Engineering Analysis with Boundary Elements 58 (2015) 1–6

Contents lists available at ScienceDirect

Engineering Analysis with Boundary Elements

journal homepage: www.elsevier.com/locate/enganabound

Efficient visibility criterion for discontinuities discretised by triangular surface meshes

Nicholas Holgate, Grand Roman Joldes*, Karol Miller

Intelligent Systems for Medicine Laboratory, School of Mechanical and Chemical Engineering, The University of Western Australia, 35 Stirling Highway, Crawley/Perth, WA 6009, Australia

ARTICLE INFO

Article history:
Received 5 September 2014
Received in revised form
8 January 2015
Accepted 25 February 2015
Available online 20 March 2015

Keywords:
Meshless methods
Discontinuities
Surgical simulation
Biomechanics

ABSTRACT

This study proposes a computationally efficient algorithm for determining which pairs of points among many predetermined pairs in three dimensions will maintain straight line visibility between one another in the presence of an arbitrary surface mesh of triangles. This is carried out in the context of meshless numerical methods with the goal of implementing near-real-time discontinuity propagation simulation. A brief overview is given of existing discontinuity modelling techniques for meshless methods. Such techniques necessitate determination of which key pairs of points (nodes and quadrature points) lack straight line visibility due to the discontinuity, which is proposed to be modelled with a surface mesh of triangles. The efficiency of this algorithm is achieved by allocating all quadrature points and surface mesh triangles to the cells of an overlayed three-dimensional grid in order to rapidly identify for each triangle an approximately minimal set of quadrature points whose nodal connectivities may be interrupted due to the presence of the triangle, hence eliminating most redundant visibility checking computations. Triangles are automatically split such that any size of overlayed cubic grid cells can be employed, and the parameters governing triangle splitting and binning have been examined experimentally in order to optimise the visibility algorithm.

Crown Copyright © 2015 Published by Elsevier Ltd. All rights reserved.

1. Introduction

This work seeks to make an addition to meshless numerical methods as applied to emerging fields such as computational biomechanics, in which one of the present challenges is near-real-time simulation of soft tissue cutting. In applications of biomechanical simulation, meshless methods have many practical advantages over FEM. Presently, the most important is that accurate model generation of patient-specific organ geometry from pre-operational images can be automated, while FEM models comprised of workable elements require days of adjustment by an analyst [1,2]. Removal of this workflow bottleneck makes meshless methods an ideal candidate for implementation in near-real-time intra-operational surgical simulations of tissue deformation, which for many procedures demands simulation of cutting.

Attempts to model discontinuities within meshless simulations have mostly been focused on cracks and their propagation. Rabczuk et al. [3,4] use nodes in the crack path to superimpose a discontinuous enrichment function to the nearby displacement field. Use of these so-called “cracked particles” applies only to finely cracking solids and not to deforming soft bodies with arbitrary discontinuities, and the existence of discontinuities only at particular particles limits the accuracy with which they can be modelled. Level-set functions proposed by Osher and Sethian [5] and applied to FEM crack growth modelling by Stolarska et al. [6], have also been applied to the problem of using meshless methods to model surgical cutting of brain tissue in two and three dimensions by Jin et al. [7]. In two dimensions, discontinuities are represented by a series of straight line segments, each of which uses the vector between the segment's beginning and end to define a level-set function with values of opposing signs on opposing sides of the segment, and another such vector and level-set function perpendicular to the end of the segment. This allows a natural division of the space into four subdomains. By calculating the two function values for each of any two points in space, it can be immediately determined whether the line segment under consideration will block straight line visibility between the points, allowing appropriate adjustment of support domains. This idea can be extended to three dimensions [8], and can also make use of level-set functions whose zeroes are not straight line segments or planes, allowing more complex discontinuities without additional segments, provided that an appropriate closed-form level-set function can be found. A potential drawback to this method is that it requires intricate piecewise function definitions to define jagged or curved shapes, which may entail speed and accuracy reductions. It is also difficult to update the level-sets for intersecting discontinuities or sharp changes in the direction of the cut.

Krysl and Belytschko [3,9] have previously proposed and implemented meshes of triangular elements for modelling of arbitrary crack growth in conjunction with the visibility criterion.

