Computer-Aided Design and Applications

http://www.cadanda.com

Novel Collision Detection Algorithm for Physics-based Simulation of Deformable B-spline Shapes

Harish Pungotra[1], George K. Knopf[2] and Roberto Canas[3]

[1]The University of Western Ontario, hpungotr@uwo.ca
[2]The University of Western Ontario, gknopf@eng.uwo.ca
[3]National Research Council, roberto.canas@nrc-cnrc.gc.ca

ABSTRACT

A structured computational framework to efficiently detect collision between deformable B-spline shapes and calculate reactive forces and deformation based on mass spring system in a virtual reality environment is described in this paper. The proposed technique utilizes the fact that the transformation matrices used to discretize the B-spline surface are independent of the position of the control points and, therefore, can be pre-calculated. The density of the points is dynamically increased at lower levels of detail. Finally, the regions which are likely to undergo collision are tessellated using these points. The algorithm is compatible with a mass spring system to calculate forces and deformation of the model, based upon its physical material properties. The algorithm is ideally suited for virtual sculpting and validating conceptual design models before extensive product detailing is performed using readily available commercial CAD software. A comparison based on the worst case scenario is presented and used to prove the computational efficiency of the proposed algorithm.

Keywords: virtual reality, collision detection, deformable object, B-spline surface.
DOI: 10.3722/cadaps.2009.43-54

1. INTRODUCTION

At the early stages of product design, the specifications and constraints have not been fully established. The industrial designers and engineers need the freedom to change and modify the product configuration and mechanical behavior to investigate a wide range of alternative solutions. Virtual reality (VR) technology can provide the environment and freedom needed to enhance the creativity of a designer. The need for a VR-based concept design tool has been supported by several studies [6, 28]. Any CAD system that seeks to support and enhance conceptual design must, therefore, enable natural and haptic modes of human-computer interaction.

A fast, efficient and robust collision detection algorithm is essential for realistic simulation of deformable bodies. The algorithm should be capable of handling complex surface models, a large area of contact, multiple contacts and high deformation. It should also work in tandem with a physics based system so as to realistically calculate reactive forces and resultant deformation of the model. The algorithm should also allow the user to use more than one model and a variety of rigid and deformable tools in the virtual reality environment. A broad selection of tools would augment the creativity of the industrial designer or artist working in a virtual reality environment. The hand can also be represented as a deformable B-spline tool and used for sculpting.

Tessellated surfaces have been widely used in geometric modeling because of the inherent simplicity of triangle-triangle collision detection and hardware compatibility for rendering. However the resolution of the surface cannot be dynamically changed during haptic interaction. Furthermore, as the model deforms during haptic interaction, the '*quality*' of the triangles deteriorates. This makes collision detection, computationally extensive and inefficient.
Recently, B-spline representation has become the standard for many CAD/CAM applications, because complex surfaces can be represented with minimal information storage. Thus it is imperative that any concept design module uses a B-spline surface virtual model to streamline the exchange of information with existing CAD/CAM systems. However it is computationally costly to continuously find the new equation of a B-spline surface while the model is deforming during haptic interaction. Thus the majority of the algorithms find the contact information of only a point-based tool with a parametric/B-spline/NURBS surface [7, 24, 27]. These algorithms cannot be efficiently used for implicit surface- or parametric surface-based tools. Gao [10] has presented an algorithm which can use implicit surface tools but this algorithm cannot be used for a point- or B-spline surface-based tool. As a consequence, a haptic based concept design module cannot use these or similar algorithms without restricting the creativity of the user.
Shape modification of a virtual object can be simulated using either geometric- or physics-based algorithms. Geometric techniques typically use Hooke's law and haptic tool penetration to determine the reaction force and adjust the vertices of the underlying mesh model in response to external forces [3]. Unlike geometric techniques, computationally intensive physics-based techniques can yield real material behavior of a multiple-material/non-homogeneous virtual model [18]. Physics-based deformation models give the designer, more opportunities to try different types of materials during the concept design phase and validate product models in real-time. The algorithm presented by Pungotra [26] can be further extended for physics-based simulation of B-spline surface-based model and tool. The transformation matrices used to discretize the B-spline surface can be used for generation of nodes for the mass spring system.
This paper describes a fast algorithm to detect collision between two or more single patch B-spline surfaces having a complex surface, a large area of contact, multiple contacts, and model deformation based on mass spring system. The proposed method takes advantage of the parametric representation of the surface and efficiency of a triangle-triangle intersection test. No assumption regarding the complexity, degree, and number of control points of the B-spline surface representing the model or the tool has been considered. Both the model and the tool can have elastic or plastic properties. The nodes of mass spring system are independent of the discrete points generated for collision detection. As a special case, the tool can also be represented as a point- or implicit surface-based tool. The paper presents both a quantitative and qualitative comparison to prove the computational efficiency of the proposed algorithm.

2. PREVIOUS WORK

The problem of collision detection has been extensively studied in literature [16, 20]. Most of the collision detection algorithms are based on various bounding geometries and spatial decomposition techniques to speed up the response of the system by performing a '*rejection test*'. Spheres as bounding volume have been used at different levels of detail [23]. Gregory [12] used a pre-computed hybrid hierarchical representation consisting of uniform grids and trees of tight-fitting *Oriented Bounding Box* (OBB) Trees. Ehmann [8] presented a unified approach to perform a set of proximity queries for rigid polyhedral objects. Krishnan[19] used shell tree to speed up the collision detection process. Klosowski [17] introduced k-dop as bounding volume for objects moving within highly complex environments. The bounding geometry techniques work very well with rigid bodies but when applied to deformable bodies, the computational cost of updating these geometries as the object deforms slows down the collision detection response. Two-level layered model representation techniques have also been investigated to develop a compromise between accuracy and the need to improve the robustness of the collision detection algorithm [9, 21].
Patoglu [24] presented an algorithm that can determine the closest point on a convex parametric surface patch to a given point. Dachille [7] used polyhedral representation which makes it easier to search for the nearest point on the surface, unlike the complicated NURBS surface intersection task proposed by Thompson [27]. In all these techniques the interaction with the virtual model is only at a point through a point-based tool. This limits the utility of the approaches, particularly where the

