Joint Resection Planning Agent for Robot-Assisted Laser Surgery

Honours thesis by:

Jake Lorkin

Supervisors:

Dr Thomas Bräunl

Dr Brett Robertson (CEO, AIRO)

Richard Chipper (CTO, AIRO)

This thesis is presented in partial fulfilment of the requirements for the degree of Bachelor of Philosophy (Honours) at the University of Western Australia.

Faculty of Engineering and Mathematical Sciences

Word count: 16874

Submitted: October 20th, 2023

ABSTRACT

Knee arthroplasty (or joint replacement) has long served as the solution for conditions like osteoarthritis, providing relief from pain and immobility. Traditional surgical methods employing mechanical cutting tools have their limitations, such as inaccurate bone resection and tissue damage. AIRO introduces a significant innovation, the HAiLO™ system, a robotic Laser Knee Surgery tool. Departing from the limitations of mechanical osteotomy seen in contemporary knee surgeries, the HAiLO™ system harnesses a robotically controlled laser, promising precision, reduced tissue damage, and enhanced implant fits.

The Joint Resection Planning Agent (JRPA) plays a pivotal role in determining bone volume removal, implant and bone fit analysis, and guiding laser ablation, aiming to amplify the system's efficiency and accuracy. This paper elaborates on JRPA's integral role in revolutionizing knee surgery through the HAiLO™ system, anticipating a transformative shift in medical technology and improved patient outcomes. This paper outlines the computational methods and techniques used to achieve the goals of the JRPA. Future work will focus on refining the JRPA's algorithms and expanding its applications.

The primary area focused on in this paper is the moving intersection of complex changing geometries. This is a mathematically non-trivial problem and is a large area of ongoing research in computational geometry. The paper presents a variety of methods attempted to solve this problem with respect to the intersection of knee joints and knee arthroplasty implants. The methods mainly focus around using graphical depth-buffers to generate said intersection volumes, and different data structures for creating, storing, and using the volume.

COPYRIGHT PROTECTED DOCUMENT

ACKNOWLEDGEMENTS

I would like to thank all the employees of AIRO for their fun lunch time games, help and wisdom. Brett Robertson and Richard Chipper for their guidance and welcome, and Nick Dumas for help with many software challenges. Brett and Richard have been some of the best mentors I have ever had, they are incredibly kind and intelligent people. The team at AIRO are revolutionizing the medical industry and working to better people’s lives, I am proud to have a part to play in it, however small. The experience and knowledge I have gained from all the wonderful people at AIRO has made this project an absolute pleasure. I would also like to thank my friends and family for the great support they have been in my life. While they will probably never be able to read this people like Shawn, Amy, Tom, Gavin, and Stef have helped me through so much. Finally, a thank you to Dr. Thomas Braunl who has given me many amazing projects and taught me so much. Helped me to take off my own business and been incredibly patient with my busy schedule.

CONTENTS

FIGURES

TABLES

ALGORITHMS

JOINT RESECTION PLANNING AGENT FOR ROBOT-ASSISTED LASER SURGERY

1 INTRODUCTION

HAiLO™ is a Laser osteotomy system, which offers a less invasive and more precise approach to bone cutting for knee joint surgical procedures [2]. The project is being developed in conjunction with the Australian Institute of Robotic Orthopaedics Pty Ltd (AIRO).

1.1 Background

Knee arthroplasty, commonly known as knee joint replacement surgery, is a procedure in which the damaged surfaces of the knee joint are removed and replaced with prosthetic components to relieve pain and restore function. This surgery is necessary for patients suffering from debilitating knee pain and limited mobility due to conditions such as osteoarthritis, rheumatoid arthritis, or post-traumatic injuries [3].

The current method of knee replacement surgery involves the use of hand-held or hand-guided mechanical cutting tools for bone resection [3]. Surgeons rely on these instruments, considered the industry standard, to shape the bone and prepare it for the placement of prosthetic components. These mechanical tools lack sub-millimetre precision and can cause damage to bone tissue due to heat transfer, debris, vibrations, and biomechanical stress [4].

The limitations of the current method of knee replacement surgery include [3, 5, 6]:

1. Inaccurate bone resection leading to imprecise alignment of prosthetic components, which can compromise long-term outcomes.
2. Damage to bone tissue and surrounding structures due to the use of mechanical cutting tools.
3. Longer operation times and increased blood loss compared to more advanced surgical techniques.
4. Higher risk of complications such as infection, stiffness, and implant failure.

While there are robotically assisted knee arthroplasty products in use, these systems primarily rely on mechanical tool heads to cut tissue [7]. The robotic control allows for better precision but retains the drawbacks of mechanical osteotomy [8]. The proposed HAiLO™ system employs a robotically controlled Laser to ablate bone tissue, replacing hand-guided mechanical cutting methods [2]. Laser osteotomy offers numerous benefits over traditional methods, including improved accuracy, reduced tissue damage, and faster healing times [9]. The use of a robot-controlled Laser removes the constraints of conventional flat cuts, allowing the creation of a proprietary bone surface pattern as described by AIRO. Prosthetics can be slimmer, lighter, and will fix securely to the shaped bone surface. This reduces the chance of implant failure rate and removes the need for bone cement [10, 11].

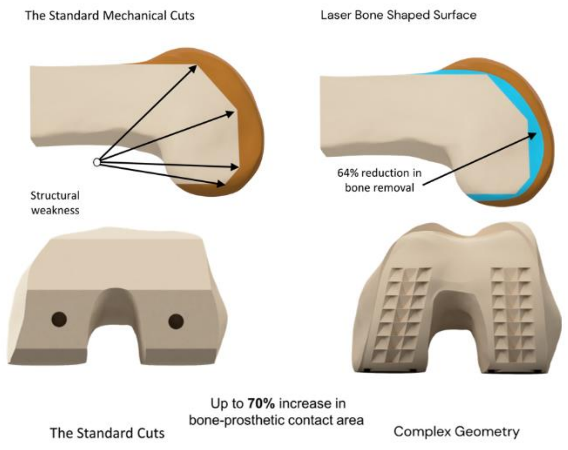

The Standard Mechanical Cuts
Laser Bone Shaped Surface
Structural weakness
64% reduction in bone removal
Up to **70%** increase in bone-prosthetic contact area
The Standard Cuts
Complex Geometry

Figure 1-1: AIRO-x vs. Standard femoral knee implant.

1.2 Problem Identification

The primary objective of this honours project is to develop a Joint Resection Planning Agent (JRPA) for the HAiLO™ Laser osteotomy control system. The control system comprises various algorithms that collaboratively position the robot, fire the Laser, and manage the ablation process [2]. A non-exhaustive list of these algorithms includes:

a) Inverse kinematics and reverse ray tracing to position the Laser beam to a specific point by controlling the robotic arm and Laser scanning head.
b) Generating and updating a point cloud of the knee from scanning data, including overlaying with pre-generated 3D models, to maintain an accurate knee model as the surgery occurs.
c) An algorithm to plan current and future ablation runs to progress from the current knee state to the completion state as efficiently as possible.
d) An algorithm to score the fit of the current knee to the implant, allowing the system to determine a completion state at a point of maximum fitness score.

The project aims to create a Joint Resection Planning Agent (JRPA) to perform task (c) and task (d). Given an initial model of the leg bone (Figure 1-2– left), the system calculates the required changes to the bone's geometry to reach a completed state that enables a successful implant fit (Figure 1-2– right). This initial bone model may be generated by a pre-operation CT scan or initial surface scan of the knee joint once exposed intra-operatively. The target implant resting position relative to the bone will be pre-determined by a surgeon [3, 12].

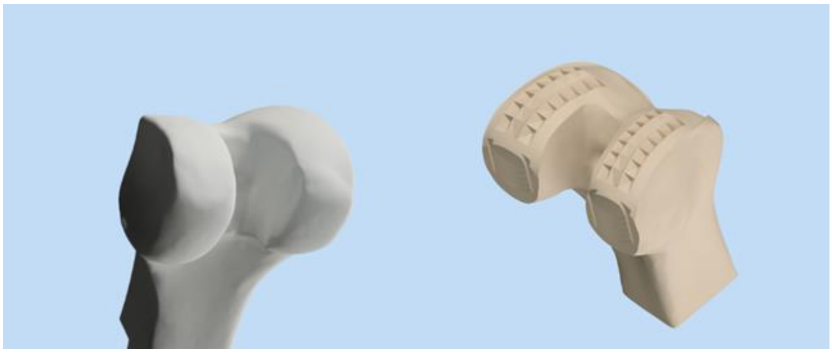


Figure 1-2: 3D model of initial femur bone – Left 3D model of completed femur bone state - Right.

The agent then continuously reperforms these recalculations throughout the surgery, taking in an updated 3D model of the knee and outputting new ablation information. This is necessary as multiple system variables are subject to change, including:

a) Location of the joint relative to the robot (i.e., Patient moves)
b) The surgeon makes a change to the final target position of the implant.
c) Inaccuracies in the system compound cause the predicted state of the bone surface to be incorrect.

Initially, if the implant is moved to its final position relative to the bone, as determined by the surgeon, there will be an intersection between the bone and the implant, as illustrated in Figure 1-3.