* Corresponding author.
E-mail address: grand.joldes@uwa.edu.au (G.R. Joldes).

http://dx.doi.org/10.1016/j.enganabound.2015.02.014
0955-7997/Crown Copyright © 2015 Published by Elsevier Ltd. All rights reserved.

At a time step in which crack growth occurs, only the new portion of the discontinuity needs to be considered to alter the necessary support domains. For each new triangle in the propagating crack, its bounding box, inflated by the simulation's largest support size, is used as a maximum region in which shape functions could possibly need to be altered due to that discrete portion of the discontinuity. When a set of new triangles is added to extend the crack, for each quadrature point enclosed within the union of all the maximum regions, the rays between it and each associated node are checked for intersection with the new triangles. If the ray intersects the triangle, the “visibility criterion” between the two points of interest fails, and the node is removed from the quadrature point's list of neighbours which influence the local shape functions.

An efficient algorithm for finding which quadrature points are contained within a given maximum region is not proposed in their paper. For a particular crack growth time step there may be many such regions requiring identification of contained quadrature points, and it may be too slow to retrieve the appropriate points from lists of global quadrature point coordinates to permit real-time growth simulation. In furthering the strategy put forward by their paper, it is worth noting that it is not necessary to check all the quadrature points found in the union of the maximum regions against every contained triangle, but rather just the ones inside each individual inflated bounding box against its associated triangle. Additionally, an inflated bounding box will always include unnecessary quadrature points near the corners, which can be mostly eliminated by instead using an appropriately inflated bounding sphere.

In many applications, surface meshes of triangles may be the most desirable method for representing either static or propagating discontinuities. An analyst may easily manually place the vertices of a set of triangles interconnected such that they closely replicate a real discontinuity. Accurate automation of surface mesh creation from an existing three dimensional image featuring a clear discontinuity is also a simple task compared to that of automatically meshing a volume. For propagating discontinuities, addition of triangles to the outer edges of the surface is a natural operation which does not require adjustment to the existing discontinuity.

In Sections 2 and 3, this paper outlines an algorithm for efficiently conducting the visibility checks required to model meshless methods discontinuities with a surface mesh of triangles. Section 4 presents experimental findings pertaining to execution times which justify its use in near-real-time applications such as surgical simulation, even for meshes composed of very many triangles.

2. Algorithm overview

Quadrature points which are more distant than the simulation's largest nodal support size from all points on a triangular discontinuity portion cannot have visibility blocked from the nodes that influence their local shape functions, and so they do not need to have their neighbouring nodes checked for straight line visibility. By cycling through every triangle comprising the surface mesh, and identifying as small as possible a set of quadrature points which contains all of those sufficiently nearby to warrant nodal visibility checks, the number of visibility criterion checks required to appropriately adjust the nodal support domains due to the discontinuity will be minimised.

As a simple and computationally efficient way of enclosing all of the quadrature points within the maximum support size distance from all points of a triangle, bounding spheres are proposed for each triangle. The surface of each sphere is such that all points are at least the maximum support size away from all points on the triangle which it surrounds. It is not viable to find which of the simulation quadrature points lie within the sphere by checking all of their Euclidian distances from its centre. Rather, inspiration is drawn from the field of computational contact mechanics, in which the “bucket search” algorithm (proposed by Benson and Hallquist [10]) is commonly used to efficiently detect the occurrence of contact between two disjoint bodies in a simulation. The adaptation of this algorithm as applied to the present problem begins with superposition of a three-dimensional grid of cubic cells over the physical coordinates. Each surface mesh triangle and each quadrature point is allocated (or “binned”) into a unique cell, with triangles being split until they each fit into a unique cell within certain overhang tolerances. With each triangle approximately confined to a single cubic cell, the bounding sphere can be approximated by a “sphere of cubes” consisting of a set of grid cells, centred upon the cell containing the triangle. This scenario is demonstrated in Fig. 1.

