed vertices are stored. So, in our implementation, transformation operations are mapped to reassignments of vertices.

example 2: *state transition including transformations*

In the state machine shown in Fig. 3, the transformations to be considered are noted along each state-transition by use of the abbreviations defined in Fig. 6. If one looks again at the state transitions of example 1, the transformations are rf *for* O0→O1, rif *for* O1→A1 *and* m *for* A1→B1. *These are relative transformations between two states. Fig. 7 gives a closer look at the incremental construction of the pattern's absolute transformation. The left column shows the tetrahedron's topological state, as it appears in the generic state machine pattern of Fig. 3. In the right column, a particular state of a tetrahedron in the process of being cut is drawn. The figures located in the middle column depict the pattern on the left after they have been transformed according to the needs of the particular intersection case which is shown on the right.*

The pattern on the top left describes the first face intersection to appear in the face (A,B,D). *Note that in reality any of the four tetrahedral faces could be intersected first. Thus, the source face* (A,B,D) *has to be mapped onto the face where the first intersection occurs. In the example shown on the right, this is the face* (C,B,D). *Therefore the nodes of the tetrahedron have to be transformed accordingly. This is done by a rotation over tetrahedral faces, namely* rf((A,B,D), (C,B,D)). *Similarly, for the state transition* O1→A1, *the transformed pattern* O1 *only describes the cut of edge* (A,B), *although the intersected face could be left over each of the three edges of the face. So the already rotated pattern has to be transformed again. The edge* (A,B) *has to be mapped onto the edge* (A,D) *which was actually cut in the particular cut situation (image in the bottom right corner) by applying a rotation inside the face* (A,B,D), *namely* rif((A,B,D), (A,B), (A,D)). *The accumulated incremental transformation of a tetrahedron during its presence in the state machine, named absolute transformation, is stored in the tetrahedron together with its current state. The absolute transformation of our example after the edge cut is* rif(rf(...)) *with parameters for* rf(...) *and* rif(...) *as given above.*

4.3. Multiple State Transitions

Thus far, it has been assumed that a tetrahedron's state always changes to the immediate successor state in the state machine, which is caused by one atomic cut event. In practice, the movement of the cut tool from one registered position to the next can simultaneously intersect several edges or faces of an individual tetrahedron. If the current state of a tetrahedron is not reachable from the previous state within a single state transition, the modification of the tetrahedron is split up into a sequence of state transitions. The path in the state machine from the previous to the current state is then determined by some fix traversal rules. If an untouched tetrahedron is completely split during one time step, direct state transitions exist in which all operations can be performed more efficiently. If multiple state transitions affect more than one edge, the correct sequence of edge selections has to be found for the individual state transitions. In the state machine shown in Fig. 3, a direct neighbor of the already processed edges is always chosen.

4.4. Lookup Table Based Topology Processing

The actual topological modifications of the tetrahedra are processed by using a lookup table that contains an entry for every state transition. In order to deal with the complex construction of the tetrahedral subdivisions, a simple script language supports various instructions for inserting tetrahedra, different types of faces, edges, as well as instructions for placing and splitting nodes. Even conditional branches are required for some subdivisions. The script language is built upon a clear nomenclature referring to a presplit reference tetrahedron including all potential edge and face splits. The clearly aligned and comment script file is parsed and translated into a fast, dynamic lookup table data structure at program start.

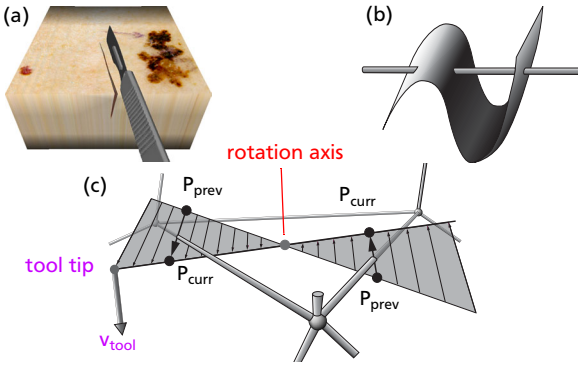

Figure 8: *Reverse cuts*

5. Reverse Movement

Up to this point, the movement of the cut tool has been restricted to forward movement, but in various situation the cut tool will cut edges or faces in reverse direction. A user may tear the cut tool back, after he has already performed an incision into the model. This case is depicted in Fig. 8 (a). Furthermore, the user's hand can tremble which causes several direction changes in the trajectory of the cut tool. This behavior is crucial when the cut tool is located very near to an edge of the tetrahedral mesh and will, therefore, cut the edge several times, as visualized in Fig. 8 (b). The described situations are very difficult to handle since the backwards movement is not a state of the entire cut tool, but rather a state of a section of the cut tool. Fig. 8 (c) depicts the situation of a cut tool that is turning around its cross axis. The cut tool section from the tip to the rotation axis is moving forward, whereas the other section of the cut tool is moving backward.

