COLLISION DETECTION AND PENETRATION DEPTH CALCULATION IN VIRTUAL SURGICAL SIMULATION

by

RUPIN PAVITHRAN

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

DECEMBER 2008

ACKNOWLEDGEMENTS

I would like to thank my supervising professor, Dr. Venkat Devarajan, for his support and encouragement during the period of my study at The University of Texas at Arlington. I am also grateful to my thesis committee members, Dr. W. Alan Davis and Dr. Karel J. Zuzak. Sincere thanks to UTA, my Alma Mater, for recognizing my aptitude and giving me an opportunity to pursue my master's studies under the guidance of noble faculty and for providing me with all the required resources and environment for success.

I am grateful to all my lab mates, especially Dibbesh Sharma Adhikari and Koyel Mukherjee for their help in the work. I would also like to express my thanks to Dr. Yunhe Shen for his valuable remote support in the work. I would like to thank my roommates, my friends for their undying encouragement and understanding.

I cannot end my acknowledgement without expressing my gratitude to my parents and sisters for their love and care.

November 20, 2008

ABSTRACT

COLLISION DETECTION AND PENETRATION DEPTH

CALCULATION IN VIRTUAL SURGICAL

SIMULATION

Rupin Pavithran, M. S.

The University of Texas at Arlington, 2008

Supervising Professor: Dr. Venkat Devarajan

Virtual Reality (VR) based surgical simulators create a simulated, realistic three dimensional surgical environments using advanced graphic and haptic rendering techniques. Virtual objects, which are geometric surface polygonal models of different human organs and instruments, are rendered in a common viewing volume. Such deformable or non-deformable polygonal models interact with each other. In order to provide a realistic response in real-time, detection of collisions in such models is the greatest challenge in the field of VR based simulators. Once collision is detected, the overlapping region and the largest penetration distance for a pair of intersecting objects need to be determined so that realistic deformations of the objects can be calculated and rendered.

This thesis presents a simple and efficient algorithm to detect collisions between two objects and to calculate the penetration depth from the overlapping region. In a surgical simulator, there are two types of object pair interactions: deformable to non-deformable and, deformable to deformable object interactions. This work examines tissue to rigid body and, tissue to tissue interactions and provides realistic deformation to the interacting tissues in both cases.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

1.1 Virtual Reality

Virtual Reality (VR) is a technology that allows a user to interact with a computer-simulated environment, so that he can manipulate and explore while feeling as if he were in that world. Most current VR environments are primarily visual experiences, displayed either on a computer screen or through special stereoscopic displays. There are advanced systems, which include additional sensory information, such as sound through speakers or headphones and, tactile information through haptic or force feedback devices. VR has a wide variety of applications in different fields such as flight simulation and medical simulation. VR based flight simulators are extensively used by the aviation industry for the design and development of prototype airplanes and, training of pilots. VR is finding its way into the training of healthcare professionals. VR based surgical simulators recreate the actual surgical procedure in a virtual environment which is very useful for the purpose of training, rehearsal or experiment. Virtual surgery is carried out with the help of advanced techniques and instruments in the field of Electronics, Robotics and Computer Graphics to create human machine interfaces and, to provide visual and force feedback to the user. With the help of these simulators, the doctors-in-training can practice their skills on extremely realistic simulators. Training surgeons in this way may create better surgeons and safer surgeries, according to a new study by New York-Presbyterian Hospital. Some of the rare pathological conditions and emergency procedures can only be trained on a simulator. Moreover, the trainees can repeatedly practice some of the complex operative tasks before entering the operation room, thereby improving patient safety. It is a great learning tool where one is allowed to go back and show the trainees what went wrong and in most cases there can be an objective evaluation of the surgeons' dexterity combined with a more intensive training activity. It brings more engagement and realism to the process. Virtual training systems can improve trainees' learning curves with safety, efficiency, flexibility and without the fear and anxiety of performing surgical procedures on a real patient for the first time.

1.2 Major issues in Surgery Simulators

1.2.1 Modeling of complicated objects

In recent years, the area of three dimensional modeling and visualization of medical data set has received great attention from the research community. The majority of these systems aid in the diagnosis and treatment planning aspects of the health care process. While realistic three dimensional models are very useful for operations like rotation, zooming and making various parts of anatomy transparent, there is need for anatomical models which behave dynamically. This means that the models should respond to deformation and manipulation in a manner similar to that in real anatomy.

There are many types of model representations used in 3D graphics. The representations are broadly classified as shown in the Figure 1.1.

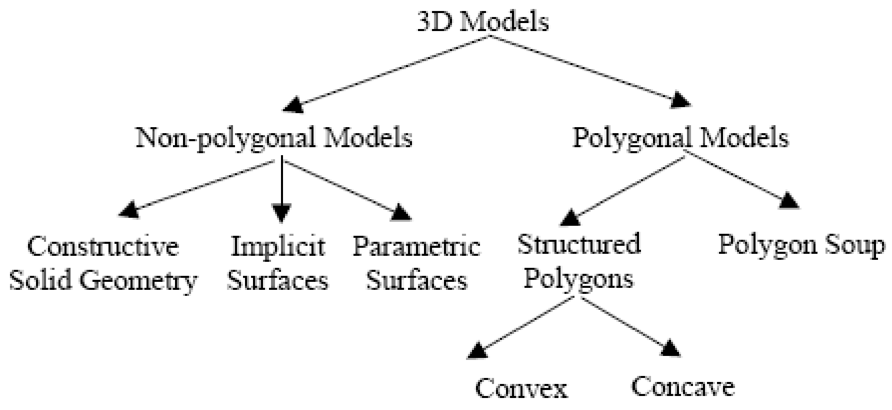


Figure 1.1. Classification of 3D model representation

Polygonal modeling is an approach for modeling objects by representing or approximating their surfaces using polygonal meshes. The basic unit used in mesh modeling is a vertex, a point in 3D space. Three vertices connected in a particular order forms a triangle, which is the simplest polygon in Euclidean space. The flat nature of triangles makes it simple to determine their surface normal, which is used for determining lighting and coloring. A group of polygons which are connected together by shared vertices is referred to as a mesh. Once a polygonal mesh has been constructed, further steps must be taken before it is useful for various applications like simulation, gaming, animation etc. The model must be texture mapped to add colors and texture to the surface. A major disadvantage with polygons is that it is incapable of representing curved surfaces, so a large number of them must be used to approximate curves in a visually appealing manner. The use of complex models has the cost of lowered speed during rendering.

1.2.2 Detecting Collisions

Collision detection has been a fundamental problem in computer animation, physically-based modeling, geometric modeling and robotics. In a virtual environment filled with virtual objects the user should be able to feel realism, i.e. objects should behave as in the real world; they should not pass through each other, and things should move as expected when pushed, pulled or grasped. Since these virtual objects are represented as polygonal meshes which are stored in data structures and rendered on the computer screen for display, there should be a technique to check whether these objects share the same volume in 3D (i.e., collide). A collision detection algorithm should be able to detect such instances, where different objects intersect or overlap each other. A fast and interactive collision detection algorithm is the fundamental component of a complex virtual environment.

The obvious problem that arises in detecting collision between all N objects is the $O(N^2)$ computation problem. Several techniques have been proposed to deal with it; one of the most common among them is the hybrid collision detection approach [38]. This approach refers to the division of the detection task into broad phase and narrow phase operations. The broad phase operations are preprocessing steps that indicate the possibility of a collision. Narrow phase operations are applied only to those data which pass the broad phase test. The narrow phase detects if the collision actually occurs.

Different techniques are used to accomplish this broad phase detection, such as sweep and prune [2] global bounding tables [3] or overlap tables [4]. Once the potential colliding pairs are found using a broad phase approach, the more computationally expensive exact intersection tests can be performed on these short listed pairs using a narrow phase approach. Lin-Canny [5], V-Clip [6] or I-Collide [7] approaches may be used in narrow phase detection. The different factors which affect collision detection are categorized based on

- Object representation: Objects are most commonly represented as polygonal meshes, with triangles as the fundamental primitives. This is the explicit way of representing an object which is defined in terms of vertices, edges and faces.
- Types of queries: Most straightforward collision query is the intersection testing. It generates a Boolean answer of whether the objects have collided or not. If objects penetrate, one may need to find the penetration depth. The penetration depth is the shortest vector over which the object needs to be translated to separate them.
- Number of objects: In a scene with N objects, $O(N^2)$ pair wise tests may be required to perform the collision test, but due to the quadratic time complexity, testing each pair for collision would become too expensive when the number of objects increases. This complexity is solved by the hybrid collision detection algorithm.

Most of the earlier work in collision detection has focused on algorithms for convex polytopes. Using hierarchical representations, an $O(\log^2 n)$ algorithm is given in

[21] for a polytope-polytope overlap problem, where n is the number of vertices. This elegant approach has not been robustly implemented in 3D, however. Good theoretical and practical approaches based on the linear complexity of linear programming problems are known [22], [23]. Minkowski difference and convex optimization techniques are used in [24] to compute the distance between convex polytopes by finding the closest points. In applications involving rigid motion, geometric coherence has been exploited to design algorithms for convex polyhedral, based on local features [25], [26], [27]. A number of hierarchies have been used for collision detection between general polygonal models. Typical examples of bounding volumes include axis-aligned boxes and spheres. They are chosen for their fast overlap tests. Other structures include cone trees, k-d trees, sphere trees [29] etc. All of these hierarchical methods do very well in performing the "rejection tests" whenever two objects are far apart. However, when the two objects are in close proximity and can have multiple contacts, these algorithms either use subdivision techniques or check a very large number of bounding volume pairs for potential contacts. In such cases their performance slows down considerably. More recent work seems to have focused on tighter-fitting bounding volumes. Gottschalk et al. [30] have presented a fast algorithm and a system called RAPID, for interference detection based on oriented bounding boxes, which approximate geometry better than axis-aligned bounding boxes.

More recently, Cohen et al. [31] have presented algorithms and a system, I-COLLIDE, based on spatial and temporal coherence, for large environments composed of multiple moving objects. The number of object pair interactions is reduced to only the pairs within close proximity by sorting axis-aligned bounding boxes (AABBs). It is output sensitive and its run time is linearly dependent on the number of objects in the environment.

In the spatial tessellation technique, which is used in this thesis, the 3D viewing space is divided into unit cells (or volumes) and the object occupancy information of each cell is stored in some form [32]. To check for collisions, the occupancy information is checked to verify if the cells are shared by other objects. However, it is difficult to set a near optimal size for each cell. Therefore, the technique requires a tremendous amount of allocated memory. If the size of the cell is not properly chosen, the computation can also be expensive. However, Overmars [32] has shown that using a hash table to look up an entry and $O(n)$ storage space, the point location query can be performed in constant time. Also for an environment where objects are of uniform size, this is a rather ideal algorithm and especially suitable for parallelization.

1.2.3 Responding to collisions

Once collisions are detected, the dynamic state of the colliding objects must be changed in order to avoid inter-penetration. In the case of rigid body collision, where colliding objects should never penetrate each other, the change depends on the type of collision and the physical and dynamic parameters of the colliding objects. In the case of non-rigid bodies (i.e., objects that are capable of deformation), different response schemes must be applied. Based on the type of collision in the virtual surgery environment object interactions, can be broadly categorized into the following cases:

- Rigid body to rigid body: This type of interaction occurs when rigid bodies like instruments collide with other instruments or bones. Here, the interacting objects are both non-deformable and should provide force feedback to the user.
- Deformable to rigid body: Such interactions are very common in interactive simulations, where the user manipulates organs, tissues etc. using instruments. The instrument – tissue collision should create a physically realistic response on the interacting tissues. The response should depend on parameters like force, penetration and movement vector.
- Deformable to deformable: This type of interaction occurs when different internal organs are pushed against each other. Showing proper response to such interactions is very difficult as both the objects in collision are capable of deformation. The response should consider surface characteristics like texture, friction, viscosity etc. and object property such as mass, density, geometric orientation etc.