The set of cells required to fully include a bounding sphere of a particular radius given a particular cell length can be precomputed, such that all the quadrature points already allocated to that

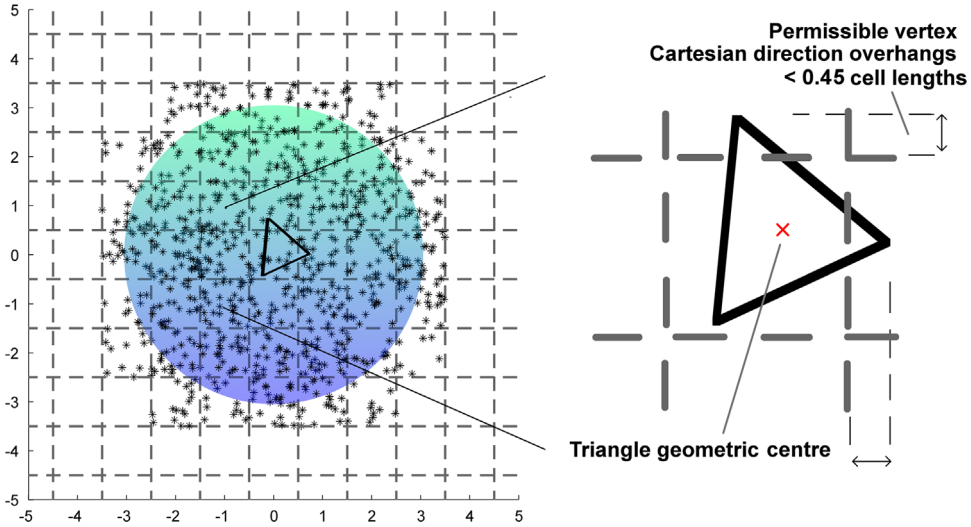

Fig. 1. A sphere of cubes viewed along the cell grid *x*-axis – with visible overestimations, the sphere of cubes includes the entire idealised bounding sphere of the triangle of concern, which in turn includes all quadrature points whose nodes may fail the visibility criterion due to the triangle. The idealised sphere is inflated slightly to allow for the 0.45 cell length maximum possible overhang of the triangle vertices into cells adjacent to the triangle's allocation cell.

set of cells can be rapidly identified. For the sphere of cubes to fully contain the sphere in which quadrature points must be checked, its outermost cubes will inevitably contain a fractionally small amount of superfluous quadrature points. The complete algorithm is summarised below.

1. Load the quadrature point locations, node locations, and lists of nodes local to each quadrature point.
2. Superimpose cubic grid cells of a fixed side length over the existing physical coordinates.
3. Allocate each quadrature point to the grid cell in which it lies, forming cubic clusters of points.
4. Split all surface mesh triangles to sizes small enough to allow allocation to unique grid cells, according to a criterion allowing a certain amount of triangle overhang, which is detailed below.
5. Loop through every triangle in the resulting surface mesh.

5.1 Find the smallest sphere of cubes which contains all quadrature points close enough to the current triangle that some of their nodes could be blocked from straight line visibility. This process is detailed below.

5.2 Loop through all of the quadrature points located within the sphere of cubes.

5.2.1 Loop through all nodes associated with the current quadrature point. Perform a straight line visibility check between the current quadrature point and current node in the presence of the current triangle. If visibility is blocked, remove the node from the list of nodes affecting that quadrature point.

The overhang is quantified for each triangle vertex as the maximum distance in a Cartesian direction that it extends beyond the edges of the cell in which its triangle's geometric centre lies. The time-optimal permissible overhang depends upon the particular splitting algorithm used, and for the algorithm proposed in Section 3, the value is experimentally found to be approximately 0.45 cell lengths. A triangle is allocated to the cell containing its geometric centre only when all of its vertices are within this overhang, or else it is split further.

To find the optimal sphere of cubes for a particular triangle, the corresponding bounding sphere is first identified. Its centre is that of the cell to which the triangle has been allocated, and its radius expressed in cell lengths is the maximum support size in cell lengths, plus 0.5 (because the triangle does not exist exactly at the centre of its allocated cell), plus the maximum overhang in cell lengths (because some of the triangle's vertices may extend outside of the cell by up to this amount). Since detection of which cells lie fully or partially in the original sphere is slow for large radii, a precomputed list has been generated, which maps spheres of various radii to the corresponding sets of cubes. Necessary cubes for radius increments of 0.1 cell lengths were precomputed, so that the radius of the sphere under consideration must be rounded up only very slightly, minimising the possible associated overestimations. The appropriate cubes for each radius are simply those which have at least one vertex, face centre or edge centre within the sphere, as this detects all cubic cells which are entirely or partially within the sphere. The time performance of the algorithm depends strongly upon the size of the cubic cells relative to the mesh triangles, since this controls the amount of necessary triangle splitting and the number of superfluous quadrature points captured by the outer layer of the sphere of cubes. An analysis of the optimal cell size due to these two competing effects is performed in Section 4.