sculpting is required to be done by hands or other surface-based tools. Gao [10] uses a B-spline to represent the surface of a virtual model and implicit surface (up to degree of two) to represent a virtual tool. The collision detection procedure is done by inputting discretized points on the B-spline surface in the equation of the implicit surface of the tool. This collision detection technique does not allow a point- or parametric/B-spline-based tool, which limits the scope of sculpting. Greß [13] used GPU based collision detection technique for parametric surfaces. Hughes [15] presented an algorithm for a surface-surface intersection test for Bezier and B-spline surfaces. However, the method needs to update the Axis Aligned Bounding Box (AABB) tree for each sub-patch and find a solution of many algebraic equations. This algorithm tends to fail, when the area of contact is large.

3. B-SPLINE SURFACE

A B-spline surface with r and s number of control points in u and v directions, respectively is given by the equation [25],

$$S(u,v) = \sum_{i=0}^{r-1} \sum_{j=0}^{s-1} B_{ik}(u) B_{jl}(v) \mathbf{P}_{ij} \tag{1}$$

where $\mathbf{P}_{ij}$ is the control points vector, $B_{ik}(u)$ and $B_{jl}(v)$ are the blending functions of the surface with degree k and l in u and v directions respectively, and in general $0 \le u, v \le 1$. The blending functions depend upon whether the surface is periodic or non-periodic (in u or v or both directions), knot vector, the degree of the surface and the number of control points in u and v directions. The blending functions are independent of the position of the control points. This surface can be discretized into a set of a parametrically uniform grid of nodes for various values of u and v as shown in Fig. 1.

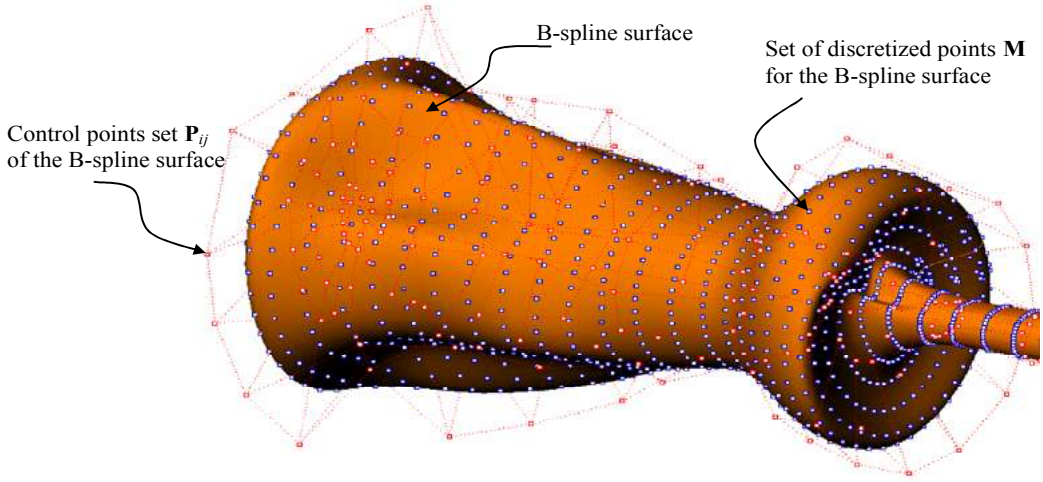


Fig. 1: Discretized B-spline surface.

The set of nodes **M** is given by the equation,

$$\mathbf{M} = \mathbf{A}_u \mathbf{P}_{ij} \mathbf{A}_v \tag{2}$$

The equation can be re-written to find the position of control points as:

$$\mathbf{P}_{ij} = [\mathbf{A}_u{}^T \mathbf{A}_u]^{-1} \mathbf{A}_u{}^T \mathbf{M} \mathbf{A}_v{}^T [\mathbf{A}_v \mathbf{A}_v{}^T]^{-1} \tag{3}$$

where $\mathbf{A}_u$ and $\mathbf{A}_v$ are transformation matrices and their values depend upon blending functions and u and v parametric values only. The transformation matrices and their inverses ($[\mathbf{A}_u{}^T \mathbf{A}_u]^{-1}, [\mathbf{A}_v \mathbf{A}_v{}^T]^{-1}$) are not effected by the positions of the control points and, therefore, can be pre-calculated and stored.

4. PROPOSED ALGORITHM

In this section the proposed algorithm for haptic manipulation and sculpting of deformable B-spline surfaces is discussed in detail. The deformation and haptic force response is estimated using a mesh-based model.

4.1 Overview

The proposed algorithm has two phases: the pre-processing phase and the run-time phase. During the pre-processing phase the transformation matrices of the given B-spline surfaces, along with their inverses, are calculated and stored. A convex hull is generated from the control point set.

The run-time phase starts by checking for the intersection of convex hulls of the B-spline surfaces. If the convex hulls are intersecting, then the corresponding minimum and maximum values of u and v parameters, associated with the control points of these surfaces, are determined. Within this range of u and v, sparse points are generated on the surfaces of the model and the tool by using intermittent rows and columns of the transformation matrices. These points are used to generate spheres on both the model and the tool and the spheres are then checked for intersection. More points are generated within these intersecting spheres. This process of generating spheres, at the lower levels of detail, continues until all the rows and columns for the transformation matrices have been used. The process of generating lower levels of detail is terminated if, at a particular level of detail, all the spheres on the model and the tool are intersecting. This is possible if the curvature of the model and the tool is similar in the region of probable collision. Fig. 2 shows the flow chart of the proposed collision detection algorithm. The graphical rendering and haptic force response system are also shown.