In the following, literature review related to collision response is provided. Projection is one physically plausible method [44] for dealing with overlapping objects. The basic idea is to move the objects out of penetration using the smallest possible displacement. M. Moore et al. [33] describe a collision response method based on conservation of linear and angular momentum for the colliding bodies. This approach is very simple and works only for rigid bodies. However, the body is not considered to be made up of discrete particles. Therefore, this approach cannot be used in our case. D. Baraff et al. [34] proposed an idea for collision response for animated cloth simulation, which couples a technique for enforcing constraints on individual cloth particles with an implicit integration method. This method takes the stretch, shear and bending forces into consideration. It also considers the damping and constraint forces. The force on the particle is a summation of all these effects. But this method cannot be implemented in real-time and therefore is not a candidate for us. Another way to deal with collision is using impulse-based [35] method for rigid bodies. The impulse-based method is one of the oldest and simplest methods for collision response. It uses instantaneous impulses (change in velocity) to prevent the objects from interpenetrating. Since this method is not very accurate, it cannot be used for deformation of colliding objects.

1.3 Organization of the Thesis

Chapter 1 has given a brief introduction of Virtual Reality, the need for VR based surgical simulators for training purposes and the different technical issues encountered while implementing a surgical simulator. Chapter 2 explains the Inguinal Hernia condition and, the repair operation procedure which is being simulated. Chapter 3 describes the theoretical background of the various components of the existing system implemented in the Virtual Environment Laboratory, the underlying tessellation algorithm and the OHC data structure which is used for detecting collisions. Chapter 4 discusses the changes made in the OHC algorithm for the detection of collisions along with the approach for calculating the penetration depth. Chapter 5 shows the results and discusses future work that can be done on this topic.

CHAPTER 2

LAPAROSCOPIC HERNIA SURGERY

2.1 Inguinal Hernia

Inguinal Hernia is the protrusion of the abdominal cavity contents through the inguinal canal as shown in Figure 2.1 (a). It is a very common condition (It is estimated that 7% of the population will develop an abdominal wall hernia), and its repair is one of the most frequently performed surgical procedures. There are two types of inguinal hernias, direct and indirect. Direct inguinal hernia occurs when abdominal contents herniate through a weak point in the fascia of the abdominal wall and into the inguinal canal. Indirect inguinal hernia occurs when abdominal contents protrude through the deep inguinal ring.

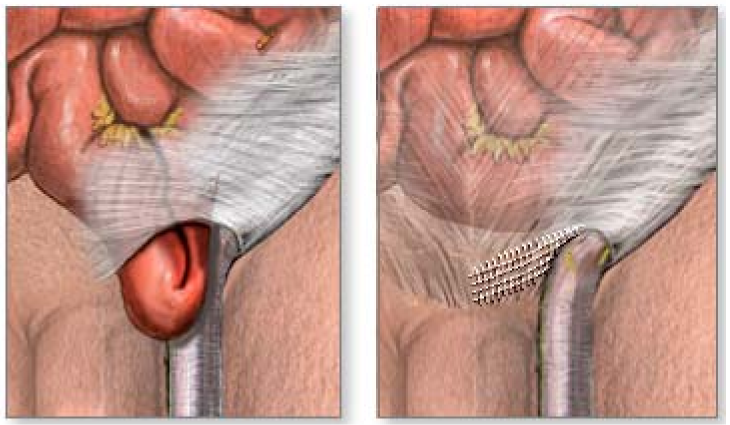


Figure 2.1. (a) Intestine passes into the scrotum or groin, (b) after surgery [40]

As the hernia progresses, contents of the abdominal cavity, such as the intestines, can descend into the inguinal canal and run the risk of being pinched within the hernia, causing intestinal obstruction. This condition can be often painful and is visible as a bulge in the groin area.

2.2 Laparoscopy and Inguinal Herniorrhaphy

Surgical correction of inguinal hernia is called herniorrhaphy or hernioplasty, which can be performed either as an open procedure or as a minimally invasive procedure (Laparoscopy). In the past decade, Laparoscopy has seen a strong acceptance over more traditional surgical techniques. Its main advantage is to avoid the traumatizing link to the opening of the patient's body. In the case of laparoscopic surgery, a video camera and few surgical instruments are introduced inside the abdomen through small incisions. The technique has the advantage of being less invasive, therefore shortening the stay of the patient at the hospital. It generally offers more rapid recovery for the patient, less postoperative pain, and a quicker return to work and normal activity.

In laparoscopic hernia surgery, a telescope attached to a camera is inserted through a small incision made under the patient's belly button. Two other small cuts are made (each no larger than the diameter of a pencil eraser) in the lower abdomen. The hernia defect is reinforced with a 'mesh' (synthetic material) and secured in position with stitches/staples/titanium tacks or tissue glue. Figure 2.2 shows the mesh placement and the incisions made for the procedure.

Laparoscopic Inguinal repair is a relatively complicated procedure to fix tears in the abdominal wall using small incisions, a patch, and special cameras to view inside the body.

2.3 Need for the VR based Simulator

The minimally invasive surgery requires specific training due to the difficulty in moving a three dimensional tool by looking at a two dimensional video image, which creates a problem of hand-eye coordination. The medical schools endow the required skills to become a physician, but when it comes to surgery, the greatest teacher is experience.

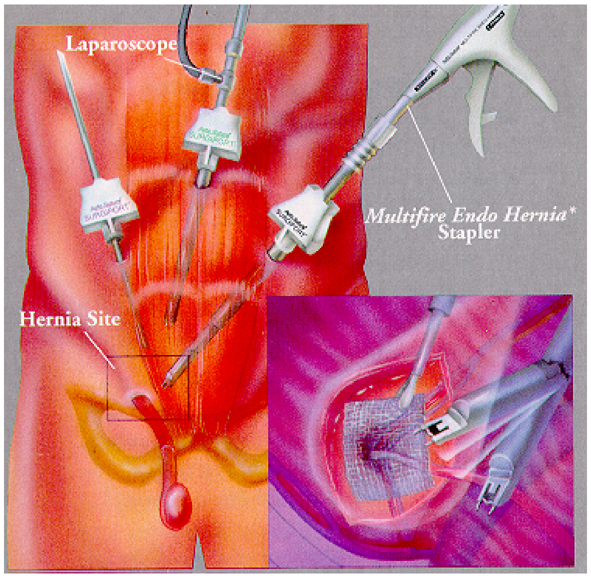


Figure 2.2. Laparoscopic Inguinal Hernia repair operation [43]

Each year about 600,000 hernia repair operations are performed in the United States. Until recently, all were performed as traditional, “open” procedures requiring a large incision in the lower abdomen. Because laparoscopy requires extensive and specialized training, only a small percentage of surgeons throughout the country are qualified to perform these procedures. A Laparoscopic surgical simulator can be used to train surgical residents and practicing surgeons to facilitate the development of the required psychomotor skills and dexterity. These trainers also have the advantage of reducing the learning curve besides ensuring patient safety. But there are limitations of providing training for such procedures. Because most of the trainers do not simulate the real surgery environment, they might be trained on animals or cadavers. Mannequin based simulators have lots of advantages since they are bloodless and provide visual simulation somewhat close to the actual surgery. Some instructor stations [14] provide an interactive graphics interface for trainee and a means to record the training exercises, evaluate them with various performance metrics and compare their simulations with reference to any other simulation runs. Considering all these advantages it is very useful to develop a VR based surgical simulator for training purpose.

CHAPTER 3

BACKGROUND

3.1 Major components of the Surgical Simulator

The major components of the surgical simulator designed and developed at the Virtual Environment Laboratory, UTA is shown in Figure 3.1. They are the following:

- Offline Processing Module: The offline processing module is responsible for the generation and visualization of the geometric database. This is a preprocessing step required for the modeling of a virtual patient. The block is detailed in section 3.2.
- Real-Time Module: This module has two sub-blocks - collision detection and deformation. The real-time collision detection algorithm [15] is a core block requiring intensive computation and interaction with physical parameters needed for simulation.
- Haptic Module: This block is designed to provide the user with the tactile force feedback at an update rate of 1 KHz. This module is explained in more detail in section 3.4.
- Graphical and Special Effects Module: This module provides a realistically rendered view of the geometrical objects included in the surgical environment. It also consists of a special FX module [12] for simulating special visual effects

that occur during virtual surgery such as bleeding, cauterization, irrigation, suction, stapling etc. A major task of stapling the mesh [13] in a virtual laparoscopic inguinal hernia surgery is also implemented with collision detection and limited collision response methods. The graphics simulation loop will run at 30 Hz.

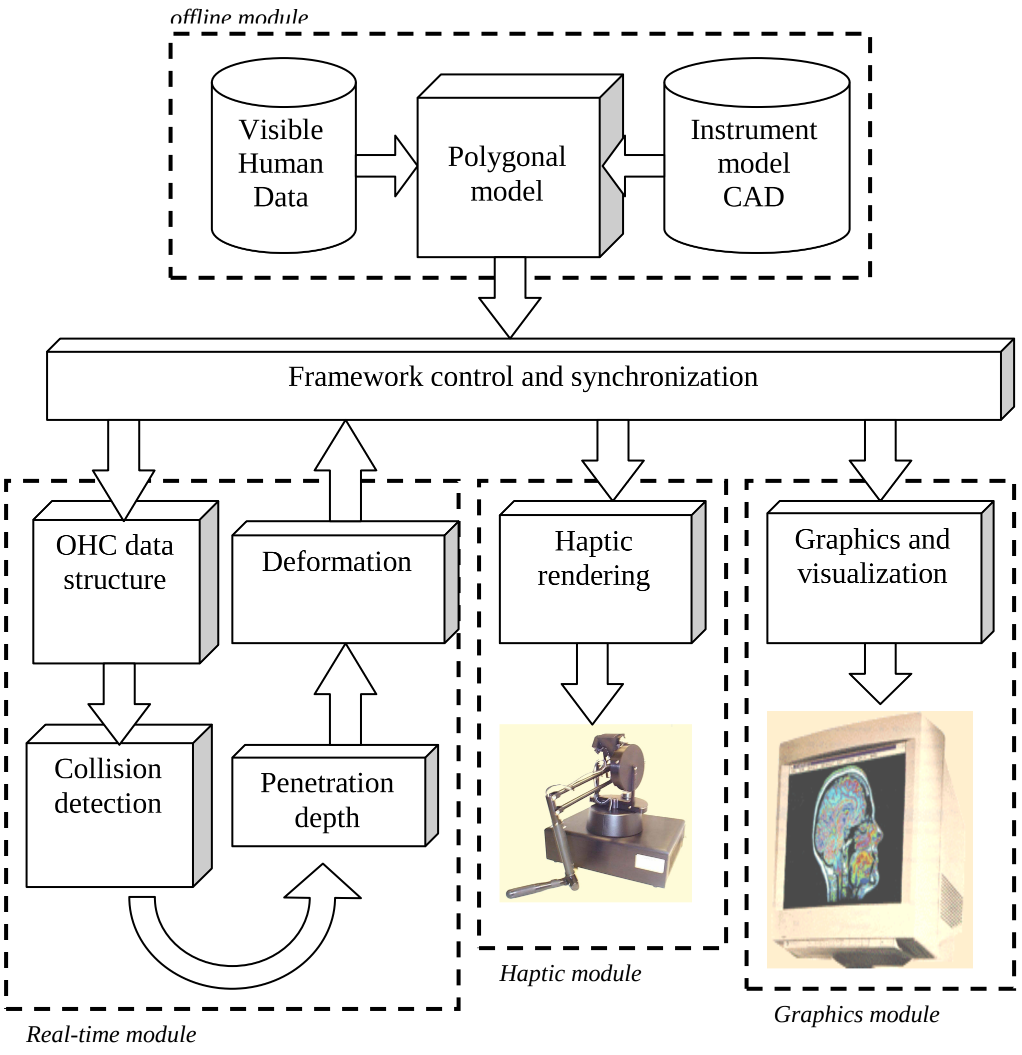


Figure 3.1. Major components of a surgical simulator

3.2 Model Representation

Geometric modeling is an offline preprocessing step applied before real time simulation starts. Advanced medical techniques frequently require geometric representation, either simulated or physical, which can be used for visualization of the organs for diagnosis, education, guided surgery and other purposes. The Visible Human Project (VHP) [40] has provided the input images to create numerically consistent, quantitative data representations of anatomical geometries. A Visible Human Data slice obtained from National Institute of Health is shown in Figure 3.2.