3. Splitting triangles

Since most triangles representing a given discontinuity surface mesh will generally extend across multiple binning cells, it is necessary to split them into smaller triangles of approximately the same size as the cells. The algorithm for splitting mesh triangles should rapidly produce low aspect ratio triangles and maintain consistency of the surface mesh. A consistent surface mesh is here defined as one in which all triangle edges connected to neighbouring triangles are spanned exactly by the neighbouring triangle's edge, rather than being divided. That is, no triangle's vertex can divide the edge of a neighbouring triangle. This preserves the ability to define the mesh as an index array used to retrieve triplets from a set of vertex coordinates, as well as the ability to treat the surface as a finite element mesh, which may be desirable should analysts wish to track the motion of the discontinuity. Below is a worded description of the proposed splitting algorithm, which operates on triangles in a mesh on an individual basis. This is accompanied by a visual example of the steps in Fig. 2. As an example of the splitting algorithm's application in preserving mesh consistency, Fig. 3 presents a simple surface mesh of triangles which undergoes subdivision.

1. Divide the three sides of the initial triangle into equally sized segments approximately equal to a given goal length.
2. Starting at the vertex common to the longest and second longest sides of the original triangle and progressing towards the shortest side, add consecutive adjacent triangles reaching between the two longest sides until the segments of the second longest side have all been used.
3. \(Conditional) If any segments connecting the longest and second longest sides of the original triangle are longer than the goal length, two consecutive triangles generated in Step 2 are merged into a quadrilateral which is then split lengthwise into a strip of low aspect ratio triangles by using the goal length to place new triangle vertices along the long quadrilateral sides.
4. (Conditional) If the longest and second longest sides of the original triangle were not divided into the same number of segments, the final remaining triangular subsection of the original triangle is filled by calling the algorithm again. The recursions of the algorithm end when a triangle is reached whose sides are all sufficiently small compared to the goal length that they escape division, so that this last triangle is not split.

The key to maintaining a consistent mesh with a splitting scheme that operates only on unique triangles is the identical splitting of shared edges. The segment goal length is a convenient way to achieve this given the need to allocate triangles to unique grid cells of known side lengths. For initial mesh triangles of low aspect ratio which encounter Step 4, triangles with very high aspect ratio can occasionally be produced using this method. There are many possible remedies for applications requiring maintenance of only low aspect ratio triangles in the mesh. For example, triangles in the potential trouble region can be checked against a chosen upper threshold aspect ratio and if necessary can be further split by an alternative, subordinate splitting scheme selected to reduce the local aspect ratios while still maintaining mesh consistency. If further splitting alone is insufficient, the skinny triangles could be merged with adjacent ones and then split to maintain mesh consistency.

In practice, even for goal lengths shorter than the cell length, the final surface mesh may still contain triangles which do not satisfy the cell allocation criterion. We have found experimentally that maximum allocation speed is obtained when the maximum permissible overhang of each triangle vertex from a cell in any Cartesian direction is roughly 45% of the cell length. If there are triangles in the split mesh which do not meet this criterion, a local split of these triangles and their neighbours can be performed.

