MESH CUTTING ALGORITHM FOR USE IN AN ORTHOPAEDIC SURGERY SIMULATOR

Boothroyd, A; Gledhill, D; De Luca, D *
*University of Huddersfield **
Huddersfield, UK

ABSTRACT

In order to develop a podiatric orthopaedic training simulator the obstacle of simulating podiatric bone surgery must be overcome. In order to simulate this surgery appropriately, it is necessary to be able to cut through a virtual representation of a patient's foot on-screen in real-time. We investigate several methods of cutting through simulated objects in general, and evaluate their usefulness in simulating real-time interactive bone surgery. We determine that none of these conventional methods are fully suitable and instead propose, develop and test a method using planar slicing of polyhedral mesh geometry.

KEYWORDS

mesh cutting, surgery, simulator, orthopaedics, podiatry, serious games

1. INTRODUCTION

Orthopaedic foot surgery is carried out by podiatric surgeons and orthopaedic surgeons, and is used to treat a variety of ailments affecting the bones of the lower extremities. The goal of this research is to lay the foundation for developing a full-featured serious game simulating podiatric orthopaedic surgery, by overcoming the obstacle of accurately simulating cutting through a bone. Such a simulator could be used to train the next generation of surgeons in a safe, controlled environment (Haluck et al. 2001).

In order to perform surgical manipulation of this type, it is necessary to be able to cut through polygon meshes in a manner simulating cutting through actual bone.

Typically, in 3D graphics, meshes are either rendered unmodified (for instance, with static scenery) or animated. Meshes can be animated in a variety of ways, including simple geometric transformations and skeletal animation. Generally this involves replaying pre-recorded animations, although some animation systems generate animations on-the-fly in response to the mesh's environment and stimuli, for example the Euphoria dynamic animation engine (McEachern 2008). However, even with dynamically generated animations and interactions, the meshes are generally animated using a pre-determined 'skin' and a collection of jointed 'bones', to provide a full range of motion. Such methods can only move vertices around and not normally break or cut into the mesh.

It is necessary to perform geometric computations to deform the shape of the mesh in real-time, whilst maintaining an adequate frame rate and the illusion of the mesh being a solid object state of the art hardware.

This paper evaluates different existing methods used to cut through meshes and then proposes a new method using planar slicing allowing real-time mesh cutting.

The work was undertaken in the University of Huddersfield in house computer games studio, Canalside Studios. The studio has published both games and serious games including visualisation software and object handlers (De Luca & Taylor 2012). Through translational research methodology the studio rationale is to help entrepreneurial academics and research groups realise their potential through commercial development.

Translational research is commonly associated with the life sciences, however translational research methods are currently being applied across many subject areas allowing high impact software and products to be realised from research groups across many disciplines. We see translational research as a means of ensuring that research deliverables can be exploited by the commercial community and that all parties involved obtain the best transfer of knowledge with a two-way flow of information resulting in best practice and an accelerated product development timeline.

The UK Technology Strategy Board (TSB) defines the following when supporting innovations in the life sciences (Wamae et al. n.d.):

- Translational Research - the new scientific methods and technologies, interdisciplinary approaches, and collaborative institutional arrangements being developed to narrow the gap between basic science and its application to product and process innovation.
- Knowledge Exchange - the multi-directional flow of information of all kinds that is required as a basis for decision making in the translational research process.
- Value chain - the range of activities required to bring a product or service from conception, through the different phases of production, to delivery to consumers.
- Value system – the wider system within which the value chain operates including: policy and regulation, finance and markets, and public and stakeholder perspectives.

2. SURGICAL SIMULATION

2.1. Soft-tissue and Bone Simulation

(Choi et al. 2002) proposed and developed a system of deforming soft tissue in real-time using a mass- spring elasticity model. Earlier simulations were typically computationally intensive, not capable of running at the refresh rates required for a real-time simulation (Delingette 1998). The authors solved this using a force-propagation algorithm.