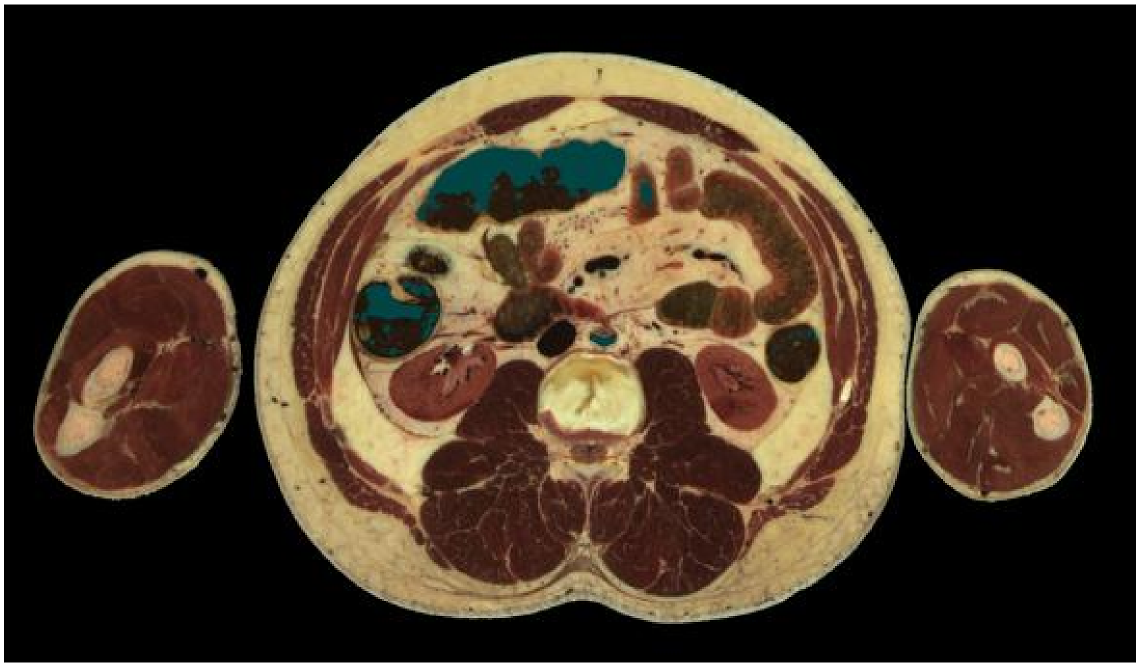


Figure 3.2. Visual Human Data (VHD) slice

To construct 3D boundaries of individual organs or tissues from medical images, one segments a set of medical images such as cryosection images, MRI or CT images. Surface boundary meshes for organs of interest are created from these segmented outlines using the Marching cubes [9] algorithm. Another modeling method is to manually create models by Computer Aided Design (CAD) which is an artistic and time consuming technique. Though it is flexible, it does not represent the normal or diseased condition and lacks anatomical fidelity. A realistic texture image [10] is added onto the models. The 3D models of all instruments [11] needed for surgery simulation are generated using a modeling software product called, 3D Studio Max.

3.2.1 Geometric data format

The algorithms developed in this thesis use as input geometric models. Therefore it is appropriate to discuss the format of such data. The geometric models are represented in a standard file format called VRML (virtual reality modeling language – now called X3D) which is loaded at the start of the simulation. Thereafter, the predefined static data are refreshed in real-time, according to the instrument manipulation and collision response. VRML is a text file format, where vertices and edges for a 3D polygon can be specified along with the surface color, texture, reflectance, transparency and so on. VRML files are commonly called “worlds” and have the *.wrl extension.

VRML describes 3D models in the form of nestable "nodes" [41]. Nodes generally define 3D physical descriptions that may be made up of 3D primitives, such as spheres, cuboids, cones and cylinders, or of complex polyhedra made up of polygon facets. In addition to these form descriptions, nodes can also define materials, colors, texture maps, lighting, shape transformations and viewing criteria. The following is a simple example code for three polygonal walls clearly showing the point vertices and the polygons formed by joining the vertices.

```
#VRML V1.0 ascii

Separator {
    PointLight {
        location 1 1 1
    }
    Coordinate3 {
        point [ -1  0 -1,#p0 these are vertex primitives
x,y,z
                -1  0  1,#p1
                 1  0  1,#p2
                 1  0 -1,#p3
                -1  2 -1,#p4
                -1  2  1,#p5
                 1  2  1,#p6
                 1  2 -1]#p7
    }
    IndexedFaceSet {
        coordIndex [ 0,1,2,3,#The floor
                         -1,#end of coord for the floor
                    0,4,5,1,#The back side
                         -1,#end of coords for back side
                    0,3,7,4,#the left side
                         -1]#end of coords for left side
    }
}
```

Figure 3.3. Simple code in VRML format to represent the polygons in Figure 3.4

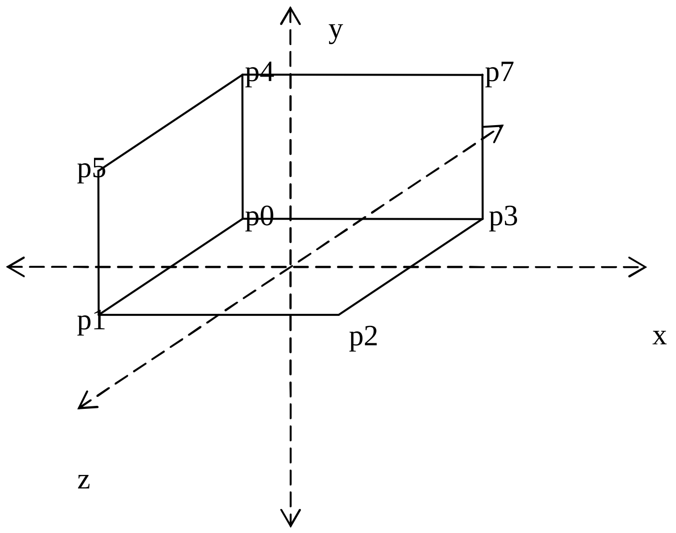


Figure 3.4. VRML sample example showing the 3 polygonal walls

3.3 OHC Data Structure and Algorithm

The OHC algorithm [16] is based on spatial tessellation. The term tessellation in computer graphics means dividing up the 3D space into uniform volumes so that the whole environment is represented by a collection of such elemental volumes. The tessellation of virtual environment results in three levels of information:

1. Cell level – Once virtual environment is tessellated, each of the uniform volumes is given a unique index which represents a specific location. The cell level information provides this index.
2. Object level – An object is defined as the geometric model representing an entity in a virtual environment. Each object is identified by a unique identifier. The object level information provides the identifier of the object occupying the specific cell volume.
3. Primitive level – Primitives are the smallest geometric elements such as a vertex or a polygon used to represent the object. In our case, the polygon is a triangle. The primitive level information contains the list of those triangle identifiers of the objects that occupy the corresponding cell volume.

The OHC algorithm utilizes this hierarchy and builds a data structure to analyze the virtual environment. The occupancy information is maintained in data containers associated with each level, so that the OHC is cascaded into three layers (COP - Cell Object Primitive) as shown in Figure 3.5. Thus, for a given cell index, we can easily determine all the triangle primitives of all the objects that occupy the cell space. The organization of data into these layers allows decreasing the complexity of the narrow phase detection.

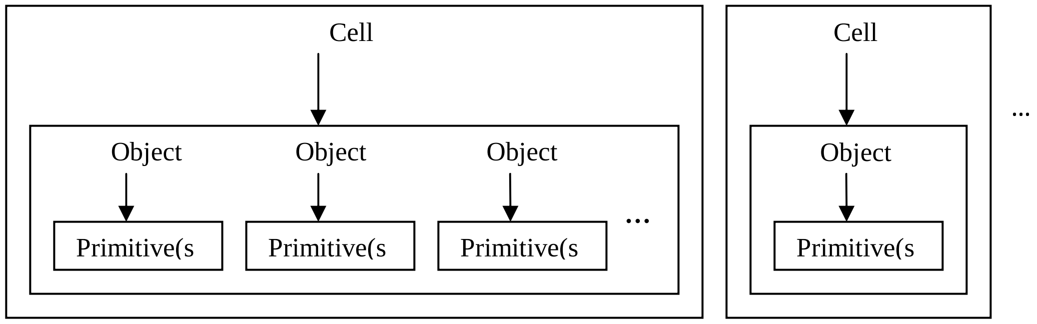


Figure 3.5. COP cascades in OHC data structure [16]

The spatial tessellation approach has been tested in several applications [6], [8] and [9]. Spatial tessellation requires data containers to store occupancy information, including the cell indices, object identifiers and primitive indices. The OHC algorithm was implemented using a combination of dictionary structures [11] such as a hash table and binary search tree, which are found to improve the efficiency of data storage and access with their dynamic memory allocation and fast search mechanisms.

The first and second layers of the COP data structure are dictionary structure types; a hash table is preferred in the first layer for its efficiency in querying an enormous number of entries. The second layer has binary search trees as data structure since it is very unlikely for one cell to contain a large number of objects in the case of surgical simulation. The third layer is a simple linear structure such as a vector or linked list.

Spatial tessellation is performed by a process called rasterization, which extrapolates the coordinates within a space occupied by the primitive and locates the cells containing these extrapolated coordinates by the coordinate hashing function (please see [16] for more details). The rasterization provides the set of cell indices which will completely include the primitives. The coordinate hash function is used for mapping a 3-dimensional coordinate point into a positive integer, which is used as the cell index [16]. The coordinate hashing ensures that the coordinates within the same cell space are mapped to the same index. The OHC algorithm for detecting collision proceeds in three steps viz. Construction, Instruction and Intersection.

3.3.1 OHC Construction

The OHC construction phase fills the three-layer data structure with cell, object and primitive information which are obtained by rasterizing all the objects in a virtual environment into the data structure. The chosen rasterization resolution ensures that a primitive is not in the same cell more than once, i.e. the primitive indices stored in the third layer are distinct. New entries of cells and objects are instantiated only if queries show that they do not exist. At the end of the construction phase, the cells can be classified into three types: Empty cells, Solo cells and Potential collision cells. Empty cells are those cells which are not occupied by any object. Empty cells are not instantiated in the data structure. Solo cells are those which contain primitives from only one object. Potential collision cells are the cells of interest to us as they contain primitives from more than one object, which may or may not be colliding. This is an important condition to be tested since objects lying within the same cell may or may not be intersecting. Figure 3.6 shows a 2D view of a typical tessellated environment, where cells numbered 3, 4, 16 and 17 are potential collision cells and those numbered 2, 5, 6, 11 are examples of solo cells. In the case of potential collision cells, we can clearly have cases where objects occupy the same cell but are not interacting (cell 17).