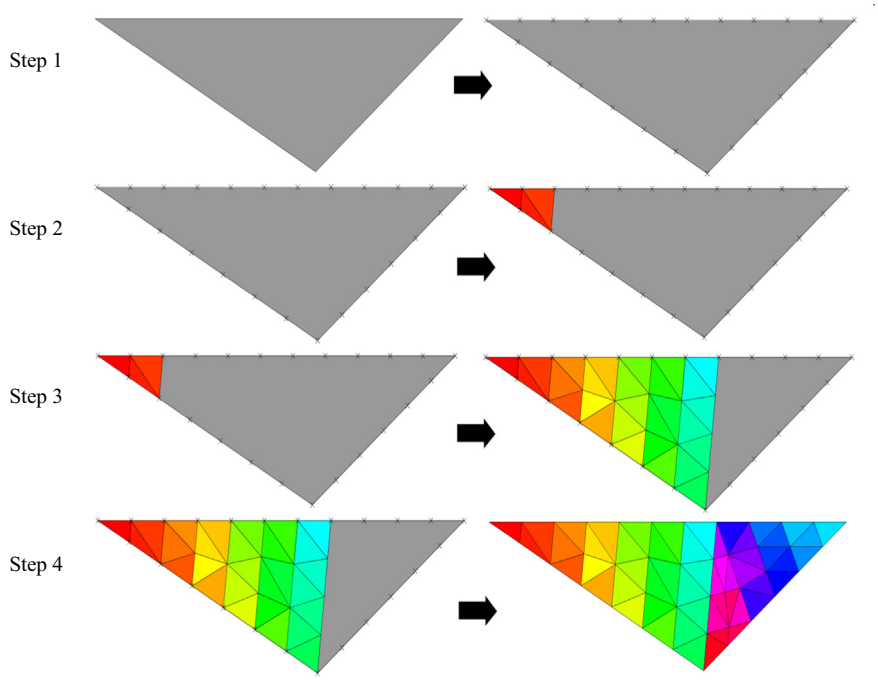

Fig. 2. A step-wise example of splitting a single triangle with the proposed algorithm.

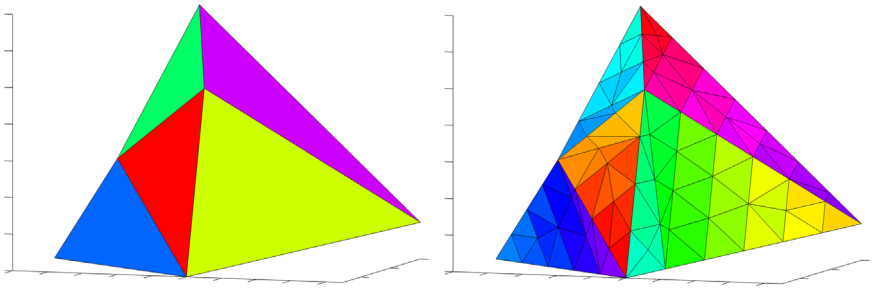

Fig. 3. The splitting algorithm maintaining consistency of a mesh of triangles.

4. Results

In addition to the maximum overhang for triangle grid cell allocation, the execution time of the algorithm depends upon the following parameters:

1. The size of the grid cells relative to the sizes of the initial mesh triangles.
2. The number of triangles in the initial mesh.
3. The quadrature point density near the discontinuity.
4. The maximum support size.
5. The number of nearest neighbour nodes associated with each quadrature point.

The former two parameters are experimentally optimised while the quadrature point density is roughly homogeneous and the latter two parameters are kept constant, with values appropriate to a practical simulation. The size of grid cells is normalised by the mean area of the initial triangles in the mesh, and termed the length-to-area ratio or LAR. For each of a set of test meshes with a broad range of initial triangle counts, the time to complete the algorithm is recorded for different cell lengths. Each mesh has a LAR value which minimises the algorithm time and it is found that this value depends upon the mesh's initial triangle count according to a power law relationship (Fig. 4). This empirically determined relationship has a strong fit and allows the ideal cell size to be predicted based upon the number of triangles in the initial mesh before the algorithm begins. The meshes are all identical in shape, size and position to the one shown in Fig. 3. To generate meshes with more triangular elements, the mesh shown is pre-split with consecutively shorter goal lengths. The problem geometry used for tests consists of 8000 quadrature points and 4000 nodes randomly distributed in a cube of side length two. Each quadrature point has its nearest eight nodes listed as those originally influencing the shape functions at that point. This results in a roughly constant density of quadrature points and a maximum support size of 0.2864 units. The mesh is small enough and positioned near enough to the centre of the cubic problem geometry that no spheres of cubes extend outside of it. All of the spheres of cubes therefore contain a similar quantity of quadrature points.