(Pflesser et al. 2000) presented a volume rendering system for visualizing bone dissections (such as mastoidectomies), which uses a sub-voxel rendering method to accurately simulate sections of bone being cut. A further development as described by (Petersik et al. 2002) incorporates haptic feedback to make the system more realistic. Similar systems are presented by (Morris et al. 2004; Morris et al. 2006), again using volume rendering to provide haptic feedback. However, this system employs a hybrid data structure for the rendering, which produces a triangle mesh from the volumetric model data for conventional rendering. The vertex positions are inferred from the volumetric data, using pre-calculated normal and texture co-ordinate information. Cutting is only performed on the volume data, with the polygon mesh being regenerated from the resulting voxels. As the cutting performed with this system simulates that of a burr a volume data structure is well suited to this task, as the volumetric nature of the geometry allows arbitrary shaped volumes to be removed at will. Similar volumetric systems are also employed by (Agus et al. 2002; Agus et al. 2003).

Cuts performed through bone with an electric saw do not typically result in an elastic deformation of the bone as when cutting through soft tissue — cuts will typically be straight, and remove a narrow channel of bone in front of the saw blade. Therefore, the mass-spring elasticity algorithms for deforming soft tissue will not produce a suitable result when simulating bone.

Volumetric approaches do not provide optimal solutions for the kind of cutting we intend to perform. This is partly because of the performance costs and complexity associated with volume rendering. The cuts we intend to simulate will not be modeled very accurately or efficiently by a volumetric model. There are three reasons for why we believe this to be the case: first, only a very narrow sliver of bone matter will be removed when cutting through a model, so the vast amount of data in a volumetric model would be unused. Second, it will be necessary for part of the volume to be severed, moved and reattached at a different angle. The movement of a severed piece of bone could be simulated within the same volumetric model as the rest of the bone, in which case a large amount of data must be moved around per frame and likely cause a large performance loss. Alternatively, the severed bone could be split off into its own volumetric model, which would be more performant, but would entail running two separate volumetric models in very close proximity. Third, when cutting through a volumetric model at any non-axis-aligned angle, a ‘stair-step’ of voxels will be left behind by the cut. Using a volumetric model of uniform density with the required fine granularity would be inefficient, as cutting operations will only involve a small part of the model. To limit overhead of increased density, a low-density model could be used, increasing the density around the cut as necessary. Voxel subdivision could increase density in these relevant areas.

2.2. Mesh Cutting

We propose a solution combining the geometric nature of the soft-tissue simulation with the non-deforming nature of the bone simulation, modeling the patient's foot as a 3D polygon mesh and respond to the cutting action of the user by employing geometric methods to remove matter from the mesh.

2.2.1. Mesh Sculpting

One of the simplest methods of altering a 3D polygon mesh is to sculpt its surfaces. Of particular note, one of the earliest computer sculpting systems allowed the user to control the sculpting operation using a polyhedral tool (Parent 1977). The system we propose will support multiple polygon-mesh-based tools based on real-world equivalent surgical tools tools that the user can control.

In most early sculpting systems, standard keyboard and mouse controllers were used to control the sculpting tool. However, (LeBlanc et al. 1991) proposed a system using the Spaceball 6D human interface device, which allows the user 6 degrees of freedom in their input. This device is used with one hand, and a computer mouse in the other hand, to more accurately perform sculpting. We use a similar, but more modern, device (the Sensable Phantom) to allow the user to control our system.

To perform sculpting, Parent used 'decay functions' to determine the effect of the user's input on individual vertices, based on proximity to the position of the user's tool. A refined technique was applied to the whole mesh by (Allan et al. 1988), and later refined again into an adaptive subdivision by (Bill 1994), in order to afford the user a finer degree of control over the manipulated surface.

2.2.2. Metaballs

Metaballs are a class of implicit algebraically-described geometries, whose surfaces contain points satisfying one or more functions of the form $F(x,y,z)=0$. Computer-rendered algebraic geometries of this nature were initially described by (Blinn 1982), with what he termed "blobby" modeling. Methods for rendering parameterized surfaces individually were already well-known at this point (Lane et al. 1980), but Blinn blended multiple surfaces together. Lone metaballs will produce a single geometric shape, typically a sphere or ellipsoid, but large groups of metaballs can be used to represent more complex shapes. Since the surfaces created by metaballs are the result of blended continuous algebraic functions (as opposed to the discrete geometric segments in polygon mesh geometry), they provide a smooth transition between their different functions and as a result produce very smooth, soft-looking structures.