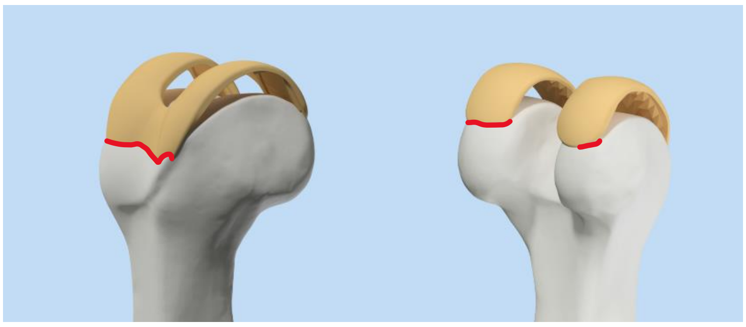


Figure 1-3: Implant intersecting femur bone

The system's end goal is for the bone geometry to reach a state where the implant:

1. Fits successfully.
2. Achieves the best possible fit.

The fit metric may be measured by the surface area contact of the implant to the bone. A successful fit is defined as a state where the implant can be inserted onto the bone and come to rest at the desired position on the surface. A successful fit also requires meeting the following conditions:

1. Achieving minimum surface area contact, where surface area contact is defined by a specified distance/flatness threshold [13].
2. The implant can be moved into its resting position without contacting any blocking geometries larger than a given threshold [4].
3. The deformation of bone tissue, as the implant is inserted, will allow the implant to remain firmly fixed to the bone [14].

Figure 1-2 shows an example of a completed state of a bone, and Figure 1-4 shows the final implant fit.

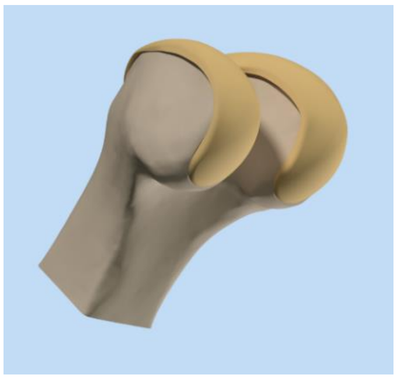


Figure 1-4: Completed femur with implant fit.

The JRPA is responsible for determining the volume and shape of the material to be removed from the bone, continuously updating the calculated volume of removed material based on new point cloud data obtained during the surgical process, assessing the current implant fit percentage and extrapolating data to predict the maximum achievable implant fit percentage, and computing an optimal path for the robotic arm to follow to remove the necessary bone volume. Additionally, the JRPA must be capable of adjusting to any changes in the relative position of the implant to the bone as dictated by the surgeon and providing a warning if the updated position results in a suboptimal predicted fit.

Several challenges need to be addressed during the development of the JRPA.

The implant is designed to securely attach to the bone, requiring a slight intersection between bone and implant volumes to enable a tight fit due to natural deformation [10, 14]. This design aspect must be considered when calculating the required bone modifications. Secondly, the non-uniformity of knee bones, including spurs and osteophytes, may limit the maximum implant fit percentage. The algorithm must account for these irregularities to ensure the optimal fit is achieved for each patient.

Third, the complex geometries of both the bone and implant, consisting of a mixture of concave and convex surfaces, complicate mathematical calculations such as integration [15, 16]. The JRPA must employ advanced computational methods to accurately model and manipulate these intricate shapes. Fourth, the position and orientation of both the implant and bone in 3D space can vary, increasing the complexity of the problem. The algorithm must be parametric and versatile to accommodate a wide range of patient-specific scenarios.

Fifth, determining a fit metric is challenging due to the porous nature of bone. The JRPA must devise an accurate and meaningful fit metric that considers the unique properties of bone tissue. Sixth, the algorithm must be efficient and fast to accommodate the real-time demands of surgery. This necessitates the use of efficient data structures, algorithms, and computational techniques to ensure that the JRPA can provide timely instructions to the robot. Finally, the surgeon may change the implant position during the surgery [3], necessitating dynamic updates to the algorithm. The JRPA must be responsive to these changes and capable of rapidly recalculating the optimal bone modifications and robotic arm paths to accommodate any alterations in the surgical plan.

2 LITERATURE REVIEW

2.1 Knee Arthroplasty

Knee joint deterioration occurs through various mechanisms, such as osteoarthritis, rheumatoid arthritis, and traumatic injury, often requiring surgical intervention through total knee arthroplasty (TKA), [3, 17]. Traditional TKA involves handheld mechanical tools, such as saws and drills, which have significant drawbacks. These tools generate considerable heat and biomechanical stress, causing tissue necrosis [9, 18]. This damage increases the likelihood of post-surgery implant failure and complications [5, 17].

Bone cement, often used in traditional TKA to secure the implant, also presents challenges, including increased risk of infection, bone resorption, and cement loosening over time [6]. Robotic-assisted products, such as the MAKO™ Robotic Arm [7], improve the accuracy and precision of TKA but still utilize mechanical tool heads like rotating cutting bits [19]. While these robotic systems have shown improved outcomes compared to manual techniques [20], they still have the inherent limitations of mechanical cutting [8].

Laser osteotomy has emerged as an alternative to mechanical cutting in various surgical applications, such as dental, spinal, and cranial surgeries [7, 11]. Lasers provide cleaner bone cuts, reduced thermal damage, and improved surgical outcomes compared to mechanical methods [18, 21-23]. Unlike traditional techniques, Laser osteotomy is not restricted to flat cuts and can create bone surface patterns for press-fit implants [23]. There is an industry gap in the use of Laser osteotomy for knee arthroplasty, which offers the listed advantages over current methods [7]. Consequently, there is a gap in academic literature surrounding the use of Lasers in TKA. The proposed HAiLO™ system integrates robot assistance and Laser osteotomy into TKA surgery [2].

The joint resection planning agent (JRPA) will combine current medical research on Laser osteotomy and TKA with engineering research on robotic control, software 3D system characterisation and heuristic algorithms.

2.2 Volume Intersection

A critical aspect of developing the JRPA is to calculate the intersection between bone and implant volumes to ensure a secure fit. This task is challenging due to the complexity of bone and implant geometries, as well as the need for real-time computational efficiency during surgery.

Calculating the intersection of two arbitrary 3D objects is a non-trivial problem [15, 24] and cannot be solved mathematically without making assumptions and constraints on the system. Although 3D CAD software has tools for calculating volume intersections, these methods typically involve using parametric models, which are computationally intensive and unsuitable for the real-time demands of the JRPA [25].

Discrete solutions such as point cloud methods [24] or depth buffer analysis [16] provide more feasible approaches. Such solutions approximate volume but come at a greatly increased computational speed [26]. For the purposes of the JRPA, the resolution of volume detection only needs to meet the accuracy of the Lasers ablation radius [27]. Point cloud methods involve representing 3D objects as sets of points in space and calculating intersections by comparing point positions [24]. Depth buffer analysis, on the other hand, is a computer graphics technique that determines the visible surface at each pixel by comparing depth values [16]. Both methods can approximate volume intersections efficiently, but they have their limitations.

CPU-based calculations are too slow for real-time surgical demands, necessitating the use of GPUs for parallel processing and faster computation [28]. However, point cloud methods can still be too slow for some applications, and depth buffer analysis runs into issues of intersecting with an object more than once [16, 24]. These challenges highlight the need for further research and development of efficient algorithms for volume intersection in the context of the JRPA.

2.3 Implant Fit Analysis

Another goal of the JRPA is to analyse the fit between the implant and the bone, which involves more than just calculating the surface area contact between the two [5, 8, 14]. Due to the porosity of bone and natural deformation [5, 13], the system must consider various factors to ensure a secure fit. Surface flatness and texture thresholds will need to be defined [29, 30], taking into account research on knee prosthetics and osteotomy [4]. Factors such as optimal prosthetic fit and bone surface characteristics must also be considered [4, 5, 31].

The JRPA must integrate various sources of information to produce one or more numerical metrics for implant fit. These metrics should consider surgical knee arthroplasty metrics [32] and be computationally efficient for real-time use during surgery. The 3D mesh of the bone model will need to be analysed by the JRPA to identify surface characteristics [13], then utilise research on prosthetic fitting and bone deformation to determine the current implant fit score. The final position of the implant can change from the predicted position due to the breaking of bone peaks and deformation [4]. To accurately determine the shape of the bone required for the implant to end up in the correct position once fit, testing will need to be done. The JRPA will need to have data on the effect particular bone surface characteristics have on the final implant position and take this into account when calculating the modified shape of the bone.

2.4 Bone Deformation / Press fitting

The AIRO-x implant, designed to clip onto the bone using undulating grooves to dig into the bone's surface [10], introduces additional considerations for the JRPA. The implant's press-fit design requires the bone to be over-extruded with respect to the inverse volume of the implant, allowing for the natural deformation of the bone to secure the implant firmly. The JRPA must account for this when calculating the completed bone shape, drawing from medical research on press-fit prosthetics and bone characteristics [12, 14]. Considering bone regrowth and post-operation issues specifically with TKA [17, 32].

2.5 Agent Output

Once the volumetric information, current and extrapolated fit analysis have been calculated, the JRPA must output paths and/or points for the Laser to ablate. There are various methods for converting a 3D volume into a series of steps for a machine to take to remove that volume [33, 34]. The HAiLO™ robot's degrees of freedom offer more alternatives than a planar-based approach for generating ablation paths [33]. The complex geometry of the system and resulting volume intersection will require the robot to move to multiple locations around the knee joint, ablating a series of paths into the bone at each one.