There is an explosion in algorithm execution time when LAR is lowered below optimal because a great deal of mesh triangles will be produced, through which the algorithm must loop. Above optimal, there is only a gentle overall increase in execution time with slight oscillations – the overall trend is due to the inclusion of more superfluous quadrature points at the sphere surface as the cell size grows, while the oscillations are due to the changes in triangle allocations throughout the mesh. Therefore, an interpolative function which slightly overestimates the observed data is

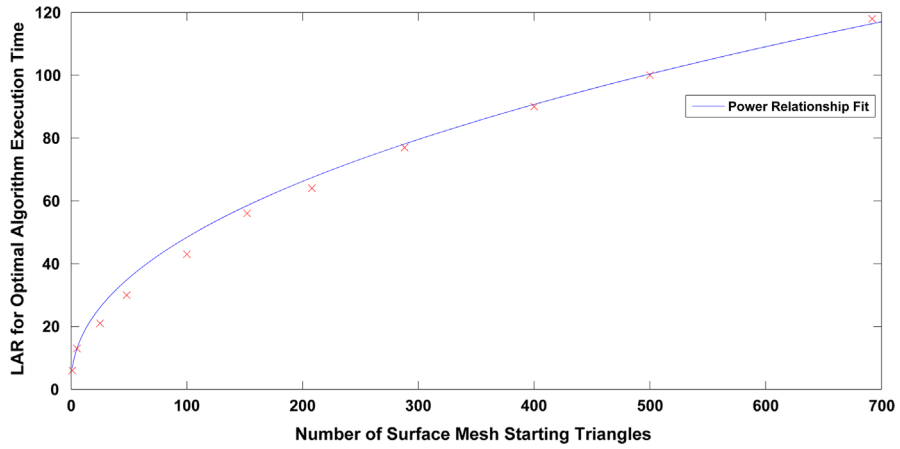

Fig. 4. The size of grid cells that leads to minimum execution time can be found for a chosen distribution of nodes and integration points based on the number of triangles in the mesh and their mean area.

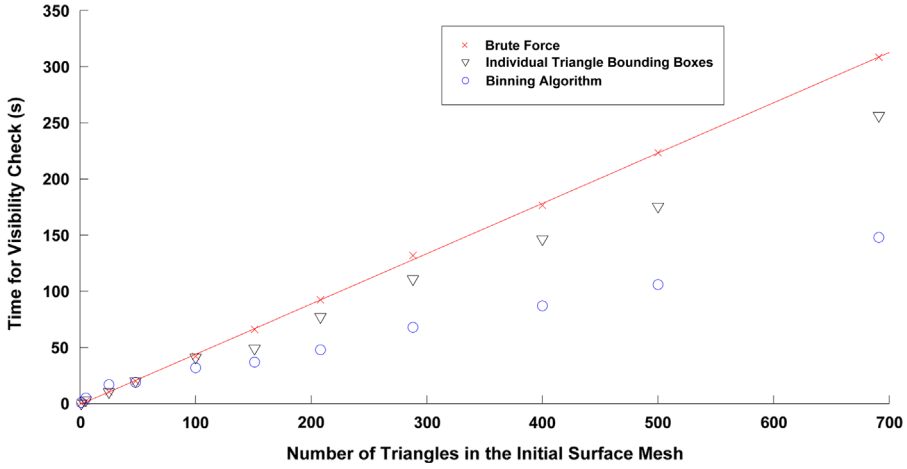

Fig. 5. Execution times for the binning algorithm and other visibility checking methods for various initial surface mesh triangle counts.

desirable, since a slightly overestimated LAR can only subtly increase the execution time from optimal, while a slightly underestimated LAR may reside below the explosion threshold and risk a sudden large increase in execution time. The algorithm time always increases with initial mesh triangle count, and so the optimal value for the initial mesh triangle count is the minimum number of triangles that adequately represent the discontinuity under consideration.

To demonstrate the efficacy of the algorithm, its times to complete all necessary visibility checks are compared to those of both a brute force method and the more refined method of Krysl and Belytschko [3,9]. The brute force method identifies candidate quadrature points by taking the bounding box of the entire surface mesh under analysis and inflating it by the maximum support size. Many discontinuities which do not fill this region well will have visibility checks performed with a large number of unnecessary quadrature points far from the mesh. Identifying quadrature points in inflated bounding boxes around each individual triangle, as suggested by Krysl and Belytschko, does reduce superfluous visibility checks. However, for large triangle counts, the process of identifying these candidate points around every triangle is slow without making use of a binning system, at which stage the method essentially becomes the sphere of cubes algorithm, but it would possess the relative hindrance of featuring unnecessary candidates near the edges and corners of each box.