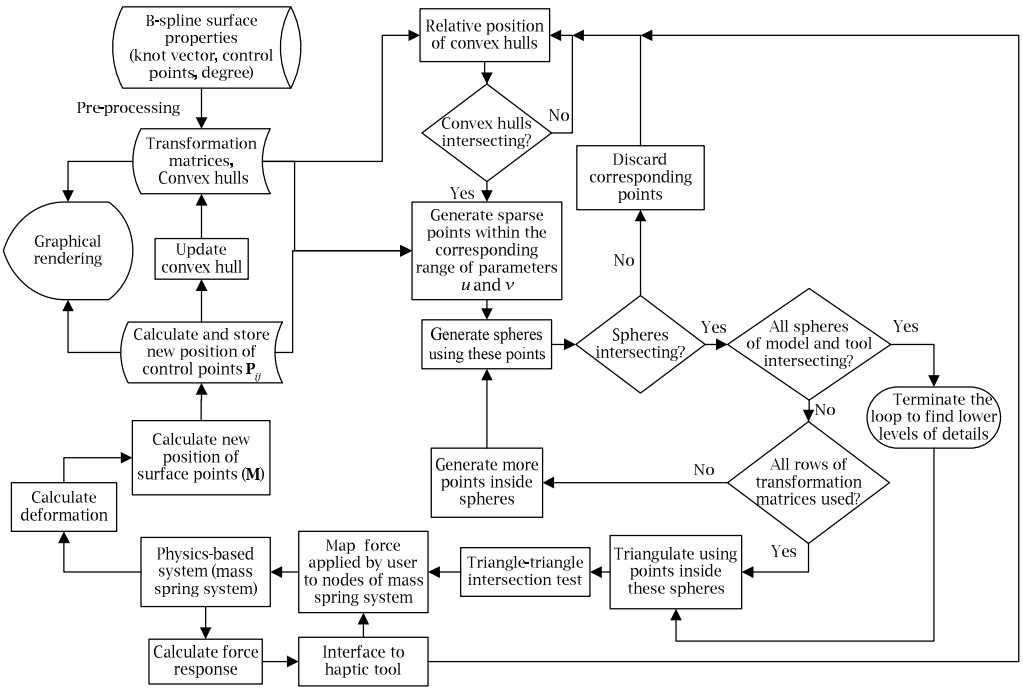


Fig. 2: Flow chart of the proposed algorithm.

The points within the intersecting spheres at the lowest level of detail are subsequently used to generate a triangulated mesh. The triangle-triangle intersection test is then carried out to find out the parts of the surfaces which are intersecting. This information is used to map the forces, applied by the user, to the nodes of the mass spring system. The physics-based model then determines the resultant deformation and the reactive forces to be sent back to the user. Using the inverse transformation matrices, new positions of the control points are determined and the convex hull for the B-spline surface is updated. The major steps of the algorithm are elaborated in detail in following subsections. Illustrations show the steps of the collision detection process for only the model, to maintain clarity.

4.2 Generation of Transformation Matrices

During the pre-processing phase, transformation matrices are calculated based upon the blending functions of the B-spline surface and the maximum number of points to be generated and then stored

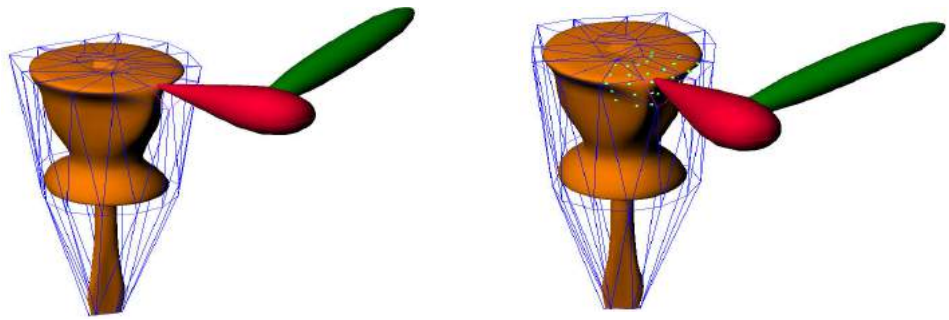
along with their inverses. Two transformation matrices for generating points on the B-spline surface are needed for each B-spline surface. Similarly, two transformation matrices are needed to find tangents at each point generated. These are computed and stored along with their inverses. The tangent information is necessary for modeling realistic properties such as friction. These tangents are also used to calculate the surface normal at the generated points.

4.3 Bounding Volume

The property of *positivity* ensures that the B-spline curve or surface always remains within the convex hull of the control points $\mathbf{P}_{ij}$ [4]. This makes the convex hull a good choice as a bounding volume. The position of the control points is always known and their number is limited which ensures that the cost of updating the bounding volume (convex hull) is minimal.
The algorithm used for the generation of a convex hull is similar to the *Quickhull Algorithm* [2]. For the worst case scenario the computational cost of this algorithm is $\boldsymbol{O}$ (n log n) where n is the number of control points. Once the physics-based force response system determines the deformation, the new set of control points is calculated using Eqn. (3). Since the inverses of transformation matrices are already stored, the computational cost during the run-time phase is substantially decreased. The convex hull is regularly updated as the positions of the control points change.