A tool path algorithm, CAM algorithm, and optimization algorithm will all need to be selected to produce the required ablation paths and robot positions [27, 34]. These algorithms will need to be chosen and modified to best fit the requirements of the HAiLO™ system while managing the six primary path mapping problems outlined in [34]. Testing will need to be performed to find the most optimal algorithms. A custom algorithm will then need to be constructed due to the unique use case of the HAiLO™ system, modified for Laser ablation rather than cutting [35].

Several variables will need to be obtained for the JRPA to calculate the tool paths, such as Laser specifications, cutting depth/width, speed, etc. [27, 35, 36]. These variables are crucial for ensuring the efficiency and accuracy of the ablation process.

3 GUI CREATION AND MODEL VISUALISATION

All the code for this project was written in the Rust programming language. AIRO use Rust as their language of choice, so it made sense to use Rust to make future integration easier. Rust is an effective and safe language with strong guarantees for memory and thread safety, which are important features for controlling a medical device.

A custom Graphical User Interface (GUI) was developed in rust to visualise models and the pipeline of operations. It can be complicated to solve issues when working with 3D objects and math. When something is wrong, all you see is a bunch of numbers. Millions of data points are meaningless to the human eye unless you have some way to visualize the data.

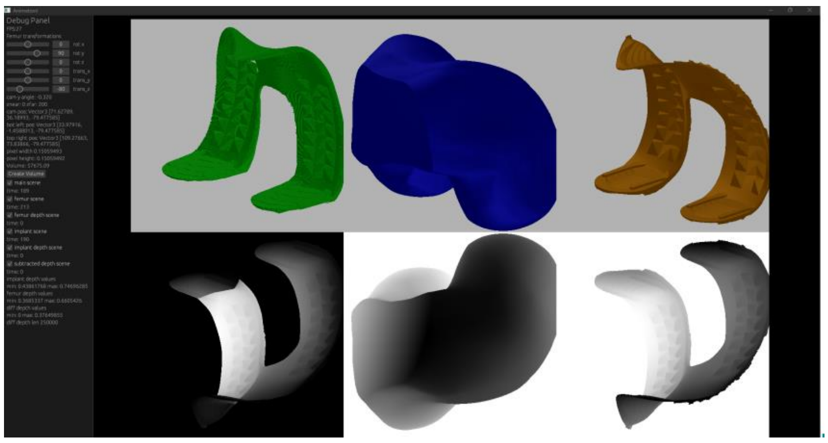
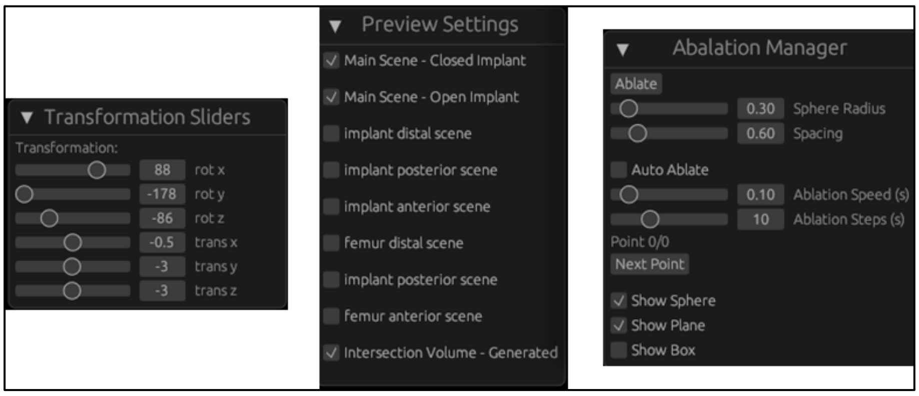


Figure 3-1: Custom GUI window showing user interface and render pipeline.

Rendering operations and window creation were implemented using the rust 'three-d' crate. All GUI elements and user interface were implemented with the 'egui' crate. As well as a mix of custom opengl shaders for rendering depth buffers.

Visualising operation pipeline was extremely useful for debugging code and maths.



▼ Transformation Sliders

Transformation:

88	rot x
-178	rot y
-86	rot z
-0.5	trans x
-3	trans y
-3	trans z

▼ Preview Settings

- ✓ Main Scene - Closed Implant
- ✓ Main Scene - Open Implant
- ☐ implant distal scene
- ☐ implant posterior scene
- ☐ implant anterior scene
- ☐ femur distal scene
- ☐ implant posterior scene
- ☐ femur anterior scene
- ✓ Intersection Volume - Generated

▼ Abalation Manager

Ablate

0.30	Sphere Radius
0.60	Spacing

☐ Auto Ablate

0.10	Ablation Speed (s)
10	Ablation Steps (s)

Point 0/0

Next Point

- ✓ Show Sphere
- ✓ Show Plane
- ☐ Show Box

Figure 3-2: Example GUI control panels for displaying and controlling JRPA.

Three dimensional models displayed in figures in this paper will be rendered using one of the following software programs:

a) MeshLab – Mesh viewing and editing program.

b) Autodesk Inventor – CAD software.

c) The custom GUI

A comprehensive list of software used in this project can be found in Table 12-1.

4 HARDWARE CONFIGURATION

To ensure reproducibility of experiments, and to apply context to the algorithm timing results presented, the hardware setup used for running the algorithms is outlined in Table 4-1.

Table 4-1: Experimental computer hardware configuration

System Component	Information
System	Surface Laptop Studio running Microsoft Windows 11 Home x64
Processor (CPU)	11th Gen Intel® Core™ i7-11370H @3.30GHz
Graphics Processing Unit (GPU)	NVIDIA GeForce RTX 3050 Ti Laptop GPU with 4096 MB GDDR6 memory
Memory (RAM)	16 GB DDR4 4267 MHz
Storage	512 GB SSD

5 VOLUME INTERSECTION

To determine the volume of bone and other tissues to be removed, the intersection of the bone and implant need to be determined. The question is, what do we need to subtract from the bone so, a desired implant will fit? This intersection is the overlapping region of two objects. Calculating the volume and the intersection volume of simple objects such as cubes is easy. For two complex non-convex 3D objects it is a complicated problem that is not trivial to solve. A range of methods were tried to calculate the intersecting volume.

5.1 Using CAD

CAD or computer automated design software can in most cases find and extract the intersecting volume of two objects. The models for the implants and bones are relatively complex and the process was slow and most of these algorithms are proprietary, meaning they couldn't be used in a commercial project. CAD software requires the objects to be in specific formats, that would require long conversion times from the HAILO scan data and pre-operation CT scan data. Section 6.2.4.1 discusses the difficulty CAD software has working with complex mesh objects.

As shown in Figure 5-1 the intersection of the femur and implant can be calculated. Section 5.2 discusses why the Boolean intersection of the implant and bone is undesirable.

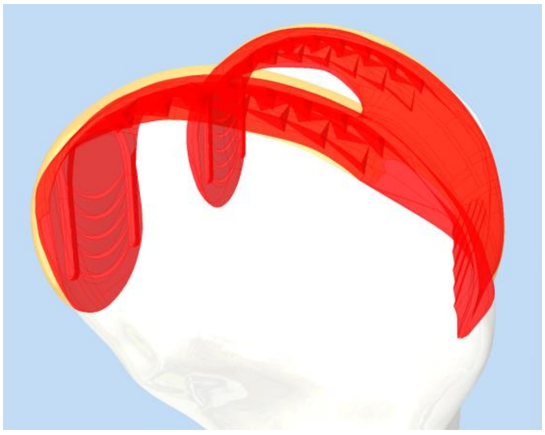


Figure 5-1: Intersection of femoral implant and femur in Autodesk Inventor.

While many of the operations discussed in this paper and used in the JRPA pipeline can be achieved with CAD software, the conversion of data required would be too slow and restrict other sections of the pipeline. Therefore, another method is needed.

5.2 Boolean Mesh Operations

To attempt to calculate the volume intersection Boolean mesh operations were attempted. These are mathematical Boolean operations (XOR, AND, OR) on mesh objects, such as subtracting one object from another or the union of objects. Using the 'libigl' geometry processing library, the result of various Boolean mesh operations was generated. Object *A* was a total knee femur implant, object *B* was a simplified model of a femur.

Table 5-1: Mesh models of femoral implant and femur.

