Simulating Drilling on Tetrahedral Meshes.

G. Turini [1] F. Ganovelli[2] C. Montani [2]

[1] Endocas Centre of Excellence for Computer Aided Surgery, Pisa, Italy
[2] Visual Computing Laboratory, ISTI-CNR Pisa, Italy

Abstract

Bone drilling is a fundamental task in several surgical procedures, including mastoidectomy, cochlear implantation, orbital surgery. It consists in eroding the part of the bone in contact with the tip of the surgical tool when a sufficient pressure is exerted.
Since the bone is an almost rigid material, the bone drilling simulations usually employ voxel-based representations of the bone, so that it is easy to show material removal by playing with material density in the voxels. Unfortunately, there are cases in which drilling is only a part of the task, and parts of the same object are also cut away or, worse, the bone is slightly deformable and therefore voxel-based representations do not work well.
We propose a novel method to simulate drilling on objects represented explicitly by means of a tetrahedral mesh. The key idea of our method is to create an alternative representation of the tetrahedron when it is partially eroded. Such representation consists of a set of smaller tetrahedra obtained by a hierarchical decomposition of the original one, and combined to represent the current status of the erosion.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: Animation I.3.5 [Computational Geometry and Object Modeling]: Physically based modeling I.3.5 [Computational Geometry and Object Modeling]: Hierarchy and geometric transformations

1. Introduction and Related Work.

Drilling, like cutting, is a task that changes the shape and possibly the topology of an object. If the object is static or at least rigid, volumetric methods employing regular decomposition of the space are a natural choice. Unfortunately, this kind of methods is hardly usable if the object is deformable. In this paper we propose a method to perform drilling on objects represented by means of tetrahedral meshes. A vast literature exists both on methods for deriving a tetrahedral mesh from CAT/RM data and for performing physical simulation of solids employing tetrahedral meshes. Tetrahedral meshes are easy to render (since their border is made of triangles) and can be used to simulate material with non uniform density thanks to the fact that, in contrast with surface meshes, the volume is explicitly represented.

If the object is densely discretized, one trivial way to remove material is simply to remove tetrahedra, as in [CA99]. However, the number of tetrahedra strongly influences the performance of the physical simulation and therefore we have two opposite constraints: a high number of tetrahedra for rendering drilling and a low number for physical simulation. We decouple the drilling from the physical simulation by allowing the representation of the drilling at sub-tetrahedral level. This representation is defined by a hierarchical decomposition scheme of the tetrahedron, obtained by recursively splitting tetrahedra in 8 new tetrahedra. When a tetrahedron of the original mesh come to contact with the tool, this scheme is applied to adapt the granularity of the description to those of the tool, so that the removal of the material can be done by simply removing tetrahedra. The advantages of our approach are twofold: 1) it is independent on the method adopted for physical simulation. 2) collision detection, haptic and visual rendering rendering are modeled at sub-tetrahedral level.

Most of the approaches in the present literature consider the object as rigid and therefore suitable for a regular discretization of the space. In [MPT99] the space containing static objects is partitioned into voxels and the surface of the moving objects is sampled with point and normals (hence the name *voxmap-pointshell*). When a point is inside

a non-empty voxel, penetration and, consequently, feedback is computed by considering the *tangent plane*, i.e. the plane passing through the center of the voxel and oriented like the point normal. In [PPT*02] this approach has been extended to represent material subtraction in the simulation of petrous bone surgery. In [MSB*04] the voxmap is used for haptic rendering while visual rendering is achieved by building a triangular mesh with vertices placed in the voxels vertices and normal and texture coordinates derived from the information contained in the voxel. In [AGG*02] the volume is stored as a stack of textures rendered back to front. Eroding a region consists of making the proper set of texels transparent, while a correct lighting can be obtained during fragment processing. With respect to the previous approaches, the latter makes a better use of the graphics hardware and is therefore more efficient, even if limited by a stricter bound for memory usage. Mesh based approaches are widely used for performing cuts [BMG99, GCMS00, NvdS00] and in some case also for rendering material removal [AGP*06, SC98] but the accuracy of the cut is strictly dependant on the the size of the mesh elements.