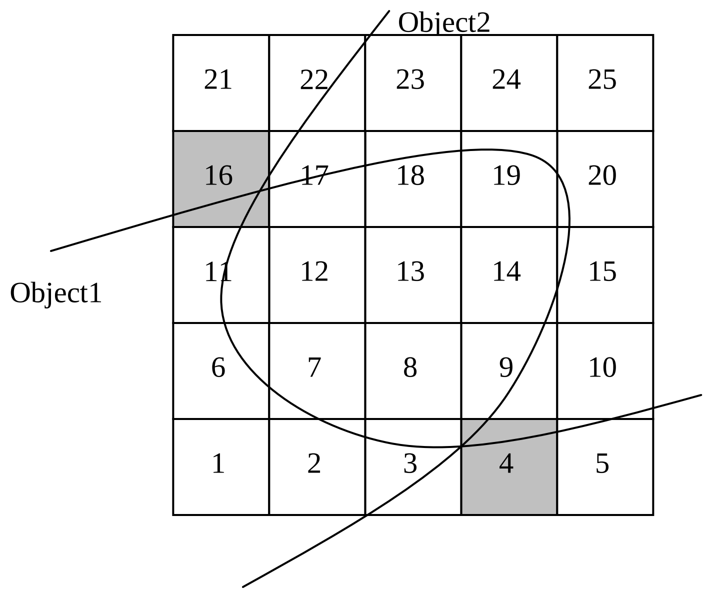
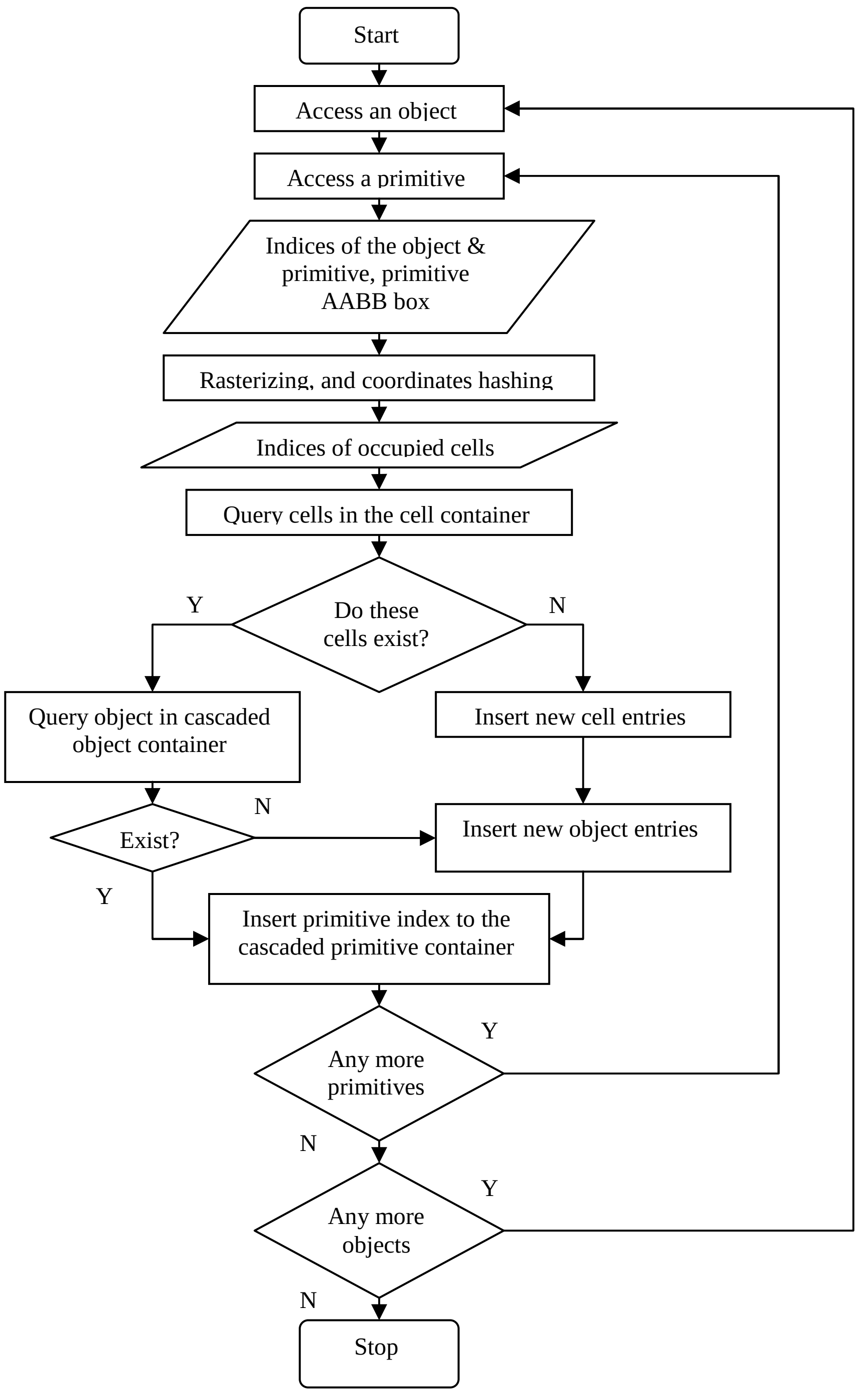


Figure 3.6. Examples of solo cells, potential collision cells and empty cells

In the OHC algorithm, only the potential collision cells need to be instantiated in the data structure, if self collision detection is ignored. Self collision detection is generally ignored in most applications since it requires extensive computation with a very little useful result. Figure 3.7 shows the OHC construction. Rasterizing an object and instantiating the results in a data structure are two sequential steps in the OHC construction. For the non self collision objects, they are first rasterized into cell indices, which are then saved in an object counting table as the indexing key. Thus, the object count of each cell is available after the first step, as shown in Figure 3.7 (b). If the object index in the entry is different from the current object, the counter is increased by one and the object index is updated. The object index ensures that the counter counts objects, but not primitives. This step can be performed as a prescreening process to eliminate the need for instantiating a solo cell since self collision detection is ignored. The detailed two pass rasterization is shown in Figure 3.8. The construction phase is followed by the instruction phase which returns all the potential collision cell indices.


Start
Access an object
Access a primitive
Indices of the object & primitive, primitive AABB box
Rasterizing, and coordinates hashing
Indices of occupied cells
Query cells in the cell container
Do these cells exist?
Y
N
Query object in cascaded object container
Insert new cell entries
Exist?
N
Y
Insert new object entries
Insert primitive index to the cascaded primitive container
Any more primitives
Y
N
Any more objects
Y
N
Stop

(a)

Cell Index	Object Counter	Object Index

(b)

Figure 3.7. (a) OHC Construction, (b) Object counting entry [16]

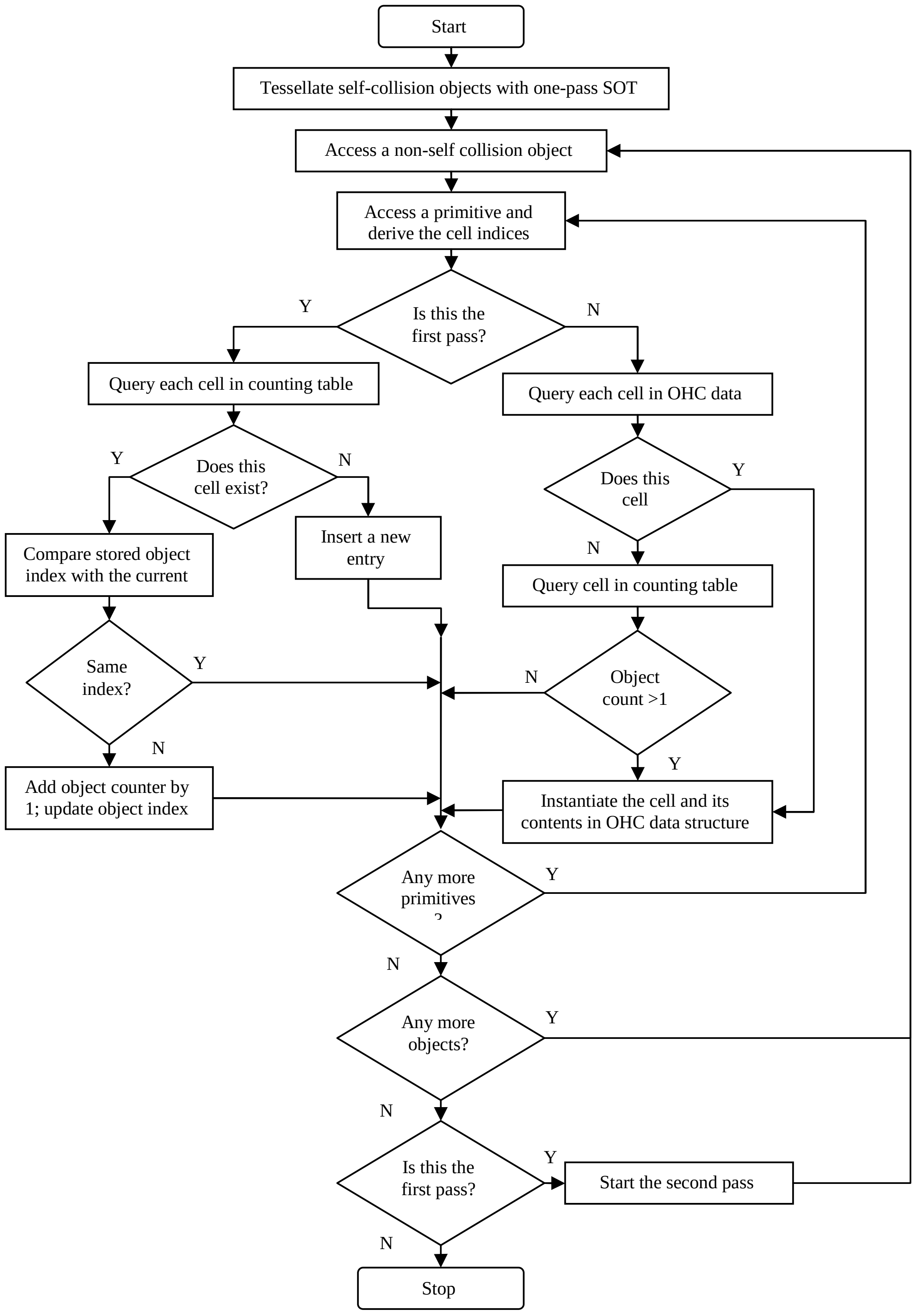

Start
Tessellate self-collision objects with one-pass SOT
Access a non-self collision object
Access a primitive and derive the cell indices
Is this the first pass?
Y
N
Query each cell in counting table
Query each cell in OHC data
Does this cell exist?
Y
N
Compare stored object index with the current
Insert a new entry
Does this cell
Y
N
Query cell in counting table
Same index?
Y
N
Object count >1
N
Y
Add object counter by 1; update object index
Instantiate the cell and its contents in OHC data structure
Any more primitives ?
Y
N
Any more objects?
Y
N
Is this the first pass?
Y
Start the second pass
N
Stop

Figure 3.8. Rasterization and construction [16]

3.3.2 Instruction and Intersection

The instruction phase performs the broad phase detection process of reducing the number of actual computation required to determine the exact contact points of collision. Thus, the instruction phase acts as a culling algorithm, refining the total list of primitives that need to undergo the computationally expensive intersection test for contact determination. The narrow phase is primitive to primitive intersection test, which is implemented using the triangle to triangle intersection test [17]. These intersection tests are implemented by traversing through the OHC structure without querying or additional rasterization, which locates the exact point of intersection for a given pair of triangles.