4.4 Surface Discretization

When two convex hull surfaces (or edges or an edge and a surface) intersect; the vertices (control points) of the surface (edge) are identified and noted. The corresponding u and v parametric values associated with these vertices (control points) of the intersecting surfaces are calculated. These minimum and maximum values of u and v (u_{min}, v_{min}; u_{max}, v_{max}) set the limits on the surface to be discretized. Limiting the area for discretization lowers the computation cost. Intermittent rows and columns of the transformation matrices are used to sparsely generate points within these limits of parametric values of u and v. Fig. 3 illustrates the generation of sparse points within the region of probable collision.



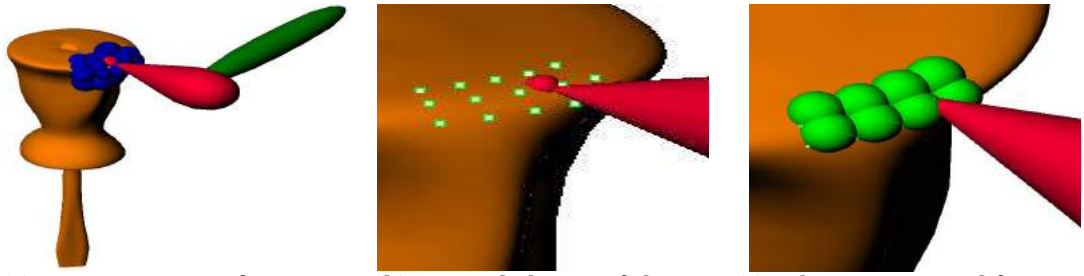
(a) Tool intersecting the convex hull (triangle) of the model

(b) Points generated within the minimum and maximum u and v parametric values

Fig. 3: Illustrations showing discretization of the region of probable collision of the model surface.

4.5 Generation of Lower Levels of Detail

The region of probable collision detection, as calculated in Section 4.4, is initially large. By using lower levels of detail, the region is further refined and simultaneously the intensity of points are increased to enhance the accuracy of collision detection. Spheres have been used to generate lower levels of detail. The algorithm to generate spheres starts from the point at u_{min}, v_{min}. It compares the diagonal distance between an array of four consecutive sparse points in u and v direction. Two points of longer diagonal distance and one of the two remaining points are selected to generate a unique circle in space. The sphere is generated with the same centre point and radius as the circle. This ensures that the fourth point lies on or within the boundary of this sphere and the size of the sphere is optimal. This process continues with the points at next values of u and v until all the points have been used. Fig. 4 illustrates the various steps for generating lower levels of detail.



(a) Intersection test for spheres generated from the discretized points

(b) Expanded view of the points generated within intersecting spheres

(c) Spheres generated from the points at lower level of detail

Fig. 4: Illustrations showing generation of lower levels of detail.

Once the process of sphere generation from sparse points is complete for the model and the tool, these spheres are used for determining intersection. The points associated with the non-intersecting spheres are discarded and more points are generated within the intersection spheres. These points are again used to generate more spheres. This process of generating lower levels of detail continues until all the rows and columns of transformation matrices have been used.

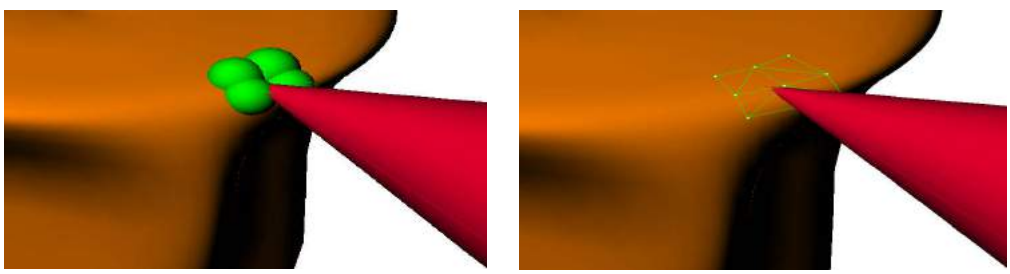
4.5.1 Interaction of Flat Surfaces

A computationally expensive situation will emerge during the interaction of two flat surfaces. There is always a strong possibility of encountering a flat surface when both the model and the tool are deformable. In such situations, the area of contact will be large. Many of the algorithms discussed in literature [9, 15], cannot tackle a large area of contact. All the spheres generated at any particular level of detail or lower levels will be intersecting. Thus, generating lower levels of detail will only add to the computation cost. In the proposed algorithm the loop for generating lower levels of detail is terminated, if at a particular level, all the spheres generated for both the tool and the model, are intersecting. In such a case the next step of tessellation of the surface is carried out.

The algorithm proposed in this paper uses a novel method to reduce the number of triangles generated on the surface in such situations. It compares the normals of the points being generated in the region. If the normal of a point being generated is within the limits of that of the previous point by a small angle δ, this point is not generated. This ensures that the number of points, to be used for triangulation of the surface, is minimal and the accuracy of the surface is also not compromised.

4.6 Triangulation of Surface

Fig. 5 illustrates the triangulation of a surface by using the points of intersecting spheres.



(a) Intersecting spheres at the lowest level of detail

(b) Triangulation using the points associated with the intersecting spheres

Fig. 5: Illustrations showing triangulation in process.

The triangulation of the surface starts by using the points within the u and v parametric values of intersecting spheres. These points are parametrically uniform, but in Cartesian space the distance between the points may not be uniform. If any algorithm from literature based on the Delaunay triangle is used, it mostly fails to tessellate some regions of the surface and sometimes wrongly tessellates a cavity. The method of generating spheres can be easily and efficiently adapted as an alternate procedure. According to one of the definitions of the Delaunay triangle, "All interior edges of a triangulation Δ of a point set are locally optimal if and only if no point from this point set is interior to any circum-circle of the triangle in Δ" [22]. The spheres are generated in such a way that three points are always on the sphere surface and the fourth point will be on or within the sphere. This information is then used to generate two optimal triangles within an intersecting sphere. If there are many intersecting spheres with common points, the same process will result in a triangulated mesh. This method of triangulation of the surface ensures that there are no 'holes' in the triangulated mesh.