2. Our Approach

Our method considers each tetrahedron as the root of a hierarchy where each node is a tetrahedron and its children are obtained by refining the parent node in 8 smaller tetrahedra as shown in Figure 1. Every frontier in this hierarchy, i.e. every set of tetrahedra such that on all paths from the root to the leaves exactly one node is contained in the set, corresponds to a representation of the tetrahedron itself. We represent the subtraction of a region of a tetrahedron by computing a frontier in the hierarchy such that no tetrahedra bigger than a given threshold crosses the border of the region and by tagging the nodes of the hierarchy inside the region as *erased*.

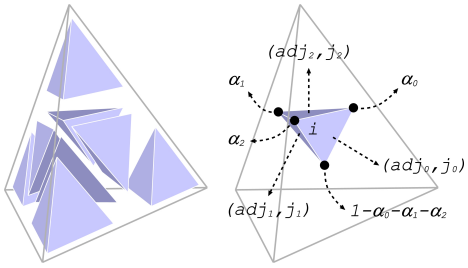

Figure 1: *Tetrahedral subdivision scheme.*

2.1. Data structures and basic operations on hierarchies

Our method relies on two basic operations: *refine* a node and *erase* a node. In both cases it must be determined which faces of the tetrahedra in the frontier are (even partially) visible and therefore are to be rendered and considered for collision detection. In other words, for each face of a tetrahedron we need to know if it is *covered* by the union of all the other tetrahedra in the frontier. If not, the face is visible.

If the frontier of the hierarchy was a simplicial complex the solution would be trivial: for each tetrahedron t belonging to the frontier, if the tetrahedron adjacent to t by the face i is not erased, then the face i is not visible. Our case is more complicated because a tetrahedron can be adjacent to several tetrahedra by the same face (see node **a** in bottom left of Figure 3).

At the beginning, when the frontier is made by the root of the hierarchy (the original tetrahedron) the visible faces of the original tetrahedron are those not shared by any other tetrahedron in the mesh. We describe how to keep the visible set of faces up to date after each operation.

Data Structure Each node of the hierarchy contains an integer index (unique within a tree) that encodes its topological location, a pointer to each of the children nodes and to the parent node, four bits to indicate which faces of the corresponding tetrahedron are visible and one bit to indicate if the tetrahedron has been erased. The index of the root is 0, the children of a node i are numbered from $8\,i+1$ to $8\,(i+1)$ and the parent of the node i is numbered $\lfloor i/8 \rfloor$. The part of hierarchy actually instantiated is only the part from the root to the frontier. To render visible faces, we should visit the hierarchy at every frame and render the faces marked as visible. Instead we link the non erased nodes in a list, called *v_list* (see Figure 2), that we keep up to date during the Erase and Split operations. Therefore two more pointers are added to the node data structure.

We also use a static table, named *Tree Table*, containing all the data we need that can be derived once for all from the subdivision scheme. The id^{th} row contains: four couples $(index, face)$ identifying the nodes at the same level of the hierarchy ($log_8\, i$) adjacent to id and the index of the face they share; four triples of floats identifying the barycenter coordinates of the four vertices of the tetrahedron id with respect to the original tetrahedron (the node 0). We also number the faces of the tetrahedra so that the following holds: if the face i of the tetrahedron t lies on a face of the parent tetrahedron p, then the index of this face in p is i as well.

Let t a tetrahedron and $i|i = 0..3$ its four faces. We define the *Neighbors Tree* $NTree(t,i)$ (see Figure 3) as the tree made of the path from the first common ancestor between t and $adj(t,i)$ to the node t (note that the face i of every node in this path includes the face i of t) plus the subtree rooted at t restricted to the nodes with a face lying on the face (t,i). Please note that the *Ntree* is not explicitly built, this is simply notation added to specify a particular visit of the hierarchy.