3.4 Haptic Module

Haptic technology refers to the technology which interfaces 3D object data with the user via the sense of touch by applying forces, vibrations and/or motion [39]. This mechanical stimulus can act as a very efficient feedback mechanism for controlling the moves of the user in the virtual environment. Haptic interface can be potentially useful for training on minimally invasive procedures and remote surgery using teleoperators.

This block provides a force-feedback effect for the practicing surgeon as he interacts with the anatomy. Surgical simulators with a haptic feedback requirement demand that their haptic rendering must be refreshed close to or above 1 KHz to keep the haptic state persistent and stable. The value is set by the somatosensory system threshold for humans. The module consists of two PHANToM haptic interface devices by SensAble Technologies Inc., which are connected at the end of each laparoscopic probe. Three small motors give force- feedback to the user by exerting pressure on the grip or thimble.

3.4.1 Ghost SDK

The SensAble Technologies Incorporated General Haptics Open Software Toolkit (GHOST SDK) is the C++ object oriented toolkit that represents the haptic environment as a hierarchical collection of geometric objects and spatial effects [39]. The GHOST SDK provides an abstraction that allows application developers to concentrate on the generation of haptic scenes, manipulation of the properties of the scene and objects within the scene, and control of the resulting effects on or by one or more haptic interaction devices. The GHOST API enables application developers to interact with haptic interaction devices and create haptic environments at the object or effects level. Using the GHOST SDK, developers can specify object geometry and properties, or global haptic effects, using a haptic scene graph. A scene graph is a hierarchical collection (tree) of nodes. The internal nodes of the tree provide a means for grouping objects, orienting and scaling the subtree relative to the parent node and adding dynamic properties to their subtrees. The terminal nodes (leaves) of the tree represent actual geometries or interfaces. Leaves also contain an orientation and scale relative to their parent nodes. The terminus of the haptic interaction device is represented as a point within the scene graph. The GHOST SDK automatically computes the interaction forces between this point and objects or effects within the scene, and sends forces to the haptic interaction device for display.

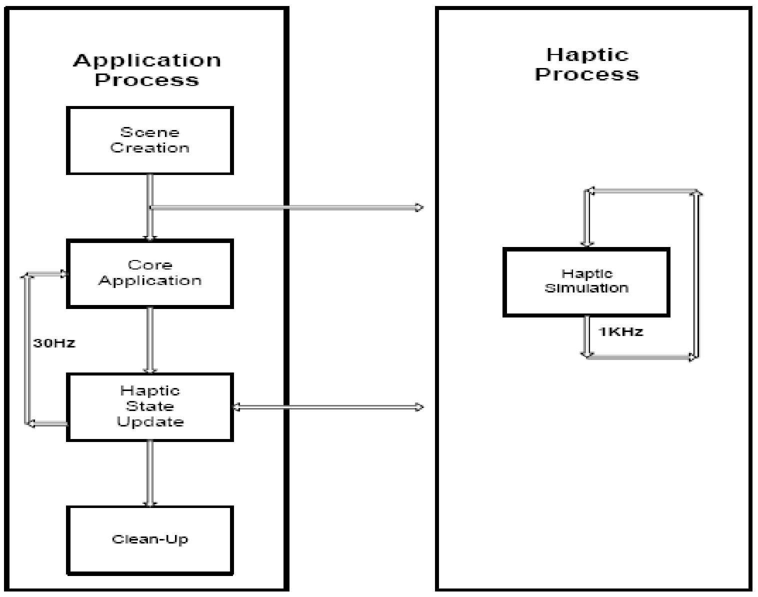


Figure 3.9. Typical Application using GHOST SDK [39]

Figure 3.9 shows the components and processes that a typical application using the GHOST SDK must have. The GHOST SDK must

- Create a haptic environment using a hierarchical haptic scene graph.
- Haptically render disparate geometric models within the same scene graph.
- Specify the surface properties (for example, compliance and friction) of the geometric models.
- Use behavioral nodes that can encapsulate either stereotypical behaviors or full freebody dynamics.
- Provide general support for the generation of haptic human-computer interfaces, including haptic manipulators for interacting with objects in the haptic scene using force feedback and spatial effects such as springs, impulses and vibrations.

- Perform application specific functions that include the generation and use of computer graphics.
- Be able to automatically parse and use the static geometry of VRML geometry files to generate haptic scene graphs.
- Perform clean up operations when the application ends.

3.5 Collision Response

The collision response section of the VR based simulator is implemented using the mass spring model for simulating the tissue deformation. The mass spring model is a particle simulation system, where each vertex of the object is given a particle property such as mass, position and velocity. The vertices are connected to their neighbors through springs to maintain their position, connectivity and orientation.

Despite the simplicity of particles and the well defined dynamics, which allow them to be simulated very easily, they can be made to exhibit a wide range of interesting behavior. They can also be constructed easily and are much suitable for simulating deformable surface models. Besides these advantages, the relative computation cost is also low, which allows it to be used for real-time simulation. The following section discusses the mass spring model in greater detail [42].

3.5.1 Mass spring model

The deformation of objects in our system (MedVR) is implemented using a mass spring model (Figure 3.9), which is described in detail in [36].

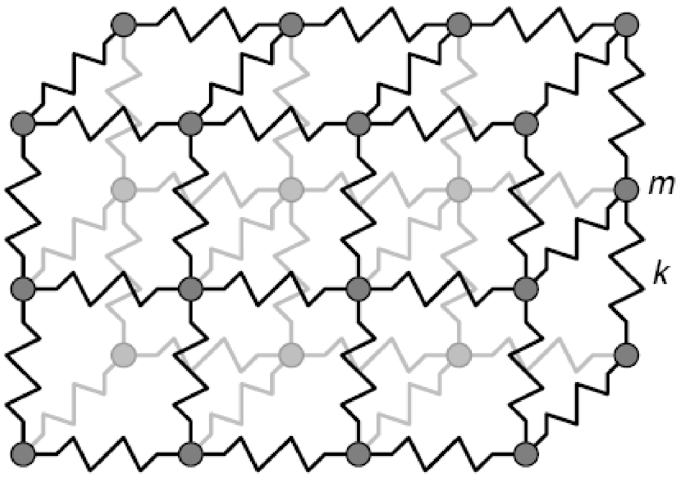


Figure 3.10. Mass Spring model [37]

CHAPTER 4

VISUAL DEFORMATION DUE TO COLLISION

4.1 The need to calculate penetration depth after collision

The current OHC algorithm provides exact collision points at the end of the intersection phase, which is performed for each triangle-triangle pair. This result cannot be used to produce an effective and continuous response in a virtual surgical environment, where a large number of object interactions occur. Moreover, since the penetration depth is not calculated, the response produced would be impulsive in nature. The object interactions can be categorized into two types based on the feedback they provide to the user. They are:

- Interactions which should provide haptic feedback: One example of such an interaction is instrument colliding with other objects. Here, since the instruments are handled by the user, the user should be given force feedback in addition to the visual feedback provided by the graphics module.
- Interactions that need not provide haptic feedback: Examples of this type of interactions are tissue pushed against another tissue or the mesh freely falling on an organ model. Since the user is not involved in such a collision, these interactions can be provided with the visual deformation alone.

In a surgery environment, where the user manipulates different organ models using instruments, providing tactile feedback is essential. But at the same time, the background interactions, which fall in the second category discussed above is also very important. Such interactions can be provided visual deformation without the need to calculate the force. Hence, the penetration depth for such interaction can be directly used to produce deformation by pushing the objects in direction opposite to the penetration by the distance which would resolve interpenetration.

It can be noticed that, detecting the actual contact points might not be of any further use by itself, if they cannot be used to produce realistic visual deformation after collision. In order to produce practical deformation response, the system should determine the magnitude and direction of penetration. In Figure 4.1, the volume of space shared by the colliding objects indicates the overlapping region and the largest distance between the primitives in the overlap region can be defined as the penetration depth.

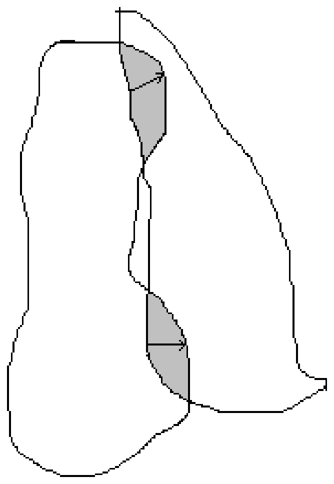


Figure 4.1. Overlap region and penetration depth

It may be recalled that the OHC algorithm proceeds by building the three layer data structure in the construction phase for the whole environment. It then performs the instruction phase (broad phase detection), where it determines the potential collision cells. Since the aim is not to calculate the contact points, we recognized that the intersection phase can be skipped. We need a new phase that would allow us to calculate the inter-penetration depth, which we realized can be obtained if we could determine the overlap between the intersection surfaces. We then introduced the Overlap phase, which should group the cell indices and, find the overlap region and penetration depth for an intersecting pair. The Overlap phase should effectively process n-body collisions, since there can be more than a pair of interacting objects in the potential collision cells.

The following steps then provide the modifications to the OHC algorithm. The Instruction phase provides the potential colliding cell indices to the Overlap phase. De-referencing these cell indices from the 3-layer data structure allows us to identify the objects involved in collision and, the exact list of all triangle primitives lying in the cells. This information is used by the Overlap phase to determine whether the objects are truly colliding. The algorithm proceeds to calculate the vertex primitives in the overlap region. The absence of these primitives validates whether collision has occurred. In case of a collision, the penetration depth is calculated. The resulting algorithm is coded in to a flowchart shown in Figure 4.2.

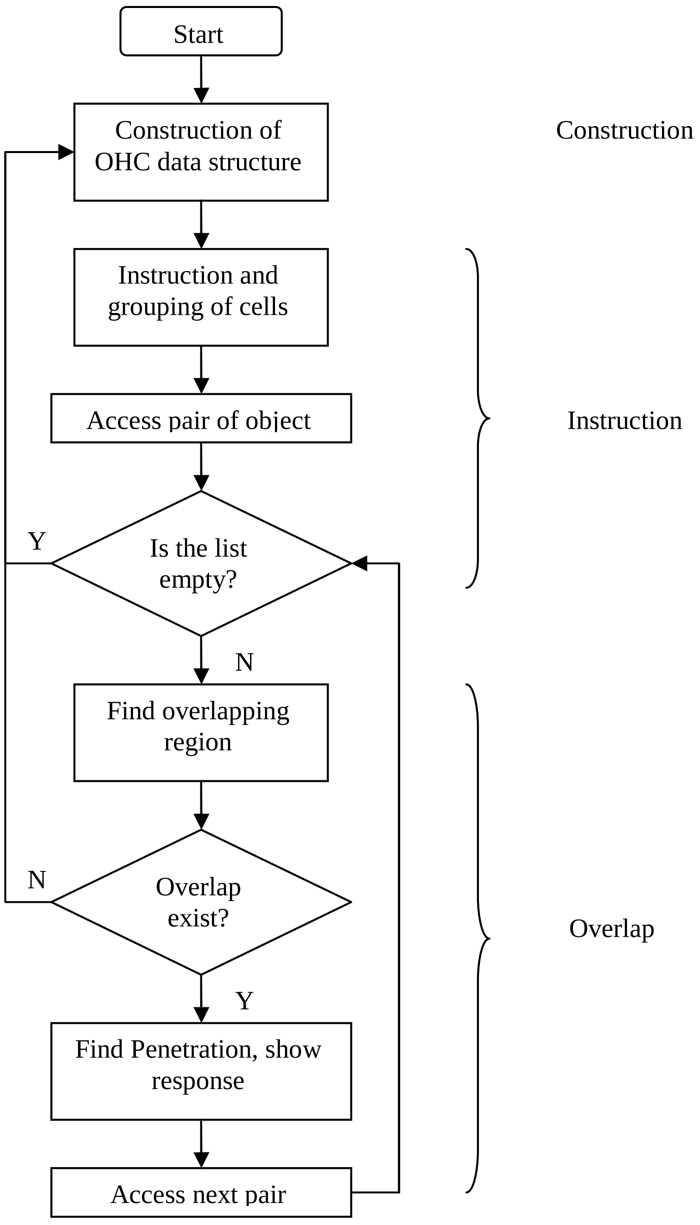


Figure 4.2. Construction, Instruction and Overlap phase

4.1.1 Overlap phase

The Overlap phase will return the list of vertex primitives which lie in the overlap region for the pair of potentially colliding objects. The overlapping vertex primitives correspond to the list of those vertices of the object that are included within the surface volume of its colliding pair. The flowchart for the function call FindOverlapRegion() is shown in Figure 4.3.