In order to overcome this problem the direction of the cut has to be determined for every single edge. To this end the position of the edge intersection is projected onto the previous and the current cut tool line. The projections are named P_{prev} and P_{curr} and depicted in Fig. 8 (c). The vector pointing from P_{prev} to P_{curr} is then compared to the direction of the cut tool's movement v_{tool}. As long as the scalar-product between the cut tool's movement direction and the direction of the edge cut is larger than zero, the edge has been cut by forward motion. In the other case, it has been cut backwards.

$$(P_{curr} - P_{prev}) \cdot v_{tool} > 0 \rightarrow \text{forward cut} \qquad (1)$$

Any backward movements should not alternate the volume model because only the front side of the cut tool's blade has a sharp edge. Consequently, edges and faces of the volume mesh are cut only if the cut tool touches them by forward movement.

5.1. Trembling

Trembling of the users hand can cause abrupt directional changes leading to many edge subdivisions. In order to suppress unnecessary edge subdivisions, intersections resulting from a backward movement get registered in the respective

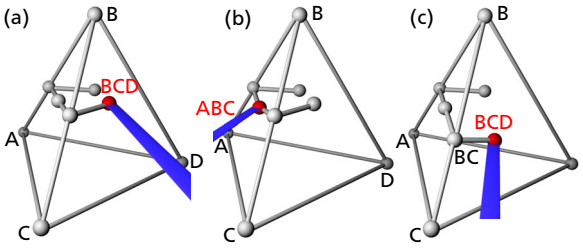

Figure 9: *Trembling over edges:* (a) *first edge cut,* (b) *backwards movement, second edge cut,* (c) *forward again, third edge cut*

edges. A further cut in forward direction is then allowed for these edges, still without any subdivision.

example 3: *trembling*

Fig. 9 shows an example of a multiple cut over an edge. Figure (a) depicts the situation after a first cut of the edge (B,C). *The cut tool currently intersects face* (B,C,D) *and has the face's mid node* BCD *attached. The cut tool moves backwards in figure (b) over the edge* (B,C) *already cut once, and the backwards intersection gets marked in edge* (B,C). *The cut tool now intersects face* (A, B, C) *and the mid node* ABC *is attached again to the cut tool. Shortly after (c) the cut tool is moving forward again and cuts the edge* (B,C) *a third time. The intersection position of edge* (B,C) *is replaced by the newer intersection, and the intersection state of this edge is reset to one forward cut. A renewed cut of the same edge, thus, results in a repetition of the described procedure.*

The trembling of the cut tool not only generates multiple edge intersections, but it can also affect faces. The tip of the cut tool can possibly penetrate a tetrahedron's face in quick succession. When entering a tetrahedron for the first time, its entry face has to be subdivided. If the tetrahedron is left again shortly afterwards, the introduced face subdivision does not yield much profit. Therefore, it is reversed as long as none of the edges of the face have been intersected. This procedure decreases the number of newly created mesh simplices and is integrated in the lookup table as the state transition from state O1 to state O3 and the state transitions O2→O1, A2→A1, B2→B1, and F2→F1 depicted in Fig. 3 as dotted arrows.

5.2. Pulling Cut Tool Back

When performing an incision it should be possible to stop the cut and pull the cut tool back. Most of the solution for this task is already provided by suppressing unnecessary edge subdivisions, as described at the beginning of this section. However, the faces in which the cut trajectory of the cut tool changes its direction to leave through the same edge as it entered have to be treated specially in order to find a practical turning point. For that purpose the trajectory of the intersection point between cut tool and face is sampled, and its incrementally calculated average position serves as a turning point. Fig. 10 (a) shows an example of an averaged position for the turning point.