Operation **Erase**.
For each face (t,i), access the table at the row t and find out the index $adj(t,i)$ of the node adjacent to t by i and the corresponding face $j \in \{0..3\}$. Then visit the nodes of

the $NTree(adj(t,i),j)$ in any order (depth first for example), setting to visible all the leaf nodes encountered and adding them to the *v_list*, if not already present. Finally remove t from the *v_list*.

Operation **Refine**.
If the face (t,i) is not visible, then all the faces indexed i of the eight children are not visible. If (t,i) is visible, we could conservatively set as visible all the faces i of its children, but some of them can actually be not visible according to our definition (see bottom left part of Figure 3). So, for each one of the four couples (c,i) referring tetrahedra with faces lying on (t,i) fetch the adjacent (node,face) $(adj(c,i),j)$ and visit the $NTree(adj(c,i),j)$. If at least an erased leaf is encountered in this visit, the face (c,i) is set to visible.

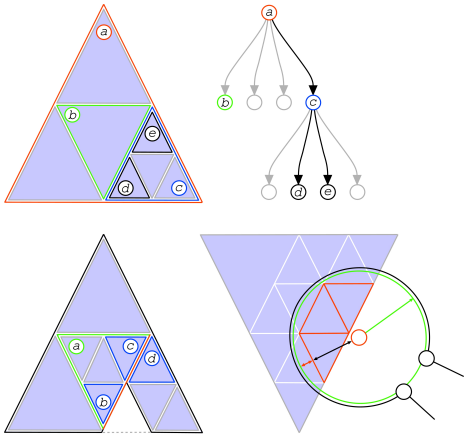

Figure 3: *In this figure we scale down to two dimensions to simplify the illustration. Top: the NTree for the node adjacent to* ***a*** *by the rightmost face. Note that the nodes in the tree are all the nodes with a face partly in common with the face of* ***a****. Bottom: the node* ***a*** *is being refined. The highlighted part is visible but not all its children have the face visible, since* ***c*** *and* ***d*** *are both in the frontier and not erased.*

3. The system

The simulator developed is a *multi-threaded* application formed by three threads: the visual rendering thread, the collision detection thread and the haptic thread.

3.1. Rendering thread

At this stage the rendering is quite naive. We simply render all the visible faces of tetrahedra. If a tetrahedron has been eroded, then we run through its *v_list* and render each visible face of the nodes in the list. We employ a simple optimization consisting in compiling a display for each tetrahedron, containing the rendering of its *v_list*. The display list is recompiled every time the *v_list* is modified and called to render the tetrahedron when it is not being eroded.

3.2. Collision handling thread

Collision detection is a crucial task because it determines all the actions to be performed on the tetrahedra (Erase and Refine) and the feedback to return to the haptic interface. The surface of the drilling instrument is sampled with an evenly distributed set of *sample points*. The sample points can be active, i.e. belonging to the eroding part of the tool, or passive. A *sample point* has: a position, an *a-flag* indicating whether it is active, a radius to define its sphere of action, an *i-flag* to dynamically mark the points that interact with the mesh. To test for collision detection, we use the sample points placed on the surface of the tool and the spatial hashing technique proposed in [THM*03]. The thread's cycle starts by updating tool position and direction.
Then we use the *hash table* to detect a subset T of border tetrahedra (i.e. tetrahedra with at least a visible face) possibly intersecting with the tool. If T is empty we are done. Otherwise we analyze all the active *sample points* on the surface of the tool (set A).
For each point in $a \in A$ and each tetrahedron in $t \in T$ we found out whether t crosses the border of the sphere with radius r centered at a. If it does, we refine the tetrahedron and recur to the children nodes. This strategy requires a dynamic updating of the *v_list* with the insertion of the newly generated visible nodes.

The refinement ends when all the nodes in the frontier are no further refinable: this happens if the maximum depth d_{max} has been reached or when their surface area is lower than a given threshold a_{min}.
When refinement is completed, the *v_list* nodes that are located within the erosion sphere are eroded. The erosion procedure consists of applying the *Erase* operation to all such nodes.