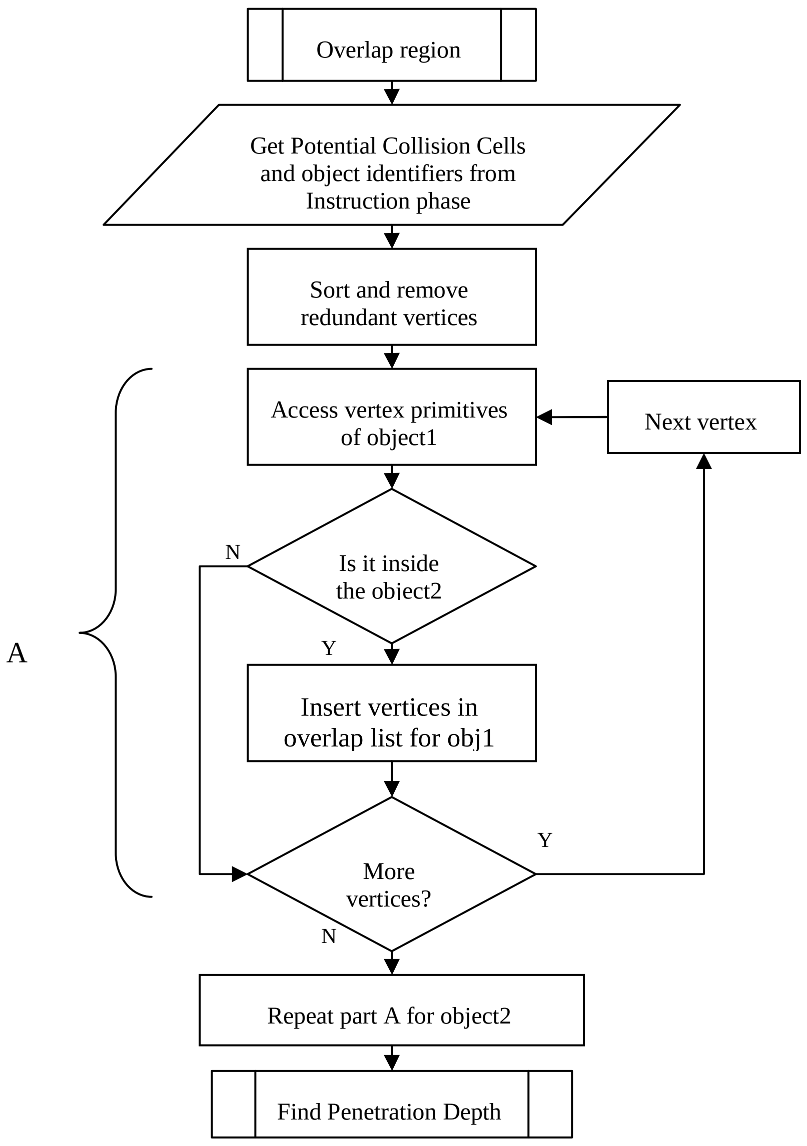

Overlap region
Get Potential Collision Cells and object identifiers from Instruction phase
Sort and remove redundant vertices
Access vertex primitives of object1
Next vertex
N
Is it inside the object2
Y
A
Insert vertices in overlap list for obj1
Y
More vertices?
N
Repeat part A for object2
Find Penetration Depth

Figure 4.3. Overlap phase

The cell indices can be de-referenced to obtain the list of triangle primitives for each object in the colliding pair. Each triangle primitive is an identifier, representing a triangle formed by joining the three vertices in a particular order. Thus, the triangle primitives can be used to obtain a list of all those vertices forming a part of the surface mesh for the object. Since triangular polygons share vertices in a mesh (Figure 4.4), there will be a large number of redundant vertices introduced in the list, while de-referencing the triangle primitives. The redundant vertices need to be removed in order to reduce the number of unnecessary computations and, to improve the performance. This is implemented by using a set of STL (Standard Template Library) algorithms.

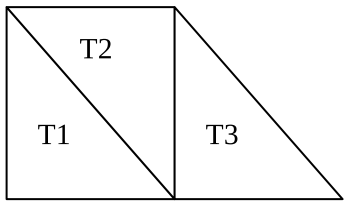


Figure 4.4. Redundant vertices in a polygonal mesh

Once the list of vertices is obtained, there is a need for further filtering of vertices, which lie inside the cell but are not included within the colliding object. Figure 4.5 shows two objects in a cell. The dots and the squares indicate vertices; it can be noticed that some dots are lying inside the cell but are not included in the object2, such vertices need to be removed from the vertices in the overlap region. This is performed by testing whether each of these points is inside or outside the polyhedron mesh of the colliding pair, utilizing the property of angle weighted pseudonormal proposed by Thürmer and Wüthrich [19] and independently by Séquin [20].

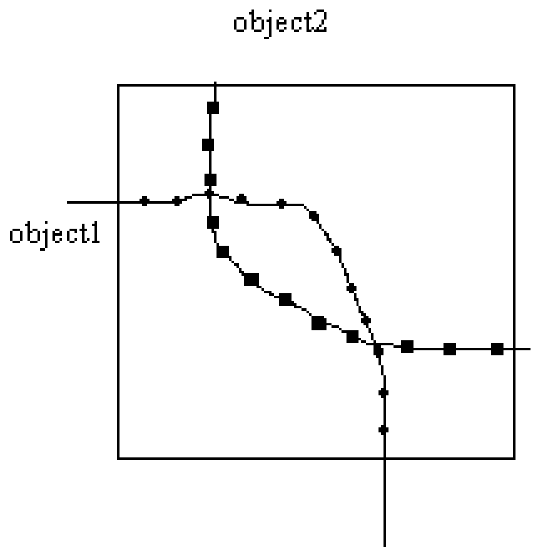


Figure 4.5. Vertices outside the overlap region

For objects with closed and smooth surfaces, the surface normal is an important tool for determining whether a given point is inside or not. However, there can be many cases where a mesh is not smooth everywhere and hence, does not have normals defined everywhere on the surface (i.e., the surface is discontinuous at edges and vertices). In such cases, it is possible to define pseudonormals, which possess some of the properties of normals. The angle weighted pseudonormal for a given point x ∈ M, where M denotes a triangle mesh, is defined as

$$n_\alpha = \frac{\sum_i \alpha_i n_i}{\left\| \sum_i \alpha_i n_i \right\|},$$

where *i* runs over the faces incident with x and α_i is the incident angle as shown in Figure 4.6.

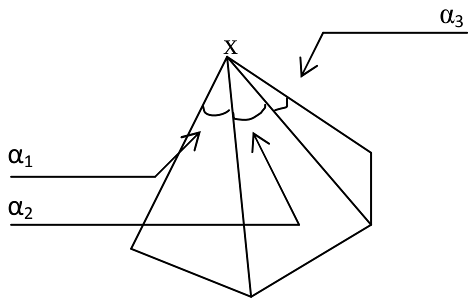


Figure 4.6. The incident angles in polyhedral [18]

A point is considered inside or outside by finding the closest point c, on the surface of the colliding object and taking the inner product of the surface normal at c with the vector between the given point p and c, i.e., r = p – c. The shortest distance obtained for each point is stored in a vector format for use at a later stage for penetration depth calculation. The angle weighted pseudonormal can be applied instead of the ordinary surface normal to accurately find whether the point is lying inside or outside the polyhedral, by using the following rule [18].

$$n_\alpha.(p-c) > 0 \quad if \;\; p \; outside\, surface$$
$$n_\alpha.(p-c) < 0 \quad if \;\; p \; inside\, surface$$
$$n_\alpha.(p-c) = 0 \quad if \;\; p \; on\, surface.$$

Thus, the overlap phase will return the list of only those vertex primitives that actually lie inside the other object volume. If this list is found to be empty, we can assume that there is no collision and the next pair of objects can be tested for overlap.

The closest point information for each vertex primitive is stored, which is used for calculating the penetration depth.

4.2 Penetration Depth

A very simple and physically plausible solution to the problem of object interpenetration is to move the interacting objects out of penetration using the shortest possible displacement. This minimum displacement required to move the objects out of penetration is called the penetration depth and the direction in which they need to be moved is called the penetration vector. The penetration depth calculation utilizes the closest point information calculated in the Overlap phase. Figure 4.7 below shows the shortest distance and the closest primitive in object1 for each primitive of object2. It also shows the largest of these shortest distances, which can be considered as the maximum penetration depth. The direction of this vector is taken as the penetration direction; all other shortest distances are projected onto this vector so as to obtain the actual penetration depth and direction at each primitive.

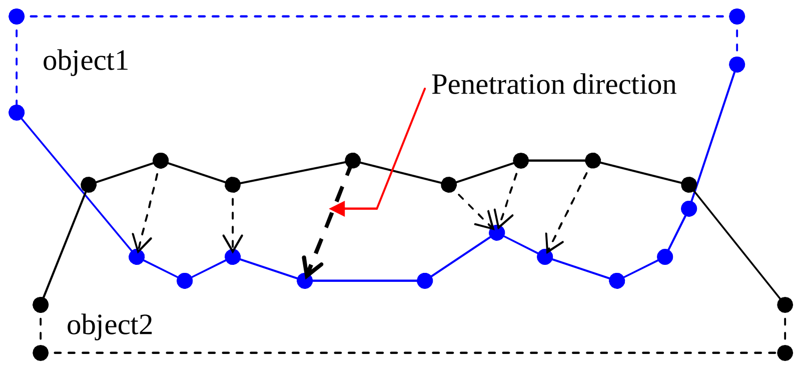


Figure 4.7. Penetration vector as the largest among the shortest distances

The projection based approach [44] for collision response is implemented by determining the shortest distance by which interpenetrating objects need to be moved in order to avoid any penetration. Object interactions in the surgery environment can be categorized as following:

- Rigid body to rigid body
- Deformable to rigid body
- Deformable to deformable

The rigid body collisions can be resolved by the haptic module, which loads the instruments and rigid tissues in the haptic scene. The haptic module will automatically calculate the interacting forces between the rigid bodies in the haptic scene. The graphics module should update the rigid body orientations based on the haptic scene response.

We are interested only in the latter two cases. In the case of deformable to rigid body collision, only the tissue will undergo deformation; hence the penetration depth should be used to displace only the tissue surface to resolve penetrations. Since the penetration depth at each vertex is known, we simply push the vertex in the direction opposite to the penetration direction calculated. The mass spring model for the deformation of organs will update the neighboring vertices of the organ so as to show that the deformation is not abrupt and creates a realistic response for collisions. In the case of deformable to deformable body interaction, the penetration depth should be divided in a ratio, which depends on the tissue characteristics. This ratio has been set to 0.5, assuming all the tissues are similar in surface characteristics. If the tissue characteristics are provided along with the various parameters like spring constant, density, surface friction etc., the simulation result would be more realistic.