4.7 Triangle-Triangle Intersection Test

The simplest approach for testing all the triangles of the model against all the triangles of the tool requires an immense number of triangle-triangle intersection tests. The proposed algorithm limits the number of triangles of the model and the tool requiring intersection test. Only those triangles, whose bounding spheres are intersecting, need to be checked for intersection. As the computational cost of a sphere-sphere intersection test is far less than that of a triangle-triangle test, the overall computational cost is reduced. Many efficient algorithms are available in literature for a triangle-triangle intersection test. An algorithm, similar to that developed by Guigue [14], is used for a triangle-triangle intersection test. One major limitation is that it cannot tackle degenerate cases. The dynamic generation of points in the uniform grid of parameters u and v, substantially reduces the chances of degeneracy.

4.8 Mapping of Forces and Deformation of Model

The forces acting on the points of contact, as determined in Section 4.7, are mapped on to the nearest nodes of the mass spring system. The force is computed as weighted bilinear interpolation of the distance of the nearby four nodes of the mass spring system to the point of contact in the parametric domain. Since the parametric values of nodes as well as of the points of contact are known, the virtual forces can be easily calculated. Fig. 6 shows a case in which the intersection is happening at a point.

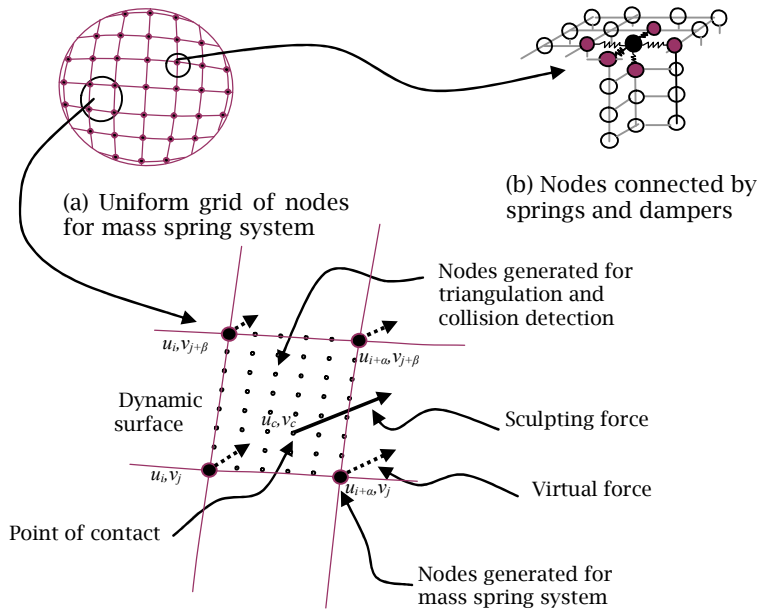


Fig. 6: Mapping of force acting on B-spline surface to nodes of mass spring system.

When two surfaces are in contact, there may be many points on which the forces are acting. The total virtual forces will then be the vector sums of the virtual forces calculated for each point of contact.
Once the forces acting on the nodes of the mass spring system are known, Lagrangian equations of motion are used to find out the new position of the nodes and the subsequent deformation of the model. The system dynamics are given by the second-order differential equation

$$\rho_i \ddot{x}_i(t) + \gamma_i \dot{x}_i(t) + g_i(t) = f_i(t) \tag{4}$$

where ρ_i is the point mass of node *i*, γ_i is the velocity-dependent damping coefficient which dissipates kinetic energy in the lattice through friction, $g_i(t)$ is the total internal spring forces, and $f_i(t)$ is the external force vector applied to node *i*. At each time step Δt it is necessary to evaluate the current nodal forces and accelerations, the new velocities, and the new node positions using the explicit Euler time-integration procedure [18]. From Eqn. (4) it is possible to compute acceleration at node *i* as

$$\ddot{x}_i(t) = (f_i(t) - \gamma_i \dot{x}_i(t) - g_i(t)) / \rho_i \tag{5}$$

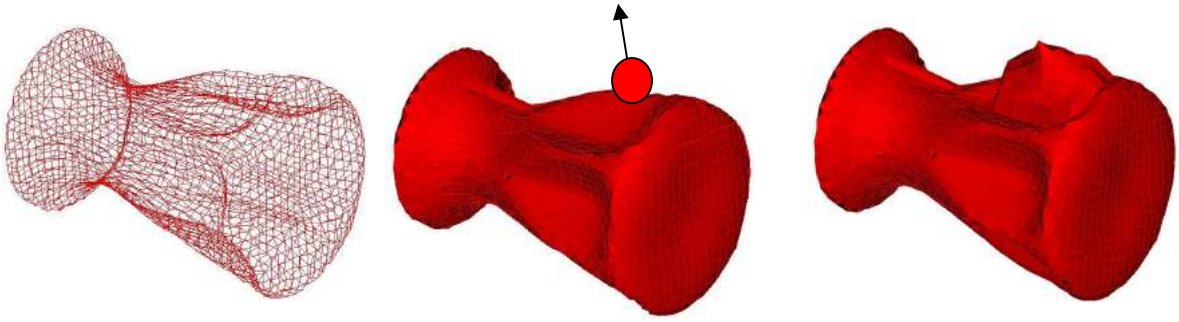
and the new velocity can be computed as

$$\dot{x}_i(t + \Delta t) = \dot{x}_i(t) + \Delta t \ddot{x}_i(t + \Delta t) \tag{6}$$

The new position of node *i*, is then calculated using the equation:

$$x_i(t + \Delta t) = x_i(t) + \Delta t \dot{x}_i(t + \Delta t) \tag{7}$$

Once the user applies force on the model through the tool, eqn. (4-7) determine the deformation of the model. Fig. 7 shows the hexahedron mass spring mesh of the model and its deformation in response to the force applied on the model.