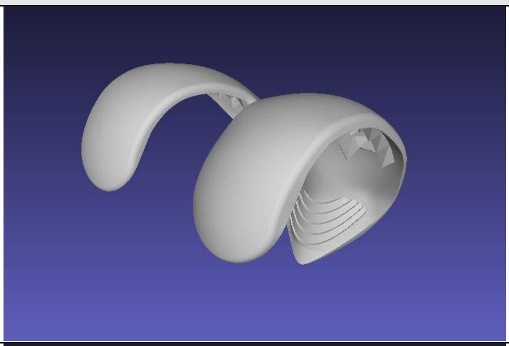
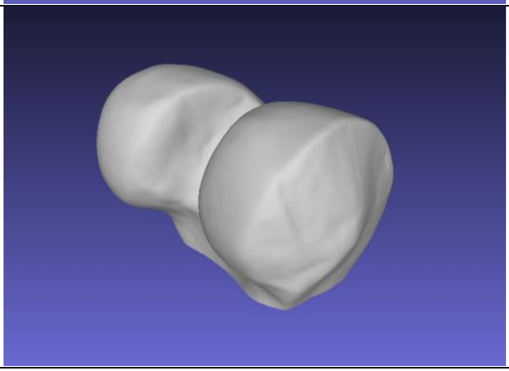
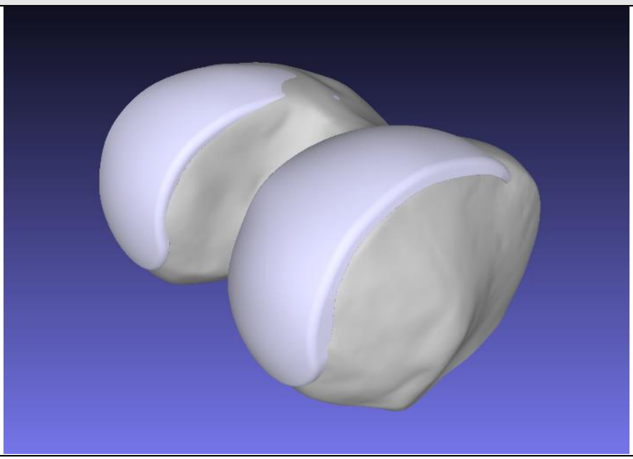
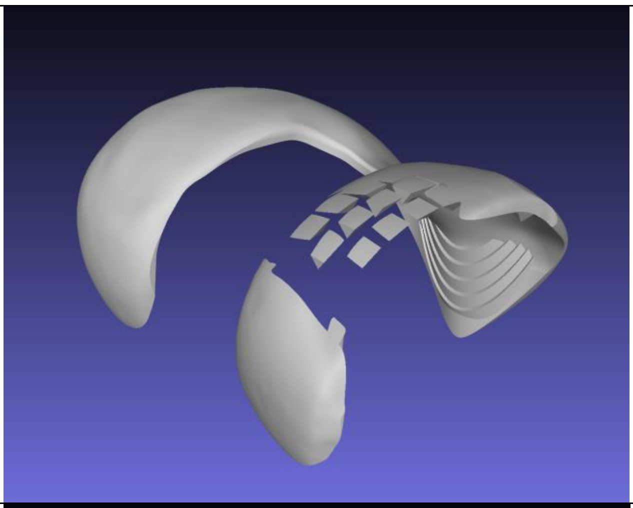
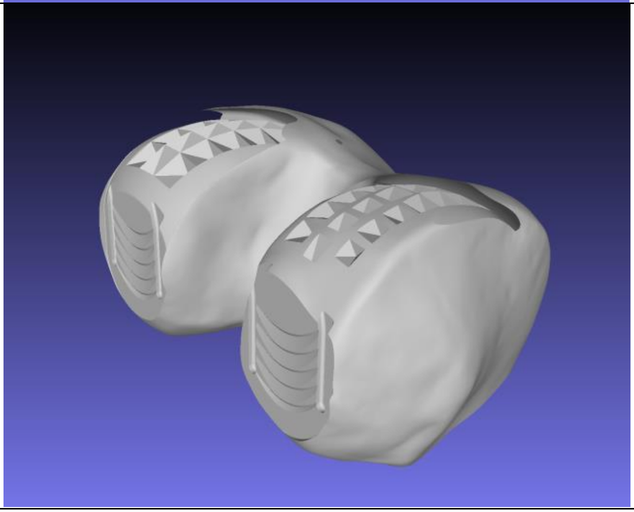
Model	Result	Mesh Triangles
$A := \text{Implant Model}$		151,878
$B := \text{Femur Model}$		378,124

Table 5-2: Boolean mesh operations on implant and femur bone.

Name	Boolean Operation	Result
Union	$A \cup B := \{x \in \mathbb{R}^3 \mid x \in A \textbf{ and } x \in B\}$	

Intersection	$A \cap B := \{x \in \mathbb{R}^3 \mid x \in A \; \mathbf{or} \; x \in B\}$	
Difference Or Subtraction	$B \setminus A := \{x \in \mathbb{R}^3 \mid x \in A \; \mathbf{and} \; x \notin B\}$	

The Boolean intersection took 18.505 seconds to compute, and the Boolean difference took 18.118 seconds to compute. These times are significantly larger than the target order of magnitude. Due to the lack of information in the STL mesh format these algorithms require a lot of computation and time scales with the number of triangles in the mesh faces.

To fit the implant onto a femur the implant needs to be placed down on to the femur as described in the AIRO-X patent [10].

For a given final implant position given by the supervising surgeon, see Figure 5-2, there will be some volume that needs to be removed from the bone for the implant to;

1. Be placed into position by a surgeon.
2. Remain securely in position.

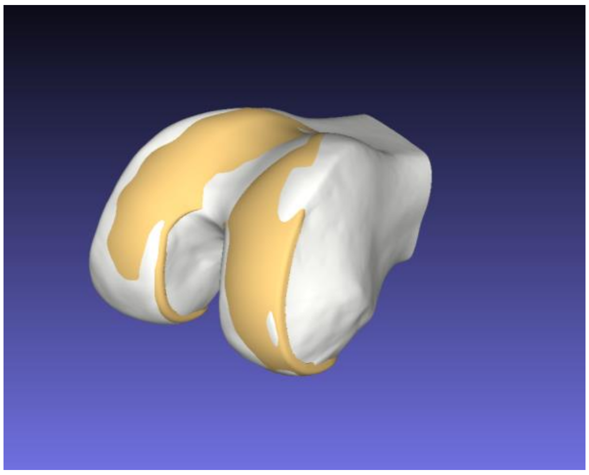


Figure 5-2: Femoral implant (yellow) in a final resting position on the femur.

When trying to place the implant onto the femur as shown in Figure 5-3, contact will be made between the implant and the femur bone.

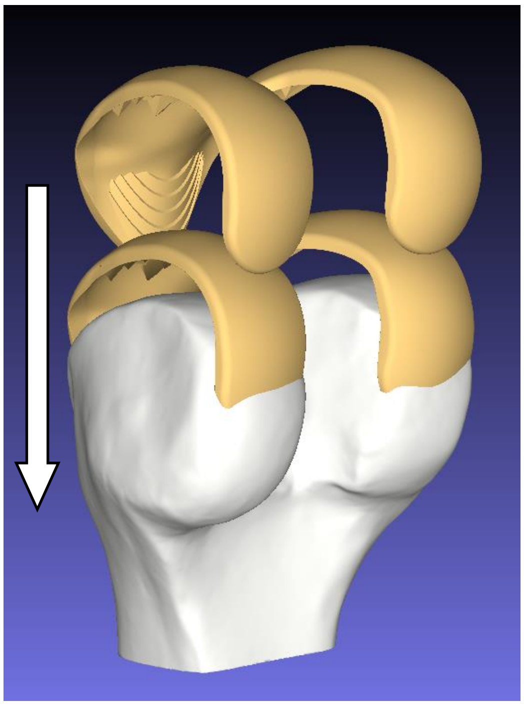


Figure 5-3: Placement of femoral implant.

One thing to try would be removing the intersection of the implant and femur shown in Table 5-2, in other words, the volume where the implant and femur both occupy. If we try this, we get the difference or subtraction of the implant from the femur, again shown in Table 5-2. While this removes the femur intersecting the implant's final resting position, there is still bone in the way of distal implant placement, so the implant cannot be placed into position, as demonstrated by Figure 5-4. It is also physically impossible to ablate the bone this way.

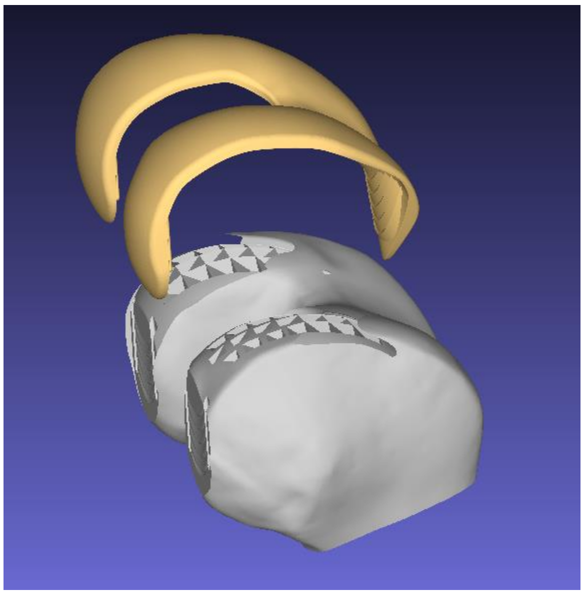


Figure 5-4: Implant and Boolean subtracted femur.

An intersection volume is required that considers both the final implant position and the path of the implant to that position.

Note:

As shown the Boolean intersection of the bone and implant is not necessarily the correct volume to be removed. When referring to the removed volume in this paper the term “intersection volume” is generally used. “Depth intersection volume” as in the total volume intersected when moving the implant from a high ‘Z’ point through to its final resting position is a more accurate description. However, for the sake of brevity this paper may just refer to the calculated volume to remove as “intersection volume”.

One thing to try would be to artificially extending the outer implant face so that the Boolean mesh intersection works. The extended implant is shown in Figure 5-5.