It should be noted that the influence provided by a given metaball need not be positive. In order to 'cut through' a metaball surface, it is possible to define metaballs with negative influence. These negative metaballs create indentations in positive metaball geometry, by reducing the total influence of points within an area below the required threshold.

Whilst the output of a metaball system can be a polygon mesh, the input can not. Traditional polyhedral geometries cannot be automatically converted to metaball-based structures. An approximation could be made using numerous tiny metaballs, however, using this approach would be very inefficient, with an excessively large number of metaballs being required. As such, in order to use metaballs within an orthopedic simulation, a model of the patient's foot must first be constructed entirely from metaballs. This would require a modelling expert to be available for the laborious task of remodeling all polyhedral meshes for the system.

2.2.3. Constructive Solid Geometry

Constructive Solid Geometry (CSG) is a method of defining geometry in terms of two or more other geometries combined using Boolean set operations (Requicha et al. 1978). The three binary set operations used in CSG are union (or addition), difference (or subtraction), and intersection, which are applied to the volume enclosed by the combined geometries. The surface produced as a result of these operations contains the volume previously occupied by either object, only one object, or both objects, respectively. Entire geometries can be represented by a hierarchy of CSG operations performed on implicit primitives and the results of previous operations, forming a binary tree of set operations (Requicha 1980). The terminal nodes in such a tree represent implicit primitives, and the nonterminal nodes represent a single binary set operation performed on the two child nodes. It allows highly accurate smooth surfaces and intersections to be represented. A polygonization of the CSG tree will necessarily reduce the level of detail and accuracy in the rendering, but its approximate nature is considered an acceptable tradeoff since it allows the geometry to be rendered at refresh rates suitable for interactive viewing and manipulation of the CSG tree, however, polygon-mesh geometry cannot be easily converted to CSG format. Therefore, reference data of medically-accurate foot models represented as polygon meshes cannot be used as a CSG tree.

2.2.4. Polyhedral Boolean Set Operations

Operations topologically equivalent to the Boolean set operations used in CSG trees may be applied directly to polygon mesh geometry (Aftosmis et al. 1998). When applied to polyhedra, the boundary representation of each object's surface is segmented based on its intersection with the other object. The segments of each object are then combined into a new polygon mesh or discarded, based on their relationship to the other object. Of particular relevance is the Boolean subtraction (difference) operation, which forms a new object containing the volume enclosed by one object but unenclosed by the other. This Boolean subtraction operation is commonly available in 3D-modelling software, and has also been used in industrial simulation software — for example, in simulated log sawing (Occena & Tanchoco 1988).

Boolean subtraction has also been used to simulate soft-tissue cutting in surgical operations. In a system used to simulate operating on gunshot wounds (Delp et al. 1997), the user can manipulate a virtual scalpel, used as an operand in a Boolean subtraction, to cut through soft tissue.

2.2.5. Conclusion

Four distinct methods widely used to manipulate geometry could potentially be used to simulate cutting through bone. Mesh morphing, as commonly used, does not provide the necessary control over the deformed polygon mesh to realistically simulate cutting. Metaballs would provide a simple method of representing the path of a cut through bone — however, they are computationally expensive to modify at real-time refresh rates, and cannot be used with polygon mesh representations of medically-accurate foot models. CSG trees present similar benefits and drawbacks to metaballs, but allow more accurate, sharply-defined cuts to be performed. The Boolean difference operation, commonly used in CSG trees, may be applied instead to polyhedral mesh-based geometry, with a modified algorithm that operates on the surfaces of the geometry.

3. SIMPLIFICATION OF SERIAL BOOLEAN OPERATIONS

3.1. Cutting Cross-sections

Consider the case where one or a series of Boolean subtractions (using a convex polyhedral cutting volume) cut through the complete cross-section of a convex triangle mesh severing to form two new meshes. The newly-hewn sides of the two new meshes (which previously intersected the cutting volume) are defined by two surfaces. These surfaces are bounded by the curves of intersection formed between the sides of the cutting volume and the mesh.

The geometry that lies between the two cutting surfaces would be discarded, and the rest would be added to whichever of the two newly-created meshes is appropriate, based on its position relative to the surfaces. The geometry of each cutting surface would be appended to the new mesh adjacent to it.

3.2. Geometry-in-polyhedron Testing

It is generally necessary to test all the polygons in each mesh to determine which lie entirely outside or inside of the other mesh, and those that lie partially inside and outside the other. The test can be performed in a number of ways — the easiest being the ray-cast method.