During collision detection we also compute the penetration depth and direction to render forces. The passive sample points are tracked frame by frame, and the segment formed by two consecutive point positions is tested for intersection with the visible faces in T. For the active sample points we compute the penetration as the difference between the radius of the active sample point and its distance to the mesh.
This makes easy to handle collisions when the eroding part is a sphere, because we use a single active point in the sphere center with radius equal to the radius of the sphere (see bottom-right part of Figure 3).

3.3. Haptic interaction thread

The haptic interface management thread essentially updates the tool accordingly to the haptic interface movements and ensure the high rendering frequency of force feedback. This last requirement is essential to simulate a realistic haptic interaction.
Since the collision detection cycle, on which penetration depth depends, has an updating rate too low, we use extrap-

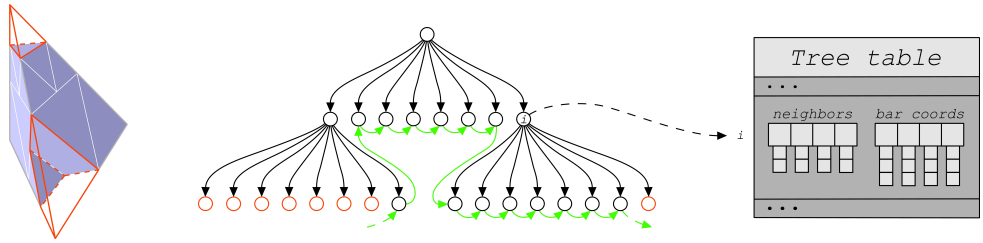

Figure 2: *A tetrahedron eroded (left), its hierarchy (center) and the TreeTable (right).*

olation to compute the forces to give to the haptic interface at high frequency (around 500Hz).

4. Results

Our early tests concern the collision detection, the handling of geometry and the updating rate of the three threads. The object is still static, even if this is not assumed anywhere in the implementation.

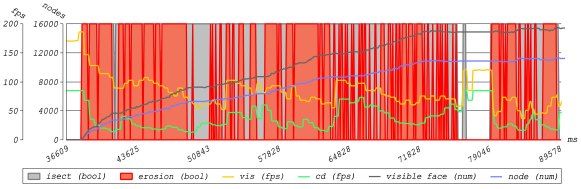

Figure 4: *Data produced during the recording of the video **cube**: the number of nodes and visible faces and the updating rates of the visualization and collision handling threads.*

All the tests were run on a Intel Pentium IV with 1G of RAM and a nVidia 6800 graphics card. Figure 4 shows the performance of the algorithm in a test case made of a cube decomposed in 12 tetrahedra by adding a vertex in the cube center. As can be seen in Figure 4.(a) the number of tetrahedra (i.e. nodes) increases rapidly during the erosion. However, this growth does not influence the performance of the method as one could expect. In terms of memory occupation, the node requires only 5 bytes instead of the 16 required by the tetrahedron to store the pointer to the vertices. In terms of rendering time, see Figure 4.(b), the visible faces of the nodes of the tetrahedron are all contained in the same display associated with the tetrahedron, which is recompiled only when its tree is being modified. One drawback of the current version of the method is the rendering of the eroded regions. Since each face is flat shaded, it is easy to spot the triangles and the appearance is jagged. This effect is only mildly mitigated by a greater number of refinement steps, or, equivalently, by a greater distance of the region to the point of view. Figure 5 shows a snapshot of the cube at the end of the process.