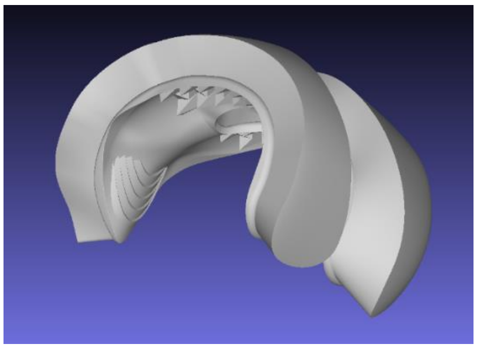


Figure 5-5: Manually extended femoral implant model.

A Boolean mesh subtraction from the femur leads to the Boolean difference and Boolean intersection shown in Figure 5-6.

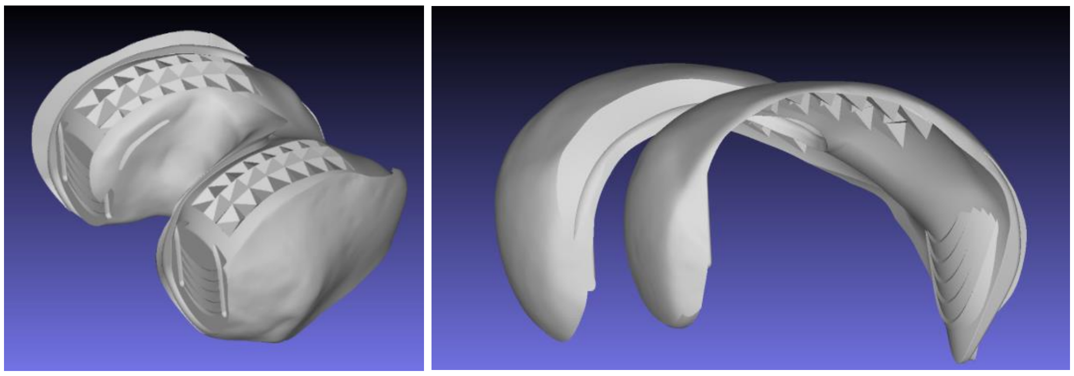


Figure 5-6: Femur Boolean operations with extended implant model – Difference (left), Intersection (right)

The Boolean intersection took 121.962 seconds to compute, and the Boolean difference was computed in 75.856 seconds. This led to a Boolean mesh difference more indicative of the volume that needs to be removed. However, the long processing time, and the difficulty of working with the STL data structure gives a need for another approach.

5.2.1 Difficulties of mesh data structure

Even with a correct intersection mesh the mesh format is still an undesirable format to work with. The points of the mesh are heterogeneous, meaning that resolution and information density is not consistent across the surface of the mesh, shown in Figure 5-7. Mesh formats are not easy to preform the required JRPA operations on.

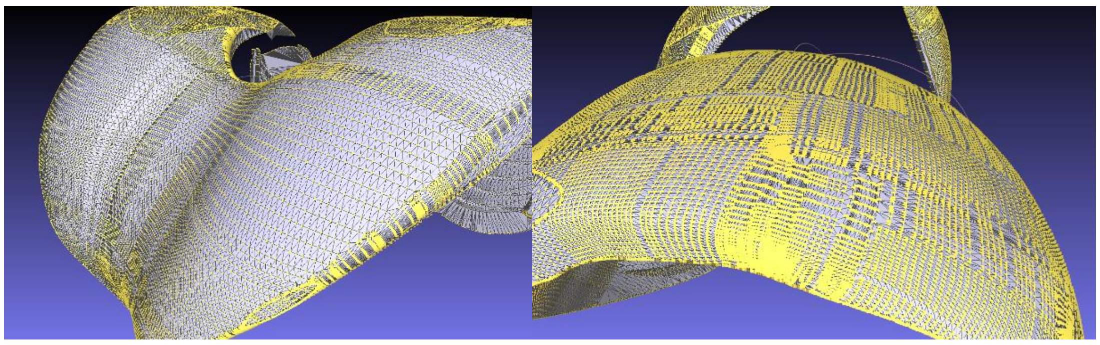


Figure 5-7: Femoral implant model with highlighted vertices.

5.3 Approximate Convex Decomposition

The V-HACD (Voxelized Hierarchical Approximate Convex Decomposition) algorithm was used to attempt to ‘decompose’ the implant and femur into multiple convex components [37]. Convex and concave polyhedra have distinct geometric properties, which lead to differences in the operations and analysis that can be performed on them. Here are some operations and properties that are unique to convex polyhedra or are more straightforward to perform on them compared to concave polyhedra [37]:

a) **Support Function**: For a convex polyhedron, you can define a support function that, given a direction, finds the furthest point on the polyhedron in that direction. This is not directly applicable to concave polyhedra because there might be multiple "furthest" points in each direction.

b) **Minkowski Sum**: The Minkowski sum of two convex polyhedra results in another convex polyhedron. This operation is more complicated for concave polyhedra.

c) **Linear Programming**: Solving linear programming problems is more straightforward with convex polyhedra. Given a direction, you can efficiently find the point on the polyhedron that maximizes or minimizes the dot product in that direction.

d) **Volume and Surface Area Calculation**: While it's possible to compute the volume and surface area for both convex and concave polyhedra, the algorithms for convex polyhedra are often simpler and more efficient.

e) **Visibility**: From any point inside a convex polyhedron, you can see the entire interior of the polyhedron. This is not the case for concave polyhedra, where parts of the interior might be occluded by other parts.

f) **Intersection Tests**: Testing whether a ray or line segment intersects a convex polyhedron can be done more efficiently than with a concave polyhedron.

Some of these convex operations and properties were relevant later in the design of the JRPA. Being able to split the concave implant and bone models up into convex components would have potentially simplified future algorithms.

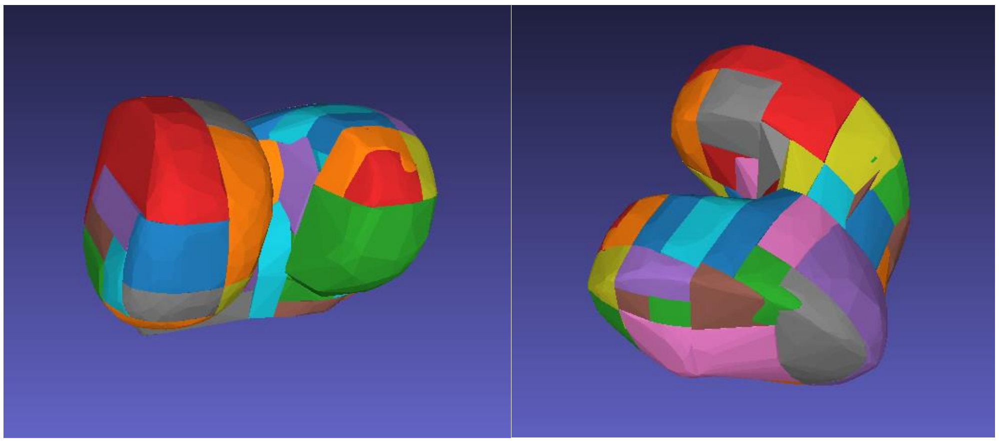


Figure 5-8: Approximate convex decomposition of a femur model with the VHACD algorithm.

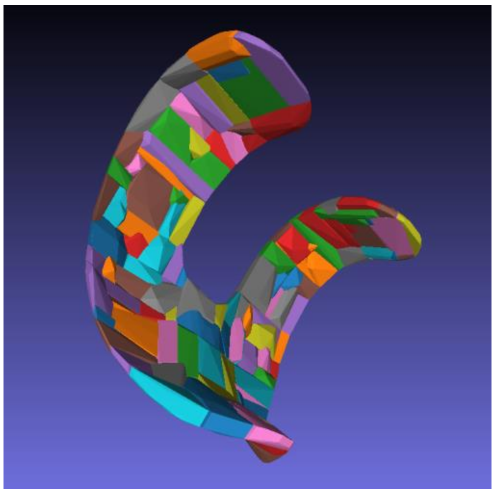


Figure 5-8: Approximate convex decomposition of an implant model with the VHACD algorithm.

This algorithm seemed to struggle with the fine details of the models and too much information was lost. Additionally, while potentially reducing time complexity or enabling the use of convex only algorithms, the multiple component form would have made future programming steps tricky. For example, properties like “What part of each component is the surface of the whole object?”, would be impractical but necessary to track. While a convex decomposition of the models with acceptable computation time and accuracy would definitely be possible, more time into this approach was decided to not be worth further investigation.

5.4 Extracting Implant Face

For most of our volume intersection methods only the inner face of the implant is required, the face that will be touching the femur. The 3D model of the implant is a closed mesh. Meaning it surrounds a volume. For ease of calculation, the inner surface of the mesh was extracted, i.e., the side that contacts the bone.