Ray-casting for the purpose of collision detection calculates the intersection of a hypothetical 'ray' with object geometry. In order to determine whether a given point lies within a given triangular mesh or not using ray-casting, rays must be emitted from the point, and the number of triangles the ray intersects counted. A simple test with a closed mesh can be performed using only two rays, both emitted in opposite directions — if both rays pass through an odd number of surfaces, then the point is enclosed on both sides. Where the number of surfaces is even or zero, the point is outside the mesh.

All polygons in each mesh must be tested to determine whether they intersect the other mesh, which entails testing all vertices in each mesh. Assuming a suitable spatial subdivision technique is used to optimise the ray test, each test will be of O(1) complexity in the average case, and O(n) in the worst case (Szirmay-Kalos & Márton 1998). Since all vertices must be tested per Boolean operation, this results in an overall complexity of O(n) (average case) or O(n^2) (worst case) for the intersection tests.

The remainder of the algorithm will require classifying all polygons based on their intersection or non-intersection of the opposing mesh (O(n)), adding each new vertex or index (reallocating buffers each time would be O(n^2); with optimised buffers this is closer to O(n)) and removing deleted vertices and indices (O(n)).

3.3. Segmentation of Triangles

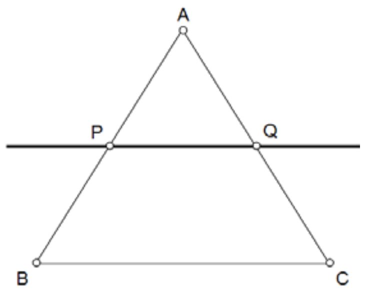


Figure 1: Before splitting a triangle along a plane (line of intersection with plane marked as thick line)

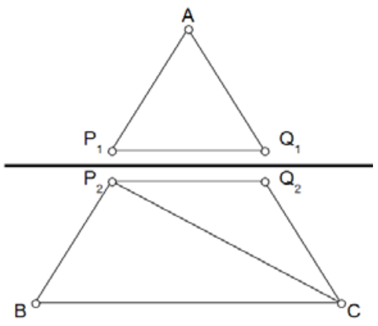


Figure 2: After triangle is split

In its simplest form, a mesh is divided along a single infinite plane, as in Figure 1. Those ‘above’ or ‘below’ the plane relative to its normal are placed into the relevant group; those lying on the plane are split along it. This usually results in three additional polygons being generated per polygon sliced — Figure 2.

The points of intersection between AB and AC are labeled P and Q respectively. In order to be able to manipulate geometry on both sides of the split independently, the vertices P and Q must be duplicated to create P_1, P_2, Q_1 and Q_2.

One triangle (AP_1Q_1) will be above, whereas the other two will form a convex quadrilateral below (BCP_2 and CP_2Q_2). Of the original edges, BC is retained to form the base of the quadrilateral. The other two edges are split at their point of intersection with the plane to form four smaller edges — AP_1 and BP_2 derived from AB, and AQ_1 and CQ_2 derived from AC.

Unless both polyhedra have a similar density of vertices around the intersecting area, the segmentation will generate a large number of small triangle fragments — there will be a constant net increase in polygons. As more polygons are generated, this leads to a constant slowdown of the cutting operation.

The problem of exponentially-increasing polygon count can be mitigated by mesh simplification. There are several techniques that reduce the number of polygons in a mesh whilst retaining an approximation of its original shape (Turk 1992). Simplification algorithms typically operate on an entire mesh and this adds to the processing time required (Kalvin & Taylor 1996). A suitable algorithm could be implemented that efficiently simplified only the geometry modified by the Boolean operation each frame, keeping the number of polygons in the mesh largely constant over several iterations of the cutting algorithms.

Calculating the point of intersection between an edge and a plane gives a distance along the edge, which can be normalized by dividing it by the total length of the edge. For edge pq where $p,q \in R3$, the distance from vertex p to the point of intersection with a plane in the form

$$Ax + By + Cz + D = 0$$

is calculated with the formula:

$$d = \frac{A(q_x - p_x) + B(q_y - p_y) + C(q_z - p_z) + D}{\sqrt{A^2 + B^2 + C^2}}$$

The normalized distance can then be obtained thus:

$$d_{norm} = \frac{d}{\left\| q - p \right\|}$$

This normalized distance can be used to produce a new vertex that lies on the plane. All the properties of the vertices at the opposing ends of the edge are linearly interpolated to produce the new blended vertex. As well as their positions, these properties will include all fields in the vertex format, including their normal vectors, colours, texture coordinates, tangents, binormals, etc.