4.3 Software Framework

The software framework for the surgical simulator was developed using Visual Studio .Net framework, with C++ object oriented programming. It communicates intensively with the operating system to improve performance. The system integration was accomplished by Dr. Yunhe Shen, who designed and implemented the framework [16]. The framework is based on object oriented programming class structure with separate functional classes defined for different functionalities. Some of the important class components are:

- CGeoModel: This class is responsible for loading the geometric models from the VRML text file format and storing them in different CGeoModel instances. The object instance of this class has parameters for defining the deformable data in space. Different organ specific parameters such as texture and tissue properties can be stored at this level. This class can be modified to implement the tetrahedral geometric organ models.
- CDeformation: This class defines a separate thread for deformable modeling of tissues. The mass spring model for each organ is built by the particle system simulation, where each primitive of the object is given a specific mass and connected to other primitives via springs. The parameters of these springs can be adjusted to vary the stiffness of the model.

- CGraphics: This class is responsible for the initialization of graphics and visualization parameters. The structure has calls to OpenGL, which is a standard graphics library that does real-time texture rendering and special effects.
- CHaptic: The Haptics class checks whether the haptics module is enabled or not. It is also responsible for creating the haptic scene graph, which automatically does the force calculation to provide a tactile feedback to the user.
- CShare: This class is responsible for synchronization and creating and updating the list of tissues and instruments in the surgery environment.
- CMI: The module initialization is responsible for interfacing, initialization and enabling of different modules. Different modules or classes require different amount of memory and processing power. To achieve this, different threads are given different priorities. There are four main threads running in the system:
    - Haptic device thread
    - Deformation thread
    - Collision Detection and Response thread
    - Real-time Graphics thread

  The graphics thread has a lower priority than the operating system since it requires lower update rates, while the remaining three threads have priority equal to the operating system, since they are time critical.

4.3.1 Collision detection and response thread

The collision detection and response thread is implemented in a separate class called COMap, which contains functions for creating object and instrument instances, filling the COP data structure and object counter. As part of this new work, new function routines were added in order to include new functionalities to the existing system. These functions include FindOverlapReg() for returning the list of primitives in the overlap region, FindPenetration() for determining the penetration vector and ShowResponse() for deforming the model geometries to reflect the deformation in the surface.

CHAPTER 5

RESULT AND FUTURE WORK

5.1 Simulation Result

The surgical simulator called MedVR is a Windows based application software developed on Microsoft .NET framework, with source code written in VC++, utilizing various libraries like OpenGL, GLUT and MFC. The specification of the PC used for simulation is as follows:

- Intel ® Xeon^{TM} dual CPU 2.8 GHz
- 1 GB RAM
- Microsoft Windows 2000 SP4
- Radeon 9700 Graphics cards – 2

The proposed algorithm is tested by loading various virtual objects in the surgery environment, and comparing the result for performance evaluation. The performance evaluation is accomplished with the help of timing plots, which calculate the time required to determine the penetration depth and provide response by deforming the tissues. Figure 5.1 below shows the total time taken for detecting collisions and providing corresponding response. The performance monitor class functions allow marking the time instance when a collision instance is detected and measure the time till the end of ShowResponse() routine. The average time taken in this plot is found to be in the range of 2.303 milliseconds. It can also be noticed from the plot that initial superficial collisions take less time as compared to when the interpenetration depth increases. But it is observed that the time taken for collision response is reasonably good for real-time simulation and haptic rendering.

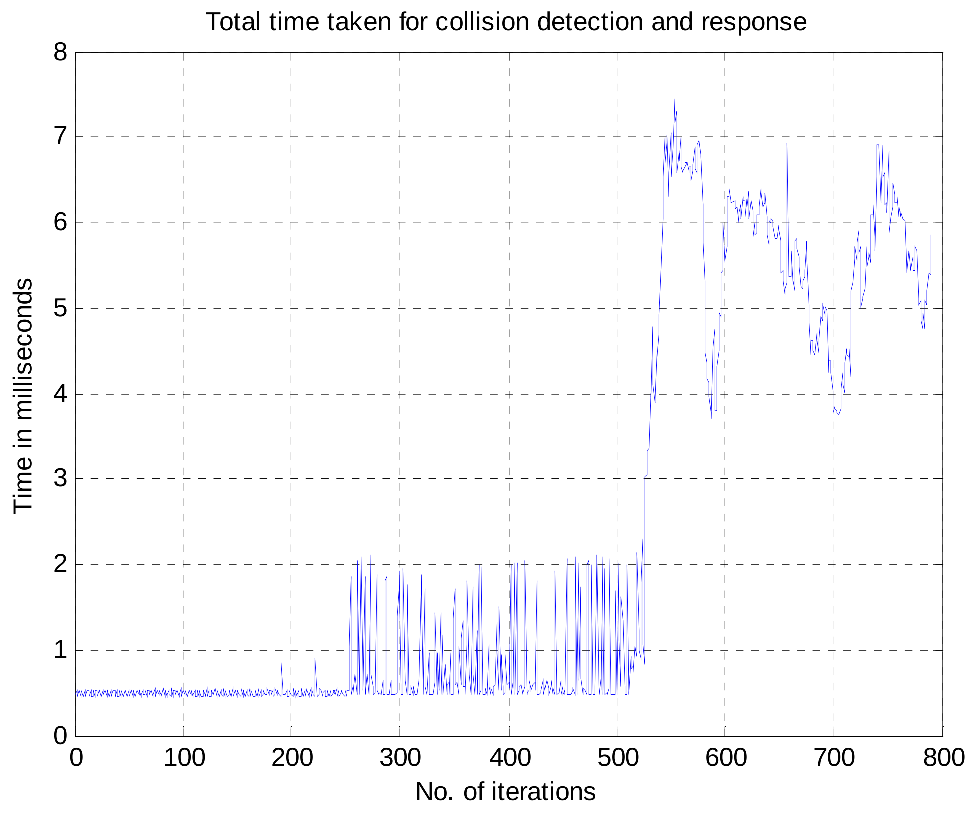


Figure 5.1. Total time taken for collision detection and response

Figure 5.2 below shows the time taken by the collision response routine. The average time for collision response is measured to be 1.884 milliseconds. The collision detection algorithm is based on sorting and retrieving of data from the data structures stored in STL format. One of the limitations of using STL is that, it consumes much time for accessing of data. In order to improve the timing response, more efficient data structures can be implemented.

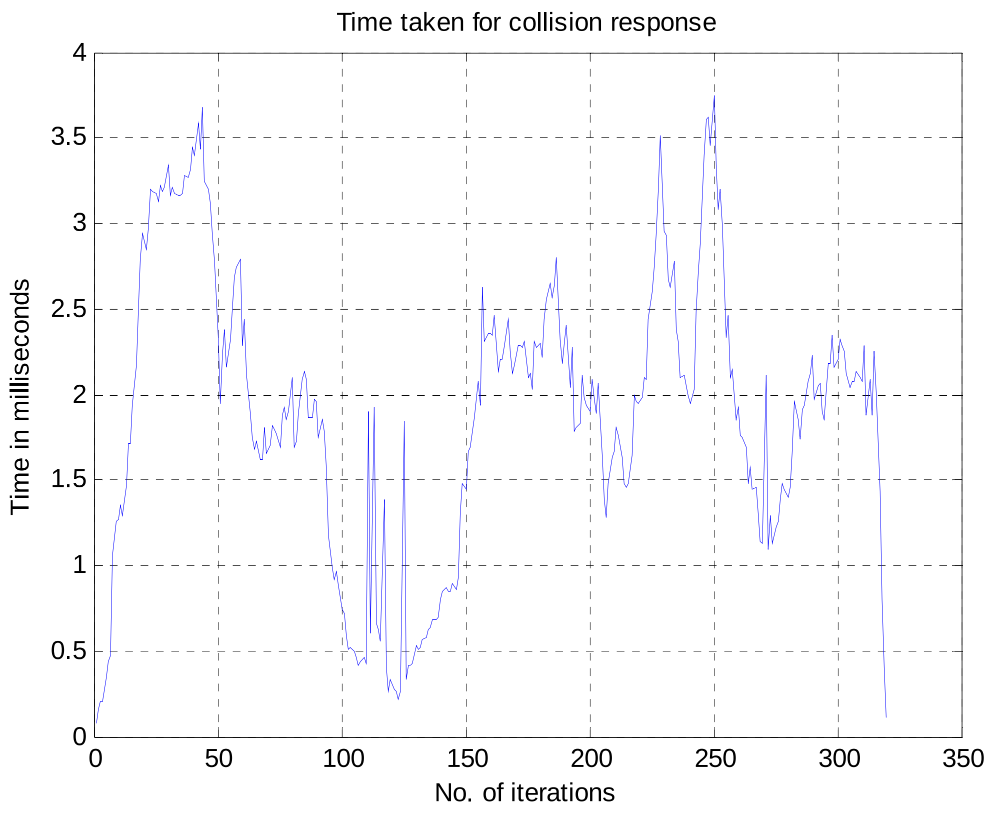


Figure 5.2. Time taken by collision response routine

Figure 5.3 shows graphical response in the case of deformable to deformable model interaction. Here one of the deformable models is assumed to be stiffer and hence the kidney, which is softer, is deformed more. The polygonal mesh structure without coloring applied is also shown. Figure 5.4 shows the graphical response for a rigid body to deformable body collision; texture mapping is enabled to provide realism in the virtual objects.