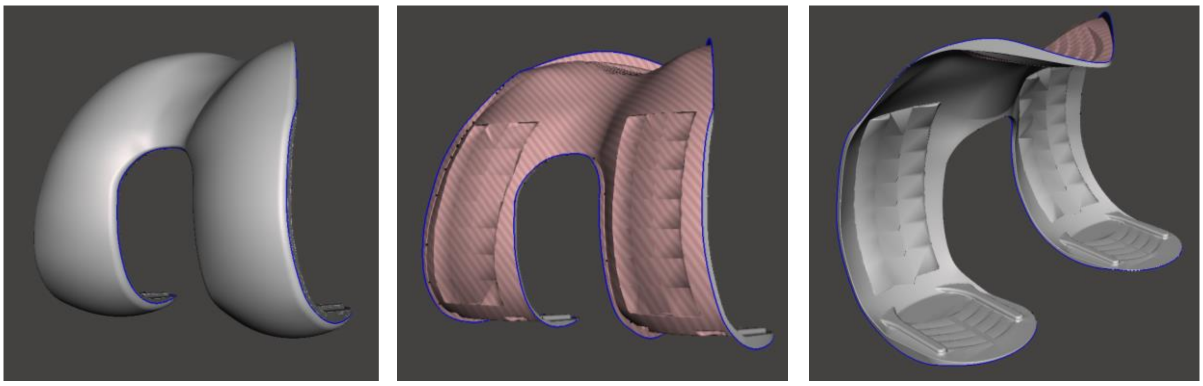

Femoral Implant: Outer face
Femoral Implant: Inner face backside
Femoral Implant: Inner face

Figure 5-9: Femoral implant face labelling.

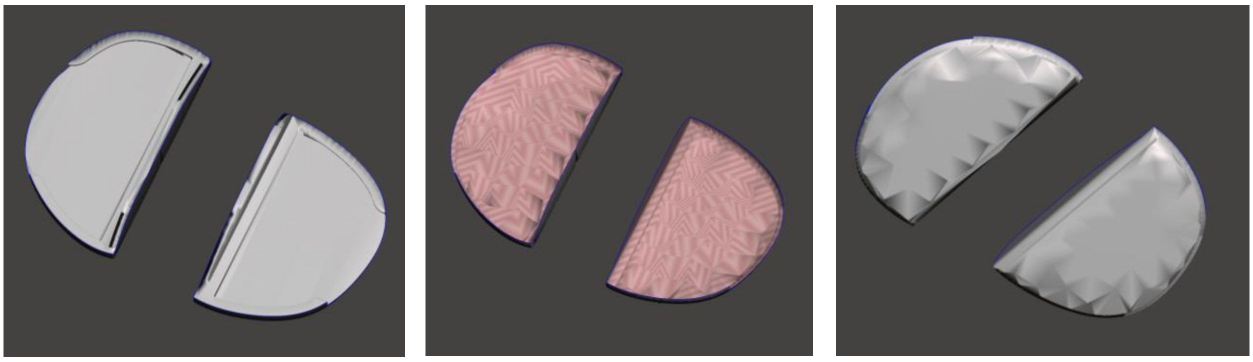

Tibial Implant: Outer face
Tibial Implant: Inner face backside
Tibial Implant: Inner face

Figure 5-10: Tibial implant face labelling.

This was done using Autodesk Meshmixer, a software program for editing triangle mesh models such as STL files. The outer face triangles were manually selected and removed. As AIRO are only developing a small range of differing implant models this process could be done manually for each one. Due to the non-convex nature of the implant model, it would be non-trivial to automate this process.

5.5 Ray Based Approach

To attempt to calculate information about the intersection volume between an implant and bone, a ray intersection approach was attempted. The implant and bone were first aligned in space. Then a virtual ray was fired in the direction of implant placement. To check the ray hit the bone or implant mesh first, ray-triangle intersection was computed for every triangle in the meshes and the intersection points compared.

If the ray hit the femur first, the femur would need to be removed at that point for the implant to be placed into position. The depth removed would be the length between the bone intersection point and the implant intersection point. If the ray intersected the inside implant face, then no removal was necessary. The ray intersecting the outside implant face doesn't yield any useful information, hence why the outside face of the mesh is removed in section 5.4. Firing many rays would give the information to recreate the intersection volume.

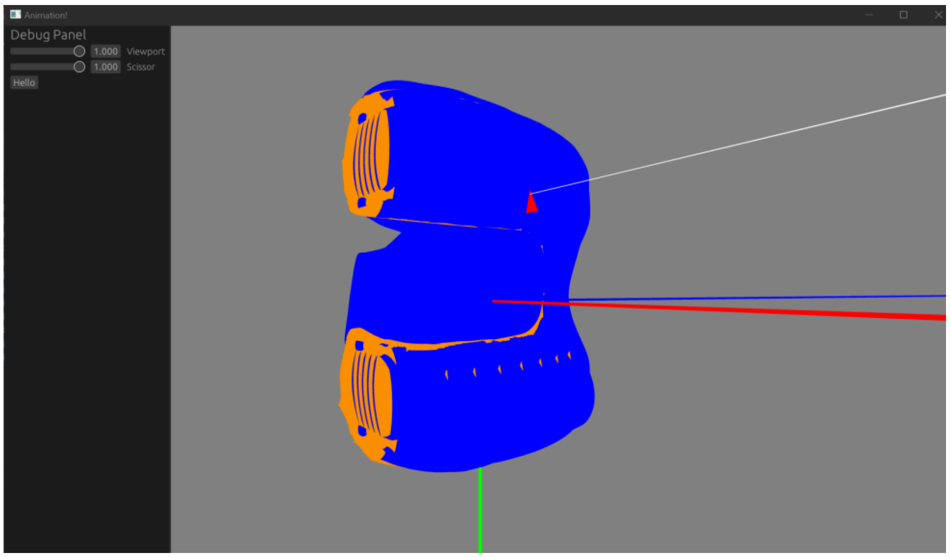


Figure 5-11: Ray test on Femur and Implant

This method was far too slow. For a 20-micron resolution, about 3000 x 3000 rays would be needed. This would take a long time as for each ray all the triangles in the meshes would need to be checked. This can be upwards of 270,000 for some CT-scans.

Data structures such as a k-d tree could be used to speed up lookups, but CPU based methods will be too slow. A similar functionality can be achieved in parallel using the depth buffer functionality of the graphics card as outlined in section 5.7.

5.5.1 Möller–Trumbore Ray-Triangle Intersection Algorithm

All the ray-triangle intersection testing used in this project was achieved using the Moller-Trumbore ray-triangle intersection algorithm. The Möller–Trumbore intersection algorithm is a fast method to determine the intersection of a ray and a triangle in three-dimensional space, commonly used in computer graphics for ray tracing [38]. The algorithm computes the barycentric coordinates of the intersection point, if it exists, by solving a system of equations derived from the parametric representation of the ray and the plane containing the triangle. If the barycentric coordinates satisfy certain conditions, the intersection point is inside the triangle, otherwise, there is no intersection.

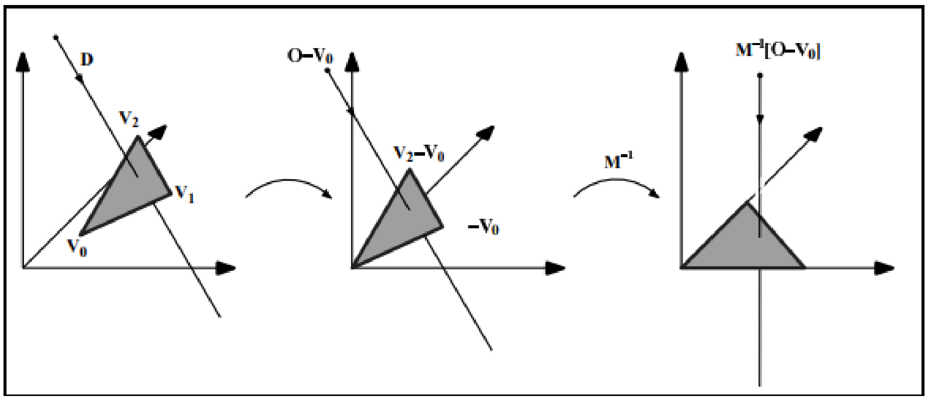


Figure 5-12: Translation of a triangle to barycentric coordinates for ray intersection testing [38].

5.6 Point Cloud Approach

Another method for volume analysis is using point clouds. Generating n random points inside the bounding box of an object we can then check if each point is inside the object or not by firing a ‘ray’ through the point and counting how many triangles in the mesh the ray intersects. Ray intersect look ups are done quickly using a k-d tree that can be used to find triangles that will appear near the ray. These triangles are then checked using the Möller–Trumbore ray triangle intersection algorithm. The method was tested on a femur model of known volume.

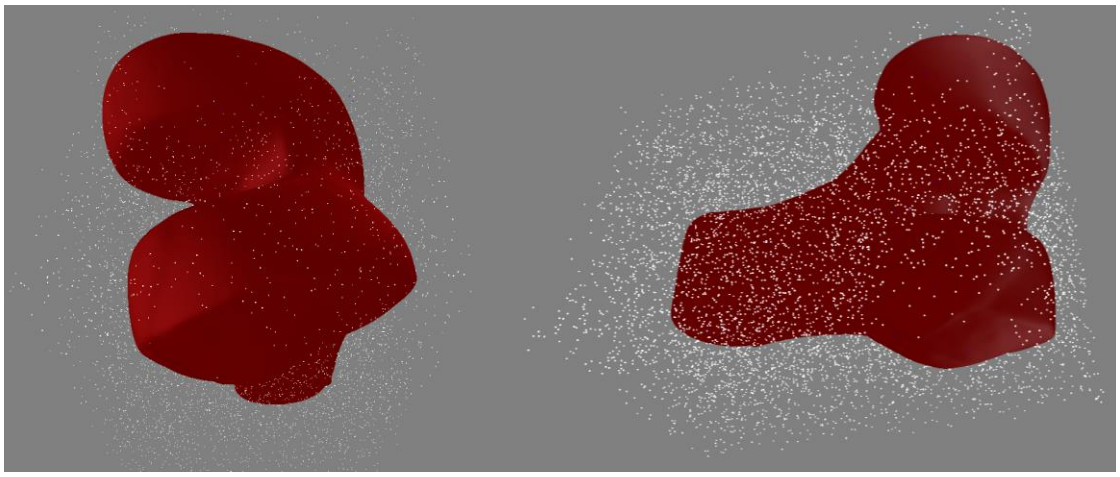


Figure 5-13: Point cloud test on femur.