Since the edge is to be divided into non-adjacent separated polygons, a copy of the interpolated vertex must be created. One of the vertices is associated with the edges and triangles on one side of the plane and the other vertex is associated with the edges and triangles on the opposite side of the plane.

3.4. Biplanar Subtraction

When cutting through bone, the path of the saw will typically follow a plane with a normal vector matching that of the saw blade. For our purposes in simulating bone surgery we shall assume that, for each cut made, the blade movement becomes limited to the plane defined by its position and normal when it first touches the bone. This implies that the saw blade cannot move along its normal vector due to matter either side of the blade, and the only axis it may rotate around is defined by its normal vector. Within these constraints, the series of Boolean subtraction operations described previously will remove a volume bounded by two surfaces lying on planes parallel to the cutting plane, separated by the width of the simulated blade, illustrated by Figure 3 and Figure 4.

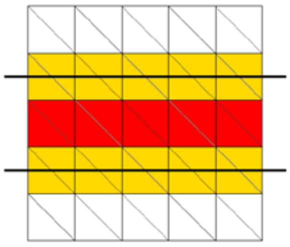


Figure 3: Before biplanar subtraction. Red triangles are removed; yellow triangles are sliced

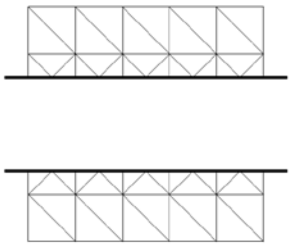


Figure 4: After biplanar subtraction

The mesh can be divided into two new meshes, with the geometry between the two planes being removed, the rest being divided into one of the new meshes, based on position relative to the cutting planes.

3.5. Capping

Once the mesh has been split along the plane, the resulting gap is then filled with non-overlapping co-planar polygons to form a cap, as shown in Figure 5 and Figure 6. A minimum number of polygons should be used to create the cap, maintaining the illusion of solidity.

For cuts with convex cross-sections, a simple method of capping the sides of the cut is to create a triangle fan. A new vertex is added at the midpoint of the cross-section, which allows the original vertices to be sorted according to their bearing from the midpoint vertex. Once the vertices have been sorted, each vertex contributes a triangle to the cap. Each vertex’s triangle is comprised of the current vertex, the vertex following it in the list, and the midpoint. To seal the cap all the way round, the last vertex wraps around to use the first vertex as its following vertex.

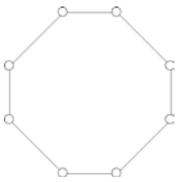


Figure 5: Cross-section vertices before capping

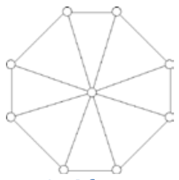


Figure 6: After capping

A more robust triangulation algorithm is the ear-clipping method (Eberly n.d.), which recursively finds 'ears' of the cross-section polygon. An ear in this context is defined as three consecutive connected vertices that form a triangle such that no other vertices lie within the triangle, and the edge joining the first and last vertices forms a diagonal of the cross-section. Each iteration of the algorithm finds an ear, adds it to the final triangulation and removes it from the initial polygon, until there is only one triangle left.

3.6. Single Plane Slicing

The caveat with using the biplanar slice method is that it appears that the user has sliced through the entire mesh. In order to overcome this, there are two possible approaches. The new geometry could be extruded to meet in the middle, and as the blade passes near to the vertices involved, they can be returned to their original locations to create the illusion of slowly separating the mesh. However, for any triangles not exactly perpendicular to the cutting plane, this will cause a visual modification to the mesh at the instant that the biplanar method is applied.