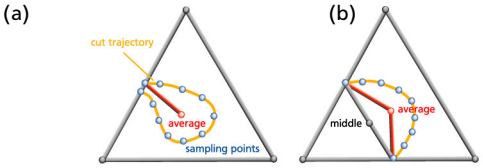

Figure 10: *Sampling for midface node position:* (a) *turning point,* (b) *face midpoint*

Although one could try to determine the farthest point inside the face as turning point, the averaged mid node has the additional advantage, that it can be used for all other face mid nodes too. In each completely split face, there exists a mid node that is used as long as the cut tool intersects the face, but has to be set to a meaningful position after the cut tool has left the face. The averaged position is practical for this purpose, it increases the smoothness of the modeled trajectory of the cut tool. Fig. 10 (b) compares the averaged mid node in a completely split face to an arithmetic middle point.

6. Recursive Continuation

While the described approach is effective for the large majority of tetrahedral intersections, it fails when the discretization assumption of only one intersection per edge and face is not met. In order to handle these cases as well, the algorithm continues the introduced state machine concept recursively. Whenever an intersection occurs that could not be handled with the basic state machine (i. e. a double cut edge) the affected tetrahedron gets completely subdivided, and the modeling of the intersections is continued on the sub tetrahedra level. The difficulty is guaranteeing the correct continuation of the interrupted cut surfaces on the sub tetrahedral level.

example 4: *hierarchical subdivision*

An example of a very simple hierarchical subdivision illustrates the idea of the hierarchical continuation of the state machine concept. Fig. 11 (a) shows a tetrahedron that is intersected over one edge. According to the state machine shown in Fig. 3, the corresponding state is A0. *A second intersection of the edge already cut, as shown in Fig. 11 (b), does not appear in the state diagram. For this reason the tetrahedron has to be completely subdivided to the end state*

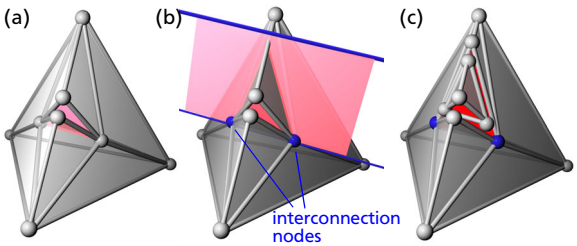

Figure 11: *Simple example for hierarchical subdivision*

A3 *of the last valid state* A0. *By applying the corresponding state transition, the tetrahedron is subdivided into six sub tetrahedra. A repeated collision detection step shows that the cut tool has left the tetrahedron and none of the sub tetrahedra is intersected with the cut tool anymore. Before processing the sub tetrahedra, the positions of the started intersection surface are marked as interconnection nodes, from where the intersection surface will be continued. In the given example, only the subtetrahedron, from which the cut tool has left, is affected and has to be updated. In order to construct a continuous intersection surface as depicted in Fig. 11 (c), the separated tetrahedron has to be split away from the two interconnection nodes.*

As one can see in this example, the continuation of the intersection surface involves a new topology of a tetrahedral intersection. Whereas tetrahedra have always been cut over edges, they can now be split along edges or over nodes.

6.1. Additional Types of Tetrahedral Subdivisions

Fig. 12 enumerates all additional topologies required for a consistent handling of the transitions between two hierarchy levels of tetrahedra. Two categories of topologies are distinguished: The first category includes tetrahedra that contain two interconnection points. Their intersections start along the edge that contains the two interconnection points. All tetrahedra that contain only one interconnection point fall into the second category, whose intersections begin over a tetrahedral node. Further differences between the depicted tetrahedra intersections depend on the number and position of the registered face intersections.

The continued intersection from *example 4* corresponds to the intersection topology BNN0 of Fig. 12 and its following state CNN0 in which the opposite edge is cut as well. The example shows that the cut topologies depicted in Fig. 12 are only the start states of the new topologies. If one includes all further possible edge and face intersections, each of the topologies of Fig. 12 results in a separate state machine, which has a similar structure as the basic state machine already described in Section 4. The realization of these additional state machines is a large effort and would require a new operator that splits existing corner nodes.