(a) The hexahedron mass spring mesh generated for the model.

(b) Mass spring mesh before application of force.

(c) Mass spring mesh after application of force.

Fig. 7: Deformation of the mass spring system of the model due to the force acting on B-spline surface.

4.9 Special Cases

The algorithm described above is for collision between two or more B-spline surface model(s) and/or the sculpting tool. In special cases a tool can be represented as an implicit surface tool or a point-based tool. This algorithm is capable of handling these special cases efficiently.

4.9.1 Implicit Surface Tool

In the special case of an implicit surface representation of the tool, only a part of the algorithm is used. First the tool is checked for intersection with the convex hull of the B-spline. Once the implicit surface intersects the convex hull, the algorithm determines the corresponding values of *u* and *v* parameters for the intersecting surfaces which lie within the convex hull. It then generates points on the surface within these limits using the transformation matrices. These points are then inputted into the implicit surface equation to detect collision. If the points lie on or inside the surface, collision is detected and penetration depth is determined. Since the points are generated only within a small region which has a

high probability of collision, the overall computational cost for determining collision detection is nominal.

4.9.2 Point-based Tool

In case of a point-based tool, the process discussed in previous subsections is followed for the model only. More points are generated in a sphere with which the point-based tool is intersecting and the process of generating lower levels of detail continues until all the rows and columns of the transformation matrices have been used. Finally, two triangles will be generated within the intersecting sphere at the lowest level of detail. The intersection test of the point with these two triangles then determines the point where the tool is colliding with the model.

5. COMPARISON AND PERFORMANCE

The proposed approach is to exploit the advantages of the B-spline surface representation and the computational efficiency of a triangle-triangle intersection test. In the following subsections the algorithm is compared quantitatively and qualitatively to prove its efficiency and robustness.

5.1 Quantitative Comparison

Bordegoni [5] relied on the algorithm of Baraff [1] for the intersection test of a tessellated deformable surface. Gottschalk [11] has proposed that his algorithm can be adopted for deformable bodies. The worst case scenario, represented as Big '***O***' notation, can be used to compare the results of the proposed algorithm with that of a tessellated surface model used by [5, 11]. For comparison, OBB is used as the bounding box [11] for a deformable model represented as a tessellated surface.

For a tessellated model the total cost of computation is comprised of the cost for generation of the bounding box for t number of triangles, $\boldsymbol{O}$ ($t \log t$); the cost of collision prediction of bounding boxes, $\boldsymbol{O}$ (1); and the cost of octree subdivision $\boldsymbol{O}$ (cd^2) (for subdividing the model into c regions having d number of triangles). For the proposed algorithm, the total cost of computation includes the cost for the bounding box (convex hull with n control points) generation, $\boldsymbol{O}$ ($n \log n$); the cost of collision prediction, $\boldsymbol{O}$ (n); the cost of discretization of the B-spline surface with m points, $\boldsymbol{O}$ (n^2m); the cost of generation of spheres $\boldsymbol{O}$ (m); the intersection test of spheres, $\boldsymbol{O}$ (m^2); tessellation of the region of probable collision, $\boldsymbol{O}$ ($m \log m$) and the cost of collision detection, $\boldsymbol{O}$ (m^2). The Fig. 8 shows the comparison of the computation cost of the tessellated surface model with that of the proposed B-spline surface model for various control points. The results are for an '*area of contact*' of two percent of the model surface area, assuming an even distribution of triangles on the surface of the model. Large area of contact is considered which is typical of the application in virtual sculpting during conceptual design.

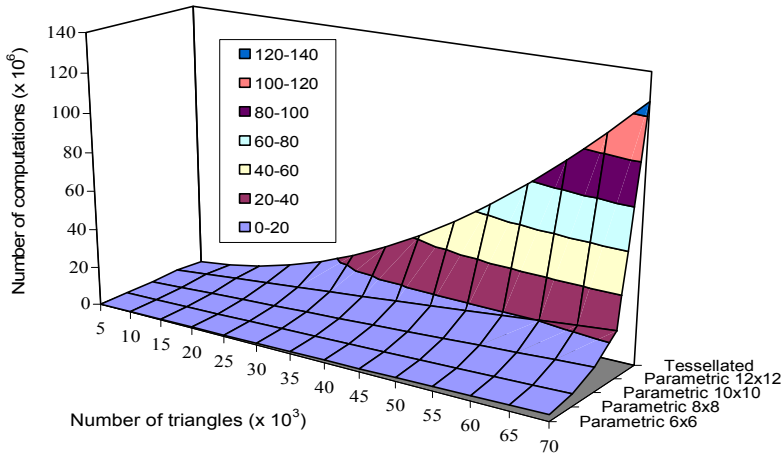


Fig. 8: A comparison of total computations required in the worst case scenario for collision detection of the tessellated model versus the B-spline model using the proposed algorithm.

The X axis shows the number of triangles for the tessellated model and the equivalent number of triangles for the B-spline surface model. The Y axis shows the number of computations required for the worst case scenario. It is clear from the graphs that the computational cost of the proposed algorithm is much less than that of a tessellated model. This reduction in computational cost is more pronounced for a higher resolution.
Another aspect of sculpting is that during the early phase, the contours are not very sharp but the area of contact is very large. As the sculpting process progresses further, the details get finer and the area of contact is reduced. The proposed algorithm can aptly take advantage of this fact.