An alternative method is to perform the slice using only one plane (see Figure 7 and Figure 8). All the triangles in the mesh lying on the slice plane will have been split along the plane, and no geometry will have been removed. This planar partitioning process is used in binary space partitioning (BSP trees), where it is commonly used to divide a 3D scene graph into segments (Fuchs et al. 1980). A scene divided into a BSP tree can be used for more efficient rendering and collision detection, and BSP trees can be merged to perform CSG operations on the polyhedra they represent. In BSP trees, segments created by slicing do not have geometry added to create a visible partition - however, this is required for our purposes.

In our method, the vertices lying on the slice plane will be used to form two cross-section caps, effectively creating two barriers across the partition sliced. At the instant that a uniplanar slice is performed, there will be no visible difference in the mesh, despite it being separated into two discrete entities. However, as the blade of the saw passes near to each pair of the coexisting vertices in the sliced cross-section, they can be moved apart to create a gap in the mesh. This is consistent with the mesh being separated only as the blade of the saw passes through the geometry.

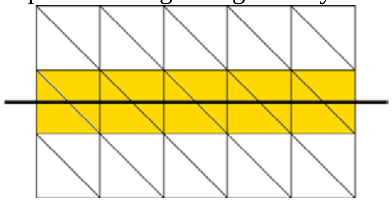


Figure 7: Before single plane sliced. Yellow triangles are sliced

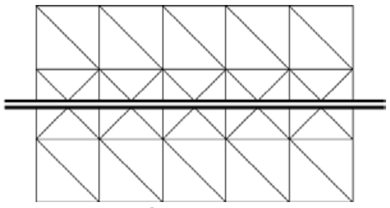


Figure 8: After single plane slice

All volume occupied by the blade throughout the previous series of iterations of the Boolean subtraction algorithm will have been removed from the initial mesh. Performing a uniplanar slice and separating the vertices as described above does not guarantee that this holds true, as the exact geometry of the blade is not removed per frame. However, since the geometry of the rendered saw blade obscures the cut whilst it is being made in the mesh, this inadequacy should not be noticeable to the user.

3.7. Localised Slicing

When cutting into the side of a toe bone, the plane on which the cut is performed will almost always intersect the other toes, so an unbounded slice along this plane would cut matter not relevant to the operation.

A localised version of the planar slice must therefore be used, by limiting the triangles on which the operation is performed.

The only relevant triangles are those whose lines of intersection with the cutting plane form a contiguous cross-section, starting with the triangle first intersected by the blade as it begins the cut.

In order to determine which polygons are relevant to a given operation, it is necessary to traverse the mesh from a given starting point, illustrated in Figure 9 and Figure 10. In our case, this will be the point at which the surgical tool first touches the mesh. The polygons forming a contiguous border around the cross-section containing the starting point can then be determined by traversing adjacent polygons that lie on the cutting plane. A starting polygon is chosen, and marked as visited and added to a stack to test.

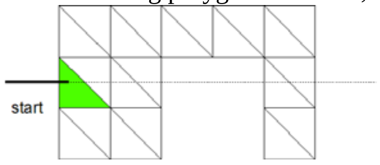


Figure 9: Initial traversal. The green triangle is marked as to be sliced

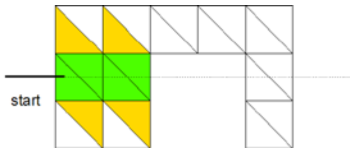


Figure 10: Final traversal. The yellow triangles are traversed to, but not sliced

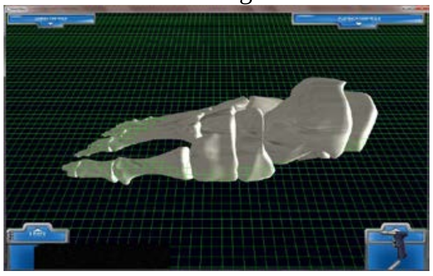
Once a list of polygons has been obtained that form a contiguous boundary around the maximum volume of the mesh to be cut, the planar slice algorithm can be applied. The effect is that the infinite plane slice is carried out on only a small part of the mesh.

3.8. Advantages of Single Plane Slicing

In terms of performance, this method requires an initial overhead of processing time (to compute the planar slice), but in subsequent frames the separation uses a minimal amount of processing time. This results in a smooth frame-rate, enabling real-time cutting to be approximated throughout the operation.