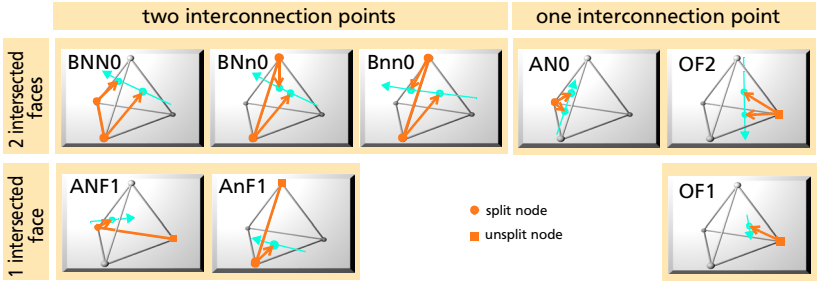

Figure 12: *Additional cut topologies*

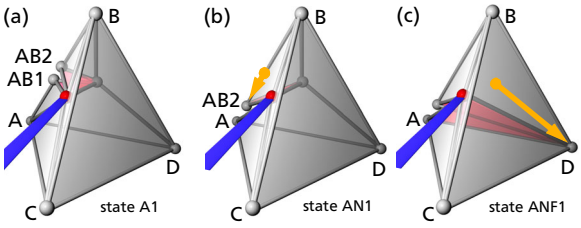

Figure 13: *Mapping of basic state to new state:* (a) *basic state A1,* (b) *merge edge mid-node,* (c) *merge face mid node*

6.2. Mapping onto States of the Basic State Machine

In order to avoid the costly implementation of eight additional state machines, the new state machines and their topologies are mapped onto the tetrahedral states of the basic state machine described in Section 4. The idea is to interpret the intersection topologies of the additional state machines as tetrahedral states, as they appear in the initial state machine diagram of Fig. 3. The split corner nodes of the new tetrahedral topologies are represented by nodes of an edge split that are shifted into the corner, and by shifted face mid nodes. One of the initial edge or face mid nodes is thereby merged with the corner node of the tetrahedron. Both of these merge operations are depicted in Fig. 13.

example 5: *mapping a basic topological state* A1 *to the new topologies* AN1 *and* ANF1

Fig. 13 (a) contains the state A1 *as it appears in the basic state machine of Fig. 3. As illustrated in Fig. 13 (b) the edge mid node* AB1 *is first mapped onto the corner node* A*, and then (c) the face mid node* ABD *is mapped onto corner node* D*. The achieved topology corresponds then, as desired, to that of state* ANF1.

The described mapping onto states of the basic state machine certainly simplifies the algorithms for the recursive continuation of the state machine considerably, but it also introduces a disadvantage. The inverse mapping from the detected new topologies to the topologies of the state machine of Fig. 3 is not unique for certain topologies. As a consequence the resulting algorithm is not able to consistently handle all situations of recursive subdivisions. In order to reach a closed algorithm, all eight additional state machines have to be realized separately by introducing the mentioned node split operator.

7. Applications

Various applications require the sort of accurate modeling of intersections which is described in this work. The following examples illustrate possible applications in the areas of visualization, surgical simulation and virtual sculpting.

7.1. Interactive Visualization of Volume Data

The 3D visualization of volume data-sets is a popular component in various application scenarios. In these scenarios, an interactive cutting of volume data-sets introduces a new intuitive interaction paradigm for volume data. It allows to clearly and simply create new interior views of a given data-set. In particular, the oil industry is interested in a fast and accurate method for processing large amounts of seismic data. The introduced intersection algorithm enables them to arbitrarily cut seismic volume data-sets along specific strata. Fig. 14 illustrates a sequence of interactions with a seismic data-set. After performing a first intersection (Fig. 14 (a)), the separated pieces can be selected and repositioned as depicted in Fig. 14 (b). The repositioning provides new views and enables further intersections. Intersection and realigning procedures can be continued until the region of interest is reached (Fig. 14. (c)-(e)). Fig. 14 (a) and (c) demonstrate that the intersection surface dynamically follows the cut tool. The intersection surface thereby is modeled up to the actual position of the cut-tool. Thus, the user gets a precise feedback of the cut-tool's current position within the volume. Additionally, the surrounding surface is transparent during an intersection in order to support the navigation in the data-set. Fig. 14 (c) illustrates the dynamic mesh subdivisions by displaying the underlying tetrahedral mesh.

7.2. Surgery Simulation

In surgical interventions, cutting is one of the most important tasks. The correct and precise execution of intersections is the basis for a successful surgical outcome, e.g. in tumor removal. There is a large need for simulators that enable physicians to train and rehearse operations, especially in the area of minimal invasive surgery. Fig. 15 presents the opening of skin tissue with a surgical scalpel, which is a task in open surgery. In this example, the skin tissue is simulated as prestressed deformable model.