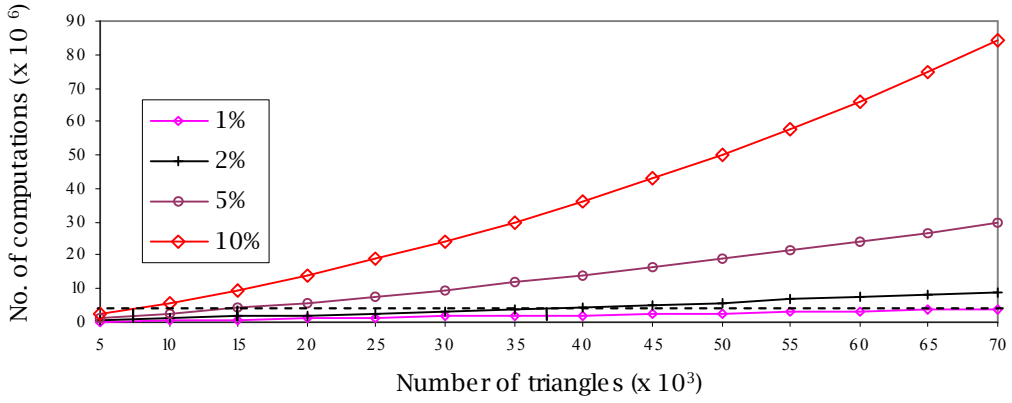


Fig. 9: Maintaining a lower number of computations by changing resolution of the surface.

Fig. 9 shows the total computation cost of collision detection for different areas of contact for a 10 × 10 control point B-spline surface model. The number of points generated on the surface can be decreased by using fewer rows and columns from the transformation matrices as discussed in previous sections. Thus, in the beginning when the surface to be sculpted does not have finer details but a large area of contact, the user can specify a lower resolution. This will reduce the number of points generated and therefore the computational cost will be lower. It is shown by the dotted horizontal line in Fig. 8. By changing the resolution of the model (using a fewer number of rows and columns of transformation matrices) a lower number of computations can be maintained. As the sculpting process progresses to the finer details, the number of points generated and hence the resolution and accuracy of collision detection can be increased without increasing the computational cost because the area of contact is simultaneously reduced.

5.2 Qualitative Comparison

During the haptic interaction with a tessellated surface model the quality of the triangles deteriorates as the model deforms. This makes the collision detection algorithm somewhat inefficient. In the proposed algorithm the points, and consequently the triangles, are generated '*on the fly*'. Thus the quality of the triangles is maintained even if the model experiences high deformation.
Most of the collision detection algorithms for parametric surfaces are based on subdividing the surface into patches and further sub-patches until these are sufficiently plane. This is done during the pre-processing phase or the run-time phase. These sub-patches are then bounded by bounding boxes, mostly *Axis Aligned Bounding Boxes* (AABB) [15]. If this process is carried out during the run-time phase, it is time consuming and computationally very intensive. Alternatively, if it is done during the pre-processing phase, the subsequent deformations of the model make the subdivision prone to large errors and inefficient collision detection. The proposed algorithm is capable of efficiently detecting the collision for any type of B-spline surface and any subsequent deformation does not affect the accuracy of collision detection.
Most of the previously published algorithms [9, 15] also tend to fail when large surface areas are in contact. The novel technique of terminating the loop for generating lower levels of detail and comparing the normals of the points generated, as discussed in Section 4.5.1, ensures a lower cost of

computation even for large areas of model contact. The user is also given the freedom to control the number of points generated. The computational cost can also be decreased by choosing lower resolution when the area of contact is large. This increases the robustness of the system.
Gao [10] used an implicit surface to represent the tool and used points generated to discretize the B-spline surface to detect collision. At any time, all the points are input in an implicit surface equation of the tool to determine if the surface is colliding with the tool. The minimum diameter of the sphere has to be more than the maximum distance between the discrete points on the B-spline surface. This largely limits the applications of the technique because it cannot handle a B-spline surface- or point-based tool. Another advantage of the proposed algorithm over the collision detection algorithm presented is that it uses the points only in the region of probable collision and not on the entire surface of the model. This reduces the cost of collision detection. Furthermore, if the tool and model are apart (convex hull of B-spline model not intersecting with implicit surface), there is no need to input a large number of points in the equation of implicit surface to detect collision.

6. CONCLUSION

In this paper a novel method for collision detection for physics based model simulation has been described and verified for computational efficiency. This method utilizes the combined advantages of parametric surface representation and the ease and efficiency of a triangle-triangle intersection test. The proposed method of storing pre-calculated transformation matrices lowers the cost of computation during the run-time phase. The transformation matrices are also independently used to generate nodes for the mass spring system. Both the model and the tool can have complex shapes, elastic or plastic properties, and multiple contacts. This allows the user to use rigid or deformable tools with complex shapes with greater ease and productivity during the sculpting or concept validation in a virtual reality environment. The '*on the fly*' generation of points and triangles helps maintain the quality of triangles. At the same time, resolution of the model can be varied during haptic interaction, as needed, thereby reducing the overall computational cost. The collision detection algorithm is independent of a physics-based model used to calculate deformation. The number of nodes of the mass spring system is determined solely based on the accuracy of the force response and is independent of the number of points generated for collision detection. This allows graphical representation of the model, collision detection and haptic force response through the mass spring system, independent of each other. This also makes the proposed haptic design module, suitable for parallel processing, thereby reducing the computation time.

7. ACKNOWLEDGEMENTS

This work has been supported, in part, by Natural Sciences and Engineering Research Council (NSERC) of Canada. The authors also acknowledge the support from the National Research Council of Canada in London, Ontario, Canada.