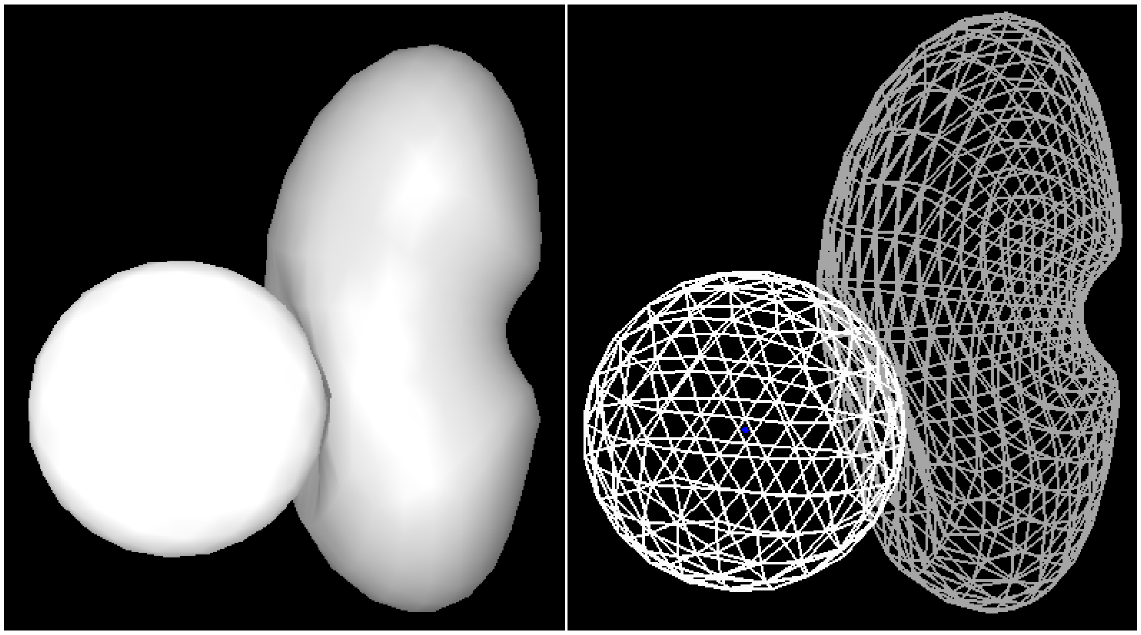


Figure 5.3. Demonstration of the deformation in tissue-tissue interaction

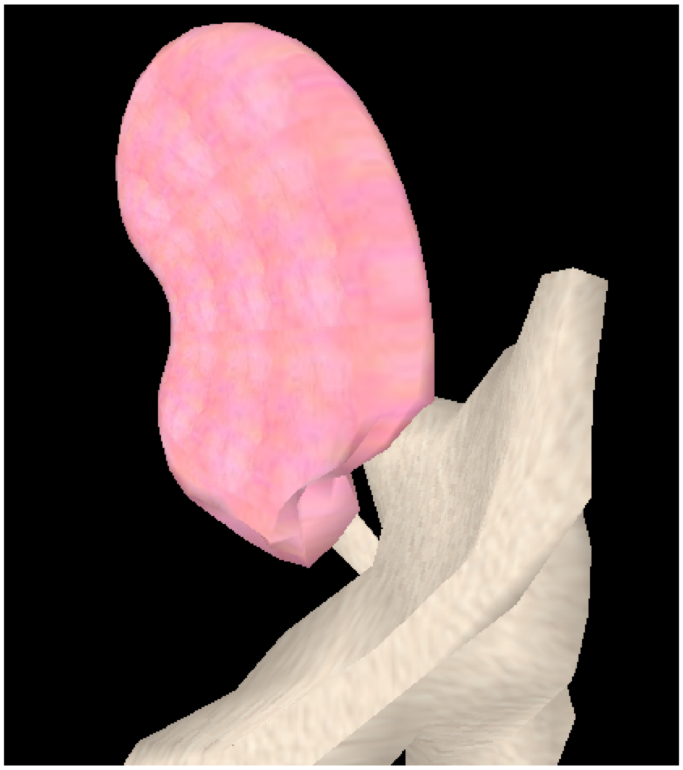


Figure 5.4. Demonstration of the deformation for rigid body to tissue collision

5.2 Conclusion

In this thesis, a novel approach for detection of collision and calculating penetration depth from overlap region in 3D space is introduced. The algorithm is restricted to smooth deformable to deformable objects and deformable to smooth rigid bodies. The algorithm is implemented on the VR based surgical simulator at VEL called MedVR and its performance evaluation indicates that, this technique can be used to detect collisions and provide visual response in real-time virtual surgical environments. The thesis has also been able to improve the functional blocks of the VR based simulator in our lab. The issue of tissue to tissue and, tissue to rigid body collision is resolved. The issue of mesh draping over other tissue organs is also resolved. However, this approach cannot be used for detecting and providing response to collisions caused by instrument interaction that have sharp edges.

The timing diagrams indicate that this approach can be used for real-time simulation of virtual surgery.

5.3 Future Work

Even though simulation results are in real-time, the performance of the system can be improved by optimizing the algorithm. Some critical time is wasted for data accessing. This might be avoided using better data structures. The algorithm could be extended to handle penetration depth calculation for the instrument collision case.

The surface geometric model should be replaced by tetrahedral models. This will invalidate the need of inserting fixed nodes, which are otherwise required to stabilize the surface models. The MedVR system in the lab can be upgraded to use 6 DOF haptic devices to provide twist, constraints and torques.

REFERENCES

[1] R. Woodcock, M. Morrison and Y. Attikiouzel “Development of a Virtual Surgery Environment”, *Third Australian and New Zealand Conference on Intelligent Information Systems, Perth, IEEE Australia and New Zealand Council, pp. 30-35, 1995*

[2] K. Chung and W. Wang “Discrete Moving Frames for Sweep Surface Modeling”, *Proceedings of Pacific Graphics’96, 19-22, August 1996.*

[3] J. Klosowski, M. Held, J. Mitchell, H. Sowizral and K. Zikan “Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs”, *IEEE Transactions on Visualization and Computer Graphics, vol. 4(1), pp. 21-36, 1998.*

[4] F. Ganovelli, J. Dingliana and O. Sullivan “Bucket-Tree: Improving Detecting Between Deformable Objects”, *Proceedings of Spring Conference in Computer Graphics, Bratislava, 2000.*

[5] M. Lin “Efficient Collision Detection for Animation and Robotics”, *PhD Dissertation, University of California, Berkeley, USA, 1993.*

[6] B. Mirtich “V-Clip: Fast and Robust Polyhedral Collision Detection”, *ACM Transactions on Graphics, vol. 17(3), pp. 177-208, 1998.*

[7] J. Cohen, M. Lin, D. Monacha and M. Ponamgi “I-Collide: an Interactive and Exact Collision Detection System for Large-Scale Environments”, *Proceedings of ACM interactive 3D graphics in proceedings, pp. 189-196, 1995.*

[8] G. Bergen “A Fast and Robust GJK Implementation for Collision Detection of Convex Objects”, *Journal of Graphics Tools, vol. 4(2), 1999.*

[9] W. Lorrenson and H. Clin “Marching Cubes: A high resolution 3D surface construction algorithm”, *Computer Graphics, vol. 21, no. 4, pp. 163-169, July 1987.*

[10] V. Gupta “Extraction of realistic anatomical texture from visual human data for laparoscopic herniorraphy”, *Master’s Thesis, The University of Texas at Arlington 2003.*

[11] R. Naidu “Creation of Static and Dynamic models of Instruments for a Virtual reality trainer for Laparoscopic surgery”, *Master’s Thesis, The University of Texas at Arlington*, 2002.

[12] L.Raghupathi “Simulation of bleeding and other special effects for virtual Laparoscopic surgery”, *Master’s Thesis, The University of Texas at Arlington, 2002.*

[13] G. Gopalakrishnan “StapSim: Virtual reality based stapling simulation for Laparoscopic herniorraphy”, *Master’s Thesis, The University of Texas at Arlington, 2003.*

[14] A. Gande “Instructor Station for Virtual Laparoscopic Surgery”, *Master’s Thesis, The University of Texas at Arlington, 2003.*

[15] Y. Shen, V. Devarajan and R. Eberhart “Haptic Herniorrhaphy Simulation with Robust and Fast Collision Detection Algorithm”, *The Proceedings of Medicine Meets Virtual Reality, Long Beach, CA, pp. 458-464, January 2005.*

[16] Y. Shen "Real Time Collision Detection and Soft Tissue Deformation for Haptic Simulation of Laparoscopic Surgery", *PhD Dissertation, The University of Texas at Arlington, 2005.*

[17] T. Möller "A Fast Triangle-Triangle Intersection Test", *Journal of Graphics Tools, vol. 2, no. 2, pp. 25-30, 1997.*

[18] J. Andreas Baerentzen and Henrik Aanaes "Signed Distance Computation Using the Angle Weighted Pseudonormal", *IEEE Transactions on Visualization and Computer Graphics, vol. 11, no.3, May/June 2005.*

[19] G. Thürmer and C. Wüthrich "Computing Vertex Normals from Polygonal Facets", *Journal of Graphics Tools, vol. 3, no.1, pp. 43-46, 1998.*

[20] C. H. Séquin "Procedural Spline Interpolation in Unicubix", *Proc. Third USENIX Computer Graphics Workshop, pp. 63-83, 1986.*

[21] D. P. Dobkin and D. G. Kirkpatrick "Determining the separation of preprocessed polyhdra – A unified approach", *Proceedings of the 17th International Colloquium on Automata, Languages and Programming, pp. 400-413, 1990.*

[22] N. Megiddo "Linear-time algorithms for linear programming in r^3 and related problems", *SIAM Journal on Computing, vol.12, pp. 759-776, 1986.*

[23] T. W. Sederberg "Techniques for cubic algebraic surfaces", *IEEE Computer Graphics and Applications, pp. 14-25, July 1990.*

[24] E. G. Gilbert, D. W. Johnson and S. S. Keerthi "A fast procedure for computing the distance between objects in three-dimensional space", *IEEE Journal of Robotics and Automation, vol. 4(2), pp. 193-203, 1988.*

[25] D. Baraff “Curved surfaces and coherence for non-penetrating rigid body simulation”, *ACM SIGGRAPH Computer Graphics, vol. 24(4), pp. 19-28, 1990.*

[26] M. C. Lin and John. F. Canny “Efficient algorithms for incremental distance computation”, *IEEE International Conference on Robotics and Automation, vol. 2, pp. 1008-1014, 1991.*

[27] M. C. Lin “Efficient Collision Detection for Animation and Robotics”, *PhD Dissertation, Department of Electrical Engineering and Computer Science, University of California, Berkeley, December 1993.*

[28] nnnnH. Samet “Spatical Data Structures: Quadtree, Octrees and Other Hierarchical Methods,” *Addision Wesley,* 1989.

[29] S. Quinlan “Efficient distance computation between non-convex objects”, *Proceedings of International Conference on Robotics and Automation, pp. 3324-3329, 1994.*

[30] S. Gottschalk, M. Lin and D. Manocha “Obb-tree: A hierarchical structure for rapid interference detection”, *International Conference on Computer Graphics and Interactive Techniques, pp. 171-180, 1996.*

[31] J. Cohen, M. Lin, D. Manocha and M. Ponamgi “I-collide: An interactive and exact collision detection system for large scale environments”, *In Proceedings of ACM Interactive 3D Graphics Conference, pp. 189-196, 1995.*

[32] M. H. Overmars “Point location in fat subdivisions”, *Information Processing Letters, vol.44 (5), pp. 261-265, 1992.*

[33] M. Moore and J. Wilhelms “Collision Detection and Response for computer animation”, *ACM SIGGRAPH Computer Graphics, vol. 22 (4), pp 289-298, August 1988.*

[34] D. Baraff and A. Witkin ”Dynamic simulation of non-penetrating flexible bodies”, *ACM SIGGRAPH Computer Graphics, vol.26(2), pp.303-308, 1992.*

[35] B. Mirtich and J. Canny “Impulse-based simulation of rigid bodies”, *Symposium on Interactive 3D Graphics, pp.181-ff, 1995.*

[36] X. Wang “Collision Response”, *Internal Report*, *Virtual Environment Laboratory, The University of Texas at Arlington, 2004.*

[37] X. Provot “Deformable constraints in a mass-spring model to describe rigid cloth behavior”, *Proceedings of Graphics Interface, pp. 147-154, 1995.*

[38] Ming. C. Lin and Stefan Gottschalk “Collision detection between geometric models: a survey”, *In proceedings of IMA Conference on Mathematics of Surfaces, pp.37-56, 1998.*

[39] Sensable Technologies, “General Haptic Open Software Toolkit Programmer’s Guide.”

[40] Online available at http://www.nlm.nih.gov/research/visible/visible_human.html

[41] VRML guide, Online available at http://www.graphcomp.com/grafman/vrml/tips.html

[42] Physically based modeling: Principles and Practice, Online Siggraph’97 course notes, Online available at http://www.cs.cmu.edu/~baraff/sigcourse/index.html

[43] Dr Michael Heyns Laparoscopic surgeon's web page, Online available at http://www.drheyns.co.za/proc_inguinal.htm

[44] Metanet software, tutorial on collision detection and response, Online available at http://www.harveycartel.org/metanet/tutorials/tutorialA.html

BIOGRAPHICAL INFORMATION

The author was born in Alibagh, India on 7^{th} February 1984. He completed his schooling from SN Vidya Mandir Secondary School, Kerala in 2001. He earned his Bachelor’s degree in Electronics and Biomedical Engineering from Cochin University of Science and Technology, India in May 2005. Thereafter he pursued Master of Science in Electrical Engineering from University of Texas at Arlington and received his degree in December 2008.