Unlike Boolean subtraction, planar slicing does not require the geometry-in-polyhedron test to determine whether the cutting mesh (in this case, the cutting plane) lies within the target mesh. All the polygons in the target mesh must instead be tested to determine which side of the plane their vertices lie on. Although this is an operation of O(*n*) complexity, it need only be performed once (at the start of a cutting operation) instead of per frame as with Boolean subtraction.

Furthermore, the number of triangles generated by this method cannot rise exponentially with the duration of the cut. The number of extra triangles added is fixed after the initial slice operation, with all subsequent frames in a given cutting operation simply moving vertices. The maximum number of triangles added is limited to 3*n* for the worst-case of all triangles in the mesh lying on the cutting plane. However, in practice, the percentage of polygons in the mesh that intersect a given plane diminishes as the total number of polygons increase. From testing, we see a maximum of around 25% of polygons intersecting the cutting plane for meshes with low numbers of polygons, and around 1% for meshes with higher totals of polygons. The system can be seen in Figure 11.



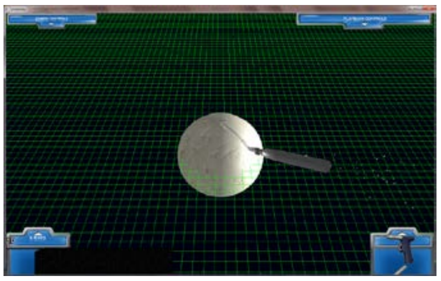


Figure 11: Showing the foot model and testing the cutting on a sphere

3.9. Disadvantages of Single Plane Slicing

With single plane slicing, the accuracy of the cut during intermediate frames is not as accurate as that possible when using Boolean subtraction. When a triangle fan is used to cap the two cross-sections, moving the vertices apart individually has the visual appearance of ‘folding’ the caps apart. However, the inaccuracy is typically only visible between the two cross-sections of the slice, which is difficult to see from a viewpoint outside of the mesh, and the geometry of the saw will tend to obscure these details. Once the cut is complete, these inaccuracies are no longer visible. A likely cause of visual inaccuracies is when vertices are moved a given distance from the plane, they share edges with other vertices whose distance to the plane is less than the movement distance. This will cause edges that previously pointed towards the plane to now point away from it (or vice-versa), which in the best case will introduce new concavities into the mesh, and in the worst case will cause the extrusion away from the plane to crease back over other geometry, creating a visible fold around the edges of the slice. One way of mitigating this is to move all vertices within the movement distance from the cutting plane, instead of just the new vertices created along it. This will collapse all nearby vertices and edges onto a plane at the given distance from the cutting plane, eliminating any overlap.

CONCLUSION

Of the conventional geometry cutting methods described, we believe the most suitable method to simulate bone cutting is Boolean subtraction operating on polygonal meshes. However, although Boolean subtraction provides an accurate cut, it is not suitable for use in real-time cutting.

Therefore, using assumptions of how the cutting will be used to perform surgery, we proposed a simplification of the Boolean process that obtains the same result by slicing mesh geometry with two planes. A refinement of this planar slice technique using one plane allows the mesh to be cut without undergoing immediate visual modification. After the initial cut using this technique, the two sides of the mesh can be separated over the course of many frames simply by moving vertices, which is an operation fast enough to run at real-time interactive rates.

Slicing along an infinite plane works well on convex meshes, but with concave meshes (especially those with multiple parallel protrusions, such as the bones of a foot) slicing this way will cause distant, unrelated parts of a mesh to be sliced if they happen to intersect the cutting plane. To overcome this limitation, we propose traversing the mesh by triangles that intersect the cutting plane to determine the set of relevant triangles. Furthermore, in order to detect if part of the mesh has been severed from the rest, a similar traversal algorithm can be used to find all contiguous groups of triangles.

REFERENCES

Aftosmis, M., Berger, M. & Melton, J., 1998. Robust and efficient Cartesian mesh generation for component-based geometry. *AIAA Journal*, 36(6), pp.952–960.

Agus, M. et al., 2002. A multiprocessor decoupled system for the simulation of temporal bone surgery. *Computing and Visualization in Science*, 5(1), pp.35–43.

Agus, M. et al., 2003. Real-time haptic and visual simulation of bone dissection. *Presence: Teleoperators and Virtual Environments*, 12(1), pp.110–122.