8. REFERENCES

[1] Baraff, D.: Curved surfaces and coherence for non-penetrating rigid body simulation, Computer Graphics, 24(4), 1990, 19-28.
[2] Barber, C. B.; Dobkin, D. P.; Huhdanpaa, H.: The quickhull algorithm for convex hulls, ACM Trans. on Mathematical Software, 22(4), 1996, 469-483.
[3] Basdogan, C.; Suvranue, D.; Jung, K.; Muniyandi, M.; Kim, H.; Srinivasan, M.: Haptics in minimally invasive surgical simulation and training, IEEE Computer Graphics and Applications, 24(2), 2004, 56-64.
[4] Bloomenthal, J.; Bajaj, C.; Blin, J.; Gascuel, M.; Rockwood, A.; Wyvill, B.; Wyvill, G.: Introduction to implicit surfaces, ed., Morgan Kaufmann Publishers Inc, San Fransisco, CA, 1997.
[5] Bordegoni, M.; Cugini, U.: Create free-form digital shapes with hands. In: Proceedings of Int. Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia, Dunedin, New Zealand, 2005, 429-432.
[6] Cheshire, D.; Evans, M.; Dean, C.: Haptic modeling an alternative industrial design methodology? In: Proceedings of EuroHaptics 2001, Birmingham, UK, 2001, 124-129.
[7] Dachille, F.; Kaufman, A.; Qin, H.: A novel haptics based interface and sculpting system for physics-based geometric design, Computer Aided Design, 33(5), 2001, 403-420.

[8] Ehmann, S.; Lin, C.: Accurate and fast proximity queries between polyhedra using convex surface decomposition, Computer Graphics Forum, 20(3), 2001, 500-510.
[9] Galoppo, N.; Tekin, S.; Otaduy, M. A.; Gross, M.; Lin, M. C.: Interactive haptic rendering of high-resolution deformable objects. In: Proceedings of 2nd Int. Conference on Virtual Reality, Beijing, China, 2007, 215-223.
[10] Gao, Z.; Gibson, I.: Haptic sculpting of multi-resolution B-spline surfaces with shaped tools, Computer Aided Design, 38(6), 2006, 661-676.
[11] Gottschalk, S.; Lin, M. C.; Manocha, D.: OBBTree: A hierarchical structure for rapid interference detection. In: Proceedings of 23rd Annual Int. Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 96, ACM Press, 1996, 30, 171-180.
[12] Gregory, A.; Lin, M.; Gottschalk, S.; Taylor, R.: Fast and accurate collision detection for haptic interaction using a three degree of freedom force feedback device, Computational Geometry, 15(1), 2000, 69-89.
[13] Greß, A.; Guthe, M.; Klein, R.: GPU-based Collision Detection for Deformable Parameterized Surfaces, Computer Graphics Forum, 25(3), 2006, 497-506.
[14] Guigue, P.; Devilers, O.: Fast and robust triangle-triangle overlap test using orientation predicates, Graphics Tools, 8(1), 2003, 25-42.
[15] Hughes, M.; DiMattia, C.; Lin, M.; Manocha, D.: Efficient and accurate interference detection for polynomial deformation. In: Proceedings of the IEEE Computer Animation Conference, Washington DC, USA, 1996, 155.
[16] Jimenez, P.; Thomas, F.; Torras, C.: 3D collision detection: A survey, Computer and Graphics, 25(2), 2001, 269-285.
[17] Klosowski, J. T.; Held, M.; Mitchell, J.; Sowizral, H.; Zikan, K.: Efficient collision detection using bounding volume hierarchies of k- dops, IEEE Trans on Visualization and Computer Graphics, 4(1), 1998, 21-36.
[18] Knopf, G. K.; Igwe, P. C.: Deformable mesh for virtual shape sculpting, Robotics and Computer Integrated Manufacturing, 21(4), 2005, 302-311.
[19] Krishnan, S.; Gopi, M.; Lin, M. C.; Manocha, D.; Pattekar, A.: Rapid and Accurate Contact Determination between Spline Models using Shell Trees, Computer Graphics Forum, 17(3), 1998, 315-326.
[20] Lin, M. C.; Manocha, D., Collision and proximity Queries, in Goodman, J. E.; O'Rourke, J., eds., Handbook of Discrete and Computational Geometry, Chapman & Hall, Boca Raton, Florida, USA, 2004.
[21] Mendoza, C.; O'Sullivan, C.: Interruptible collision detection for deformable objects, Computer & Graphics, 30(3), 2006, 432-438.
[22] Øyvind, H.; Morten, D., eds., Triangulations and Applications, Springer-Verlag Berlin Heidlberg, The Netherlands, 2006.
[23] Palmer, I. J.; Grimsdale, R. L.: Collision detection for animation using sphere-trees, Computer Graphics Forum, 14(2), 1995, 105-116.
[24] Patoglu, V.; Gillespie, R. B.: A closest point algorithm for parametric surfaces with global uniform asymptotic stability. In: Proceedings of 1st Joint Eurohaptic Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleporter System (WHC'05), Pisa Italy, 2005, 348-355.
[25] Piegl, L.; Tiller, W.: The NURBS book, 2nd ed., Springer, New York, 1997.
[26] Pungotra, H.; Knopf, G. K.; Canas, R.: Efficient algorithm to detect collision between deformable B-spline surface for virtual sculpting, Computer-Aided Design, 40(10-11), 2008, 1055-1066.
[27] Thompson, T.; Johnson, D.; Cohen, E.: Direct haptic rendering of sculptured models. In: Proceedings of 1997 Symposium on Interactive 3D Graphics, ACM New York, NY, USA, Providence, Rhode Island, United States, 1997, 167-176.
[28] Ye, J.; Campbell, R.; Page, T.; Badni, K.: An investigation into the implementation of virtual reality technologies in support of conceptual design., Design Studies, 27(1), 2006, 77-97.