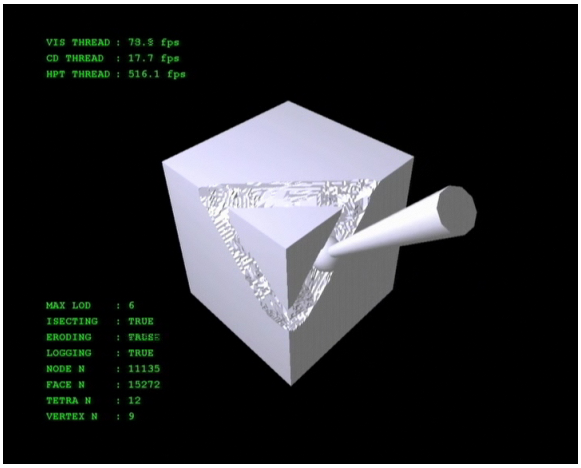

Figure 5: *A screenshot of the video **cube**.*

5. Conclusions and Future Work

This paper presents a novel non volumetric approach to simulate drilling on tetrahedral meshes. In contrast to existent methods, the object under drilling does not need to be rigid, and no assumption is done on the way the physics is simulated. Although the method is fairly efficient, a technique for a more accurate and possibly efficient visualization of the drilled regions is under investigation. A second issue concerns the topological and physical changes due to erosion. As previously said, the sub-tetrahedron subdivision in used only for visual and haptic rendering. This implies that even if the erosion causes the splitting of the object in two parts, these will still be connected as they were before. Fortunately we can use the stored hierarchy of nodes to find out if a single tetrahedron has been split and consequently update the physical simulation. These algorithms are being integrated at the time of this writing.

References

[AGG*02] AGUS M., GIACHETTI A., GOBBETTI E., ZANETTI G., ZORCOLO A.: Real-time haptic and visual simulation of bone dissection. In *IEEE Virtual Reality Conference* (Conference held in Orlando, FL, USA, March 24–28, Feb. 2002), pub-IEEE, pp. 209–216.

[AGP*06] AGUS M., GOBBETTI E., PINTORE G., ZANETTI G., ZORCOLO A.: Real-time cataract surgery simulation for training. In *Eurographics Italian Chapter Conference* (Conference held in Catania, Italy, 2006), Eurographics Association.

[BMG99] BIELSER D., MAIWALD V., GROSS M.: Interactive cuts through 3-dimensional soft tissue. *Computer Graphics Forum (Eurographics'99 Proc.) 18*, 3 (Sept. 1999), C31–C38.

[CA99] COTIN H. D. S., AYACHE N.: A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. In *CAS99 Proceedings* (May 1999), pp. 70–81.

[GCMS00] GANOVELLI F., CIGNONI P., MONTANI C., SCOPIGNO R.: Enabling cuts in multiresolution representation. In *CGI 2000 Proceedings* (2000), N.Magnenat-Thalmann, D.Thalmann, (Eds.), p. (in press).

[MPT99] MCNEELY W. A., PUTERBAUGH K. D., TROY J. J.: Six degree-of-freedom haptic rendering using voxel sampling. In *SIGGRAPH* (1999), pp. 401–408.

[MSB*04] MORRIS D., SEWELL C., BLEVINS N., BARBAGLI F., SALISBURY K.: A collaborative virtual environment for the simulation of temporal bone surgery. In *MICCAI (2)* (2004), Barillot C., Haynor D. R., Hellier P., (Eds.), vol. 3217 of *Lecture Notes in Computer Science*, Springer, pp. 319–327.

[NvdS00] NIENHUYS H.-W., VAN DER STAPPEN A. F.: Combining finite element deformation with cutting for surgery simulations. In *EuroGraphics Short Presentations* (2000), de Sousa A., Torres J., (Eds.), pp. 43–52.

[PPT*02] PETERSIK A., PFLESSER B., TIEDE U., HÖHNE K. H., LEUWER R.: Haptic volume interaction with anatomic models at sub-voxel resolution. In *Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems* (2002), pp. 66–72.

[SC98] STEPHANE COTIN HERVE DELINGETTE N. A.: *Efficient Linear Elastic Models of Soft Tissues for real-time surgery simulation*. Tech. rep., Institut National de Recherche en Informoatique et en Automatique, 1998.

[THM*03] TESCHNER M., HEIDELBERGER B., MÜLLER M., POMERANETS D., GROSS M.: Optimized spatial hashing for collision detection of deformable objects. In *Proceedings of the Conference on Vision, Modeling and Visualization 2003 (VMV-03)* (Berlin, Nov. 19–21 2003), Ertl T., Girod B., Greiner G., Niemann H., Seidel H.-P., Steinbach E., Westermann R., (Eds.), Aka GmbH, pp. 47–54.