For 10,000 points a volume of 183846.61mm^3 was calculated, a -0.5246406% error taking 0.548 seconds. This method is not viable for the purposes of the JRPA as it is not deterministic, and the resulting point clouds are ill-equipped for other types of operations.

5.7 GPU Depth Buffer

The depth buffer is used in computer graphics to represent depth information about objects. The GPU's depth buffer functionality allowed for swift comparisons between two models, determining their differences. Rendering the implant and a bone the graphics card quickly gives back a depth buffer or depth map of the objects, seen in Figure 5-9.

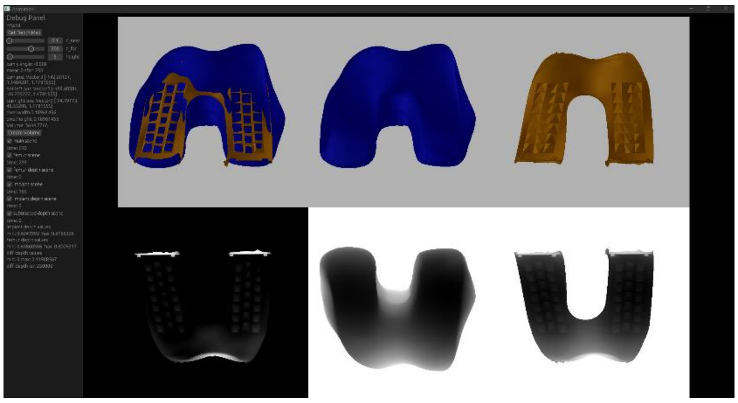


Figure 5-9: Rendering of bone and femur for depth maps.

The depth buffer tells us how far away that object is from the virtual camera at each point on the screen. In this example darker points mean closer and lighter points are further way. Comparing gives a difference in height for the two objects. This can be stitched back together into an intersection volume. The volume is a discrete approximation due to the pixel nature of the depth buffer, but accuracy can be improved by rendering at a higher resolution at the cost of computation speed. Using the rust ‘three_d’ library to render the objects an OpenGL call could be made to retrieve the depth buffer, calculated by the graphics card when it renders a scene.

5.7.1 Extracting Depth Information

To extract the depth map, first the bone or implant is rendered with an orthographic camera, explained in Figure 5-10. An orthographic camera is used as we don't want scale of objects changing with distance.

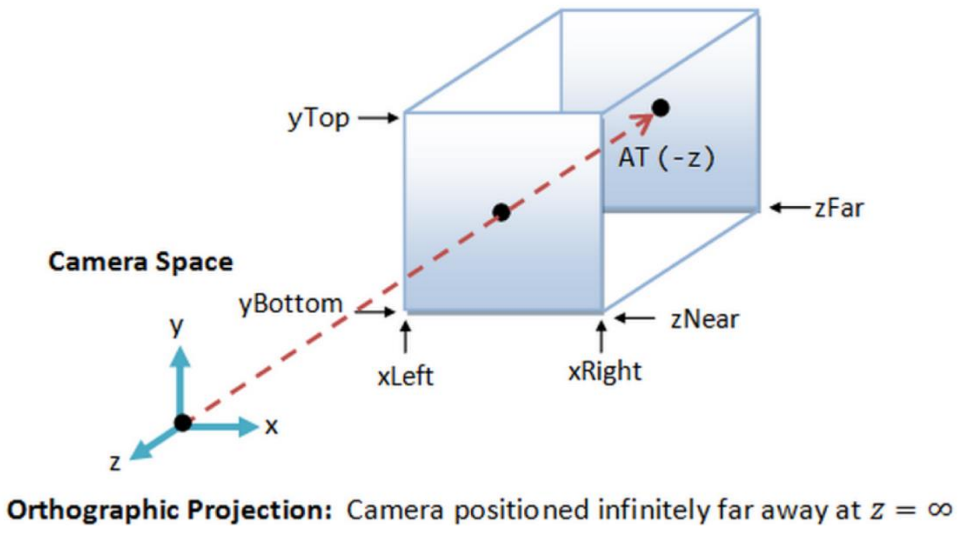

yTop
AT (-z)
zFar
Camera Space
yBottom
zNear
y
xLeft
xRight
x
z
Orthographic Projection: Camera positioned infinitely far away at $z = \infty$

Figure 5-10:Orthographic camera diagram [39].

The depth buffer is a 24-bit number scaled to a 32-bit float from 0 to 1.0, which maps from the zNear to the zFar plane of the camera frustum. To maximise resolution of the depth buffer we set the zNear = 0 and the zFar = the depth of the bounding box of our scene. Then the camera is positioned at the edge of the bounding box.

The resolution of the depth buffer is approximately equal to the f32 epsilon = 1.1920929E-7f32. This is the difference between 1.0 and the next larger representable number. Or the approximate accuracy of a f32 number. This value is so small that compared to the accuracy loss from the discretisation of the objects into a depth buffer. So is accuracy maximised but treated at negligible for the purposes of volume calculation.

A render height in pixels is specified and then the render width is calculated to make the pixel width and height the same. To visualize the depth maps their min and max non-zero values are found, the depth values are then mapped from the min being 0 to the max being 255. Which is then exported as a greyscale image.

6 CUBOID STRUCTURE

A data structure is required to turn the extracted depth information into a usable volume. The data structure must allow look ups for information about the volume and editing the volume for the purpose of simulating ablation. To begin with a custom cuboid structure was made, storing each pixel in the depth map as 8 points, one for each vertex of a cuboid representing the volume of that pixel.

6.1 Creating Cuboid Structure

To generate the structure the implant depth map and bone depth map are subtracted from one another to find the intersection depth. Figure 6-1 shows a diagrammatic representation of the process.

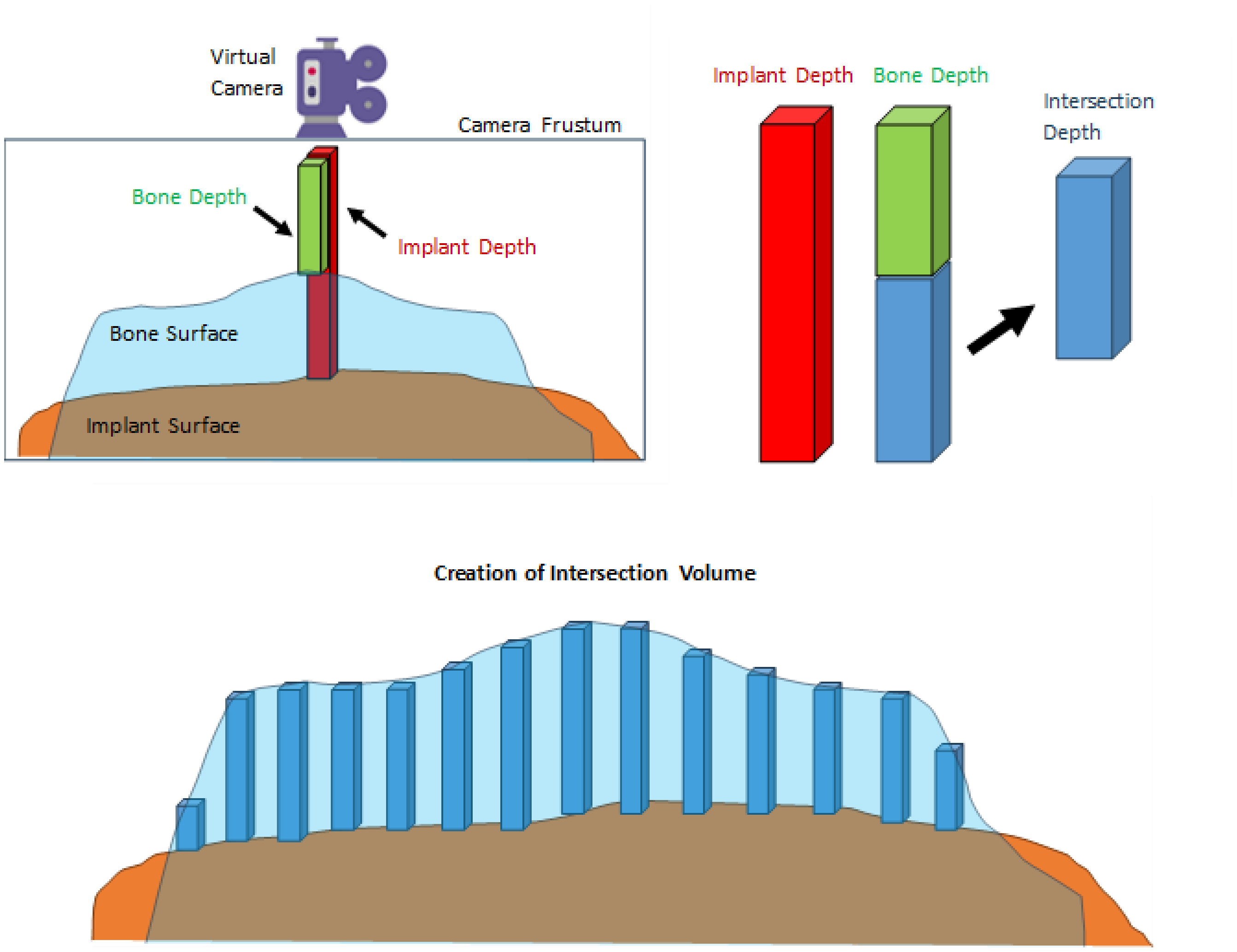


Figure 6-1: Creation of intersection volume using depth buffers of bone and implant.