7.3. Virtual Sculpting

The presented intersection algorithm is an intuitive tool for interactively processing meshed volumetric models, as they are used in many simulation environments. With such a tool, a mechanical component can be rapidly prototyped or adjusted. Fig. 16 presents the processing of an irregular tetrahedral model consisting of 3000 tetrahedra. In Fig. 16 (a) a first notch is cut-off. The intersection surface is drawn up to the current position of the cut-tool and the surface of the model is transparent during cutting. Fig. 16 (b) depicts the tetrahedral mesh representation after two further intersections. It is clearly visible how the node densitiy increases towards the intersection surfaces. This provides sufficient degrees of freedom for the accurate modeling of the performed intersections. In conclusion, Fig. 16 (c) presents the resulting model.

8. Conclusion and Future Work

We have presented an algorithm that consistently and accurately processes arbitrary intersections in tetrahedral meshes in real-time. The algorithm is based on a state machine, that tracks the topology of tetrahedra and controls their progressive subdivision. Three examples have exhibited the large field of applications for this algorithm.

The recursive continuation of the algorithms, that breaks the discretization assumption of only one intersection per edge and face could still be improved to ensure the convergence of the algorithm in all cases. Future work will mainly focus on this problem.

References

[1] C. Basdogan, Chih-Hao, and M. A. Srinivasan. "Simulation of tissue cutting and bleeding for laparoscopic surgery using auxiliary surfaces." In *Medicine Meets Virtual Reality*, pages 38–44. IOS Press, 1999.

[2] D. Bielser and M. H. Gross. "Interactive simulation of surgical cut procedures." In *Proceedings of the Pacific Graphics 2000*, pages 116–125, 2000.

[3] D. Bielser and M. H. Gross. "Open surgery simulation." In *to appear in: Proceedings of Medicine Meets Virtual Reality 2002*, 2002.

[4] D. Bielser, V. A. Maiwald, and M. H. Gross. "Interactive cuts through 3-dimensional soft tissue." In *Proceedings of the Eurographics '99*, volume 18, pages C31–C38, 1999.

[5] C. D. Bruyns and K. Montgomery. "Generalized interactions using virtual tools within the spring framework: Probing, piercing, cauterizing, and ablating." In *Medicine Meets Virtual Reality 02/10*, pages 74–78, 2002.

[6] C. D. Bruyns and K. Montgomery. "Generalized interactions using virtual tools within the spring framework: Cutting." In *Medicine Meets Virtual Reality 02/10*, pages 79–85, 2002.

[7] S. Cotin, H. Delingette, and N. Ayache. "A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation." *The Visual Computer*, 16(8):437–452, 2000.

[8] F. Ganovelli, P. Cignoni, C. Montani, and R. Scopigno. "A multiresolution model for soft objects supporting interactive cuts and lacerations." In *Proceedings of the Eurographics 2000*, volume 19, pages C271–C282, 2000.

[9] F. Ganovelli, P. Cignoni, C. Montani, and R. Scopigno. "Enabling cuts on multiresolution representation." *The Visual Computer*, pages 274–286, 2001.

[10] S. Gibson, J. Samosky, A. Mor, C. Fyock, E. Grimson, T. Kanade, R. Kikinis, H. Lauer, N. McKenzie, S. Nakajima, H. Ohkami, R. Osborne, and A. Sawada. "Simulation arthroscopic knee surgery using volumetric object representations, real-time volume rendering and haptic feedback." In *Proceedings, CVRMed II and MRCAS III*, Mar. 1997.

[11] A. B. Mor and T. Kanade. "Modifying soft tissue models: Progressive cutting with minimal new element creation." In *CVRMed-Proceedings*, volume 19, pages 598–607, 2000.

[12] H.-W. Nienhuys. "Supporting cuts and finite element deformation in interactive surgery simulation." In *may appear in: TransVCG*, 2002.

[13] H.-W. Nienhuys and A. F. van der Stappen. "Combining finite element deformation with cutting for surgery simulations." In *Proceedings of the Eurographics '00*, pages 274–277, 2000.

[14] D. Serby, M. Harders, and G. Székely. "A new approach to cutting into finite element models." In *Procs. of the Fourth International Conference on Medical Image*, pages 425–433, 2001.