The discontinuity used for this demonstration is again that shown in Fig. 3, which has a roughly cubic inflated bounding box which is well spatially filled by its mesh. As such, this represents a best-case scenario for the brute force method, since the number of superfluous quadrature points within the bounding box is minimal. The number of initial triangles is increased by pre-splitting the mesh, so that its shape and bounding box are not changed. In this scenario, the algorithm performs visibility checks in only about half the time of the brute force method, as shown in Fig. 5, while the method of Krysl and Belytschko exhibits intermediate performance.

When the experiment is repeated for meshes which are such that they do not spatially fill their bounding box, the factor of time reduction between the brute force and other methods becomes much higher, while the relative advantage of the binning algorithm over the method of Krysl and Belytschko is unaffected by the shape of the discontinuity. The implementations of the present algorithm and the other methods have been done using MATLAB. The results shown are indicative only of the relative advantage of employing the algorithm over the other approaches; the absolute times would be considerably less for a streamlined and parallelised implementation.

Both the mesh splitting algorithm and the visibility check are well suited for parallel implementation on a Graphics Processing Unit (GPU). The splitting algorithm handles each triangle defining the cut independently, and similarly, the visibility criterion is checked for each triangle in the split mesh independently. The computations can therefore be performed in parallel by a number of threads equal to the number of triangles that need to be handled,

similar to the algorithms performing finite element computations presented in [11].

5. Conclusion

This paper describes an efficient algorithm for visibility checking when discontinuities are represented by triangular surface meshes. The main advantage of the present algorithm is that its speed is independent of the shape of the discontinuity, since it treats each triangle in the mesh individually and locally. The second advantage is that it can handle meshes of many triangles with minimal increase in execution time because binning avoids the need to conduct a search through the global list of quadrature points for every triangle. A dedicated implementation of the algorithm would help to facilitate near-real-time simulation of cutting in which the discontinuities were flexibly and robustly modelled as surface meshes of triangular elements, even for meshes which are highly accurate models of their represented discontinuities and which therefore consist of very many triangles.

Acknowledgements

The financial support of the Australian Research Council (Discovery Grants nos. DP1092893, DP120100402) and the National Health and Medical Research Council (Grant no. APP1063986) is gratefully acknowledged.

References

[1] Wittek A, Joldes G, Couton M, Warfield SK, Miller K. Patient-specific non-linear finite element modelling for predicting soft organ deformation in real-time: application to non-rigid neuroimage registration. Prog Biophys Mol Biol 2010;103(2–3):292–303.
[2] Miller K, Horton A, Joldes GR, Wittek A. Beyond finite elements: a comprehensive, patient-specific neurosurgical simulation utilizing a meshless method. J Biomech 2012;45(15):2698–701.
[3] Rabczuk T, Belytschko T. Cracking particles: a simplified meshfree method for arbitrary evolving cracks. Int J Numer Methods Eng 2004;61(13):2316–43.
[4] Rabczuk T, Song J, Belytschko T. Simulations of instability in dynamic fracture by the cracking particles method. Eng Fract Mech 2009;76:730–41.
[5] Osher S, Sethian JA. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton–Jacobi formulations. J Comput Phys 1988;79:12–49.
[6] Stolarska M, Chopp DL, Moës N, Belytschko T. Modelling crack growth by level sets in the extended finite element method. Int J Numer Methods Eng 2001;51:943–60.
[7] Jin X, Joldes GR, Miller K, Yang KH, Wittek A. Meshless algorithm for soft tissue cutting in surgical simulation. Comput Methods Biomech Biomed Eng 2012;17(7):800–11.
[8] Jin X, Joldes GR, Miller K, Wittek KA. 3D Algorithm for Simulation of Soft Tissue Cutting. Computational Biomechanics for Medicine: Models, Algorithms and Implementation. New York: Springer; 2013. p. 49–62.
[9] Krysl P, Belytschko T. The element free Galerkin method for dynamic propagation of arbitrary 3-D cracks. Int J Numer Methods Eng 1999;44:767–800.
[10] Benson D, Hallquist J. A single surface contact algorithm for the post-buckling analysis of shell structures. Comput Methods Appl Mech Eng 1990;78:141–63.
[11] Joldes GR, Wittek A, Miller K. Real-time nonlinear finite element computations on GPU – application to neurosurgical simulation. Comput Methods Appl Mech Eng 2010;199(49–52):3305–14.