Allan, J.B., Wyvill, B. & Witten, I.H., 1988. A methodology for direct manipulation of polygon meshes. In *CG International*. CG International. University of Calgary.

Bill, J., 1994. *Computer Sculpting of Polygonal Models using Virtual Tools*, University of California.

Blinn, J.F., 1982. A Generalization of Algebraic Surface Drawing. *ACM Trans. Graph.*, 1(3), pp.235–256.

Choi, K. et al., 2002. A scalable force propagation approach for web-based deformable simulation of soft tissues. In *Proceedings of the seventh international conference on 3D Web technology*. Proceedings of the seventh international conference on 3D Web technology. ACM Press, pp. 185–193.

De Luca, D. & Taylor, R., 2012. A case study of the development of a 3D virtual object handler and digital interactives for museums by Canalside Studios (University of Huddersfield).

Delingette, H., 1998. Towards realistic soft tissue modeling in medical simulation. *Proceedings of the IEEE*, 86(3), pp.512–523.

Delp, S.L. et al., 1997. Surgical Simulation: An Emerging Technology for Training in Emergency Medicine. *Presence: Teleoperators and Virtual Environments*, 6(2), pp.147–159.

Eberly, D. ed. *Triangulation by Ear Clipping* D. Eberly, ed. Geometric Tools LLC. Available at: http://www.geometrictools.com [Accessed March 26, 2012].

Fuchs, H., Kedem, Z. & Naylor, B., 1980. On visible surface generation by a priori tree structures. *ACM SIGGRAPH Computer Graphics*, 14(3), pp.124–133.

Haluck, R.S. et al., 2001. Are surgery training programs ready for virtual reality? a survey of program directors in general surgery. *Journal of the American College of Surgeons*, 193(6), pp.660–665.

Kalvin, A.D. & Taylor, R.H., 1996. Superfaces: polygonal mesh simplification with bounded error. *Computer Graphics and Applications, IEEE*, 16(3), pp.64–77.

Lane, J. et al., 1980. Scan line methods for displaying parametrically defined surfaces. *Commun. ACM*, 23(1), pp.23–34.

LeBlanc, A. et al., 1991. Sculpting with the `ball and mouse' metaphor. In *Proceedings. Graphics Interface 91*. Proceedings. Graphics Interface 91.

McEachern, M., 2008. Force to be Reckoned With. *Computer Graphics World*, 31(9), pp.20–26.

Morris, D. et al., 2004. A Collaborative Virtual Environment for the Simulation of Temporal Bone Surgery. *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 1, pp.319–327.

Morris, D. et al., 2006. Visuohaptic Simulation of Bone Surgery for Training and Evaluation. *Computer Graphics and Applications, IEEE*, 26(6), pp.48–57.

Occena, L. & Tanchoco, J., 1988. Computer graphics simulation of hardwood log sawing. *Forest Products Journal*, 38(10), pp.72–76.

Parent, R.E., 1977. A system for sculpting 3D data. In *SIGGRAPH*. SIGGRAPH. pp. 138–147.

Petersik, A. et al., 2002. Realistic Haptic Volume Interaction for Petrous Bone Surgery Simulation. In *Computer Assisted Radiology and Surgery*. Computer Assisted Radiology and Surgery. pp. 252–257.

Pflesser, B. et al., 2000. Volume Based Planning and Rehearsal of Surgical Interventions. In *Computer Assisted Radiology and Surgery, Proc. CARS 2000, Excerpta Medica International Congress, 1214*. Computer Assisted Radiology and Surgery, Proc. CARS 2000, Excerpta Medica International Congress, 1214. Elsevier, pp. 607–612.

Requicha, A., 1980. Representations of rigid solid objects J. Encarnacao, ed. *ACM Computing Surveys*, 12(4), pp.437–464.

Requicha, A.A.G., Tilove, R.B.University of Rochester. Production Automation Project, 1978. Mathematical foundations of constructive solid geometry: general topology of closed regular sets.

Szirmay-Kalos, L. & Márton, G., 1998. Worst-case versus average case complexity of ray-shooting. *Computing*, 61, pp.103–131.

Turk, G., 1992. Re-tiling polygonal surfaces. *ACM SIGGRAPH Computer Graphics*, 26(2), pp.55–64.

Wamae, W. et al., *Translational Research and Knowledge in agriculture and food production*, RAND.