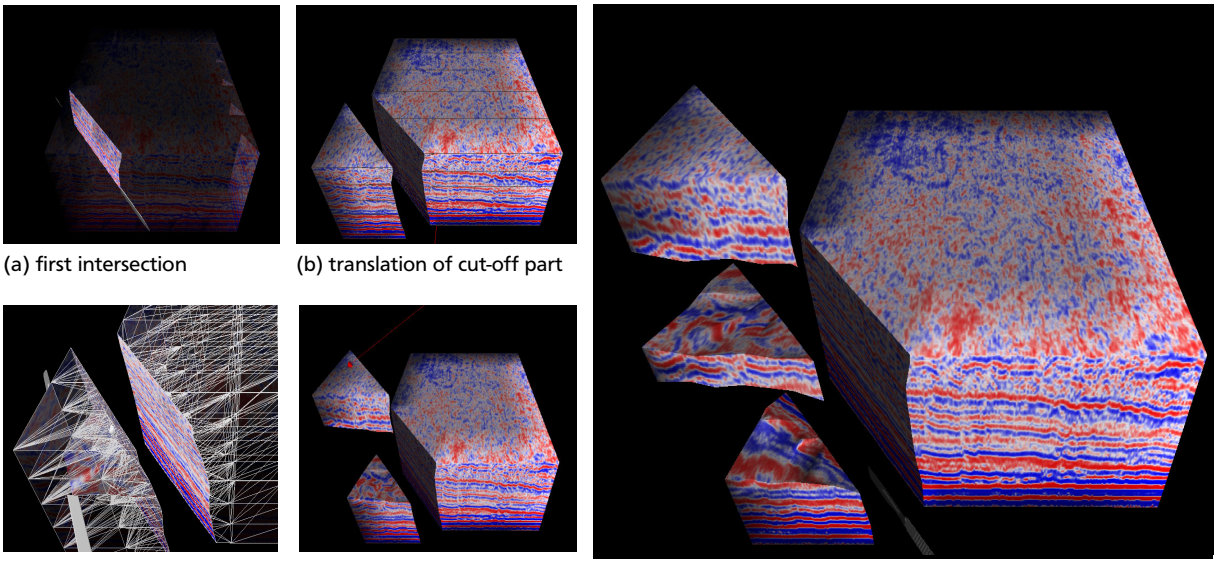

Figure 14: *Interactive visualization of a seismic volume data-set*

Figure 15: *Simulation of open surgery*

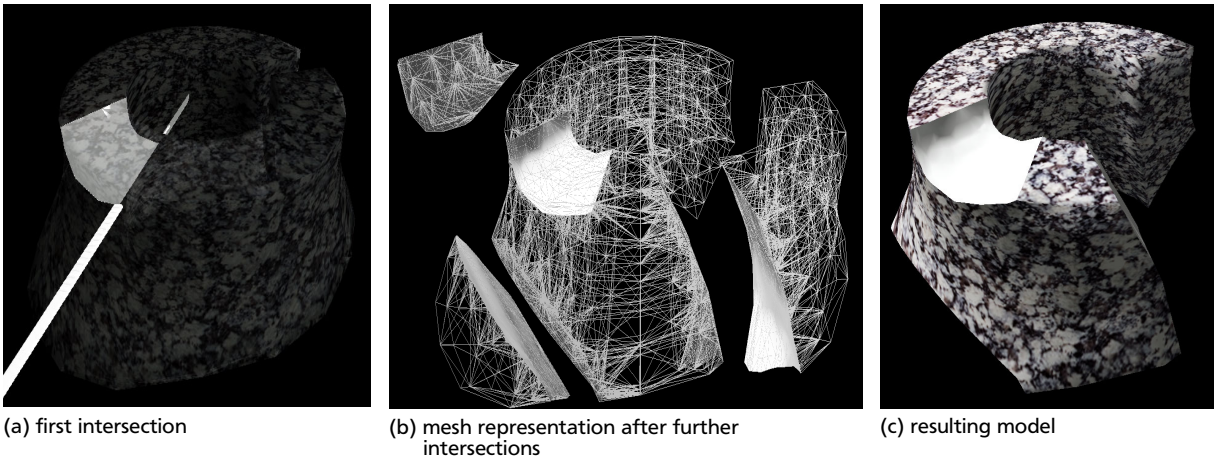

Figure 16: *Virtual sculpting*