First calculate the implant depth, i.e., the distance between the camera and the implant surface, and the bone depth i.e., the distance between the camera and the bone surface.

Subtracting the bone depth from the implant depth gives the intersection depth. This is the distance between the surface of the bone and the surface of the implant. Doing this to the inner surface of the implant will give the distance at a point that needs to be removed for the implant to fit.

Then to recreate the intersection volume from the individual cuboids translate the cuboid by its half-depth and then the depth to the bone at that point, cuboid translation is shown in Figure 6-2.

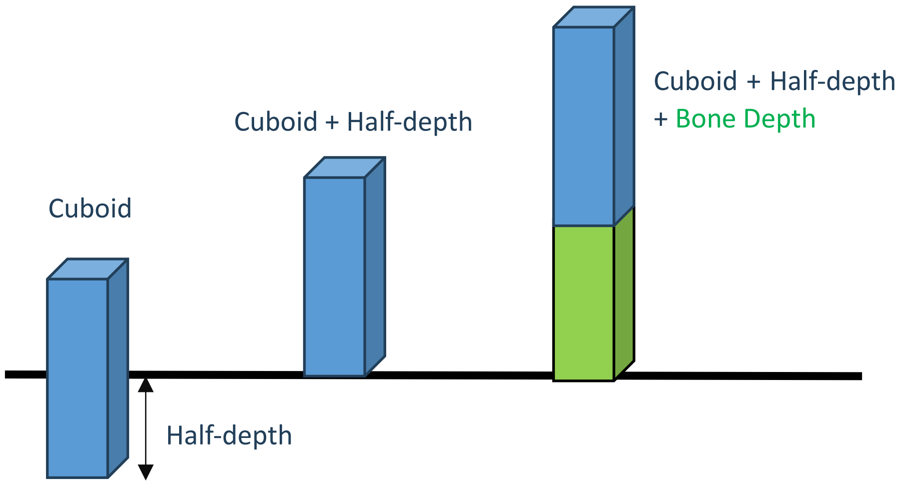


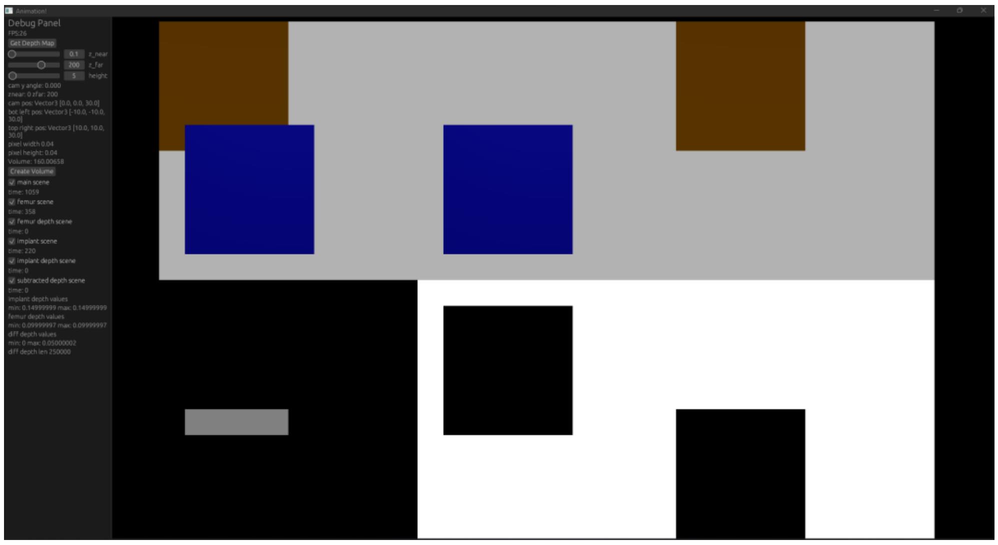
Figure 6-2: Translating cuboids.

The width and height of the cuboids is determined by the bounds of the camera frustum divided by the rendering width and height in pixels. To test the volume of a cuboid depth-intersection every cuboid is looped through, and the volumes are summed.

6.2 Cuboid Results

6.2.1 Cube Test

To test the method two cubes with side length 10mm were used as test objects. The orange cube had its back face removed, to represent the implant and the blue cube representing the bone. The cubes were offset from one another by small values, making the actual volume of the intersection easy to calculate. The rendering resolution was set to 500x500 pixels. Figure 6-3 shows the rendered scene cubes on the top row of the GUI and the generated depth maps below.



```
Debug Panel
FPS:26
0.1  z_near
200  z_far
5    height
cam y angle: 0.000
znear: 0 zfar: 200
cam pos: Vector3 [0.0, 0.0, 30.0]
bot left pos: Vector3 [-10.0, -10.0, 30.0]
top right pos: Vector3 [10.0, 10.0, 30.0]
pixel width 0.04
pixel height: 0.04
Volume: 160.00658
✓ main scene
time: 1059
✓ femur scene
time: 358
✓ femur depth scene
time: 0
✓ implant scene
time: 220
✓ implant depth scene
time: 0
✓ subtracted depth scene
time: 0
implant depth values
min: 0.14999999 max: 0.14999999
femur depth values
min: 0.09999997 max: 0.09999997
diff depth values
min: 0 max: 0.05000002
diff depth len 250000
```

Figure 6-3: GUI showing cube test.

The cubes were translated to have a theoretical depth intersection volume of 160mm^3 the calculated volume was 160.00658mm^3, a 0.004% error. The result was visually inspected in a CAD program to check that the models lined up at expected. Figure 6-4 shows the intersection volume in black, lining up as expected with the two cubes.

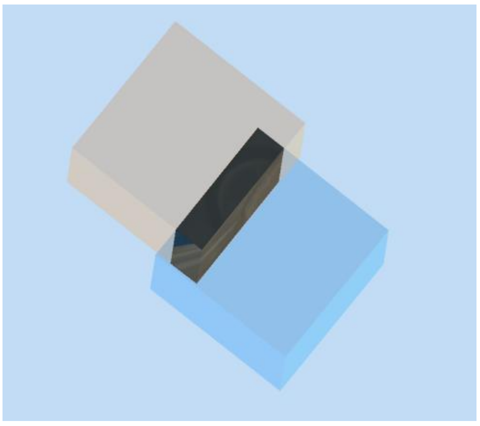


Figure 6-4: Visual inspection of cube intersection volume.

To test angled surfaces one of the cubes was rotated 45 degrees relative to the other. This gave a triangle cross-section area of 25mm^2 with a depth of 5mm leading to an expected volume of 125mm^3. The calculated volume was 125.021484mm^3, a 0.017% error. The larger error is expected for angled surfaces being approximated by the cuboids.

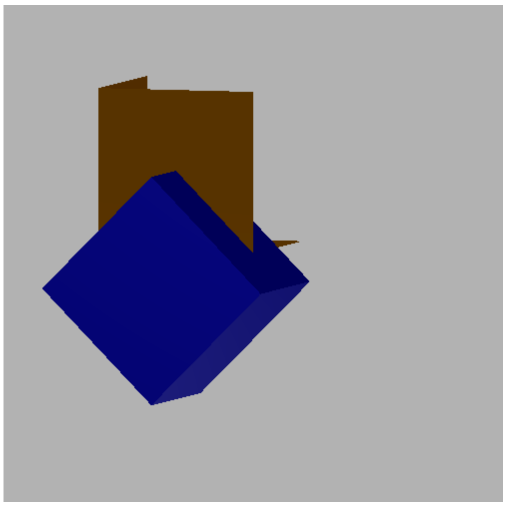


Figure 6-5: Angled cube intersection test.

6.2.2 Implant and Bone Positioning

The implant and bone are positioned relative to each other with a given axis rotation and translation for x, y and z. These values would be determined by a surgeon for each operation, however for the sake of testing these values have been arbitrarily chosen. The relative position of the bone and respective implant have been selected to be indicative of approximate placement, and provide thorough testing, but are in no way surgically accurate.

6.2.3 Ideal Femur Test

6.2.3.1 Total Knee

To begin experimenting with the cuboid data structure and viability of the method the depth-intersection volume of an ideal femur mesh model and a total knee implant was calculated. Rendering the intersection depth without offsetting by the bone depth, which in this case would be the contour of the femur, gives a symmetrical volume representing the intersection depth of each pixel in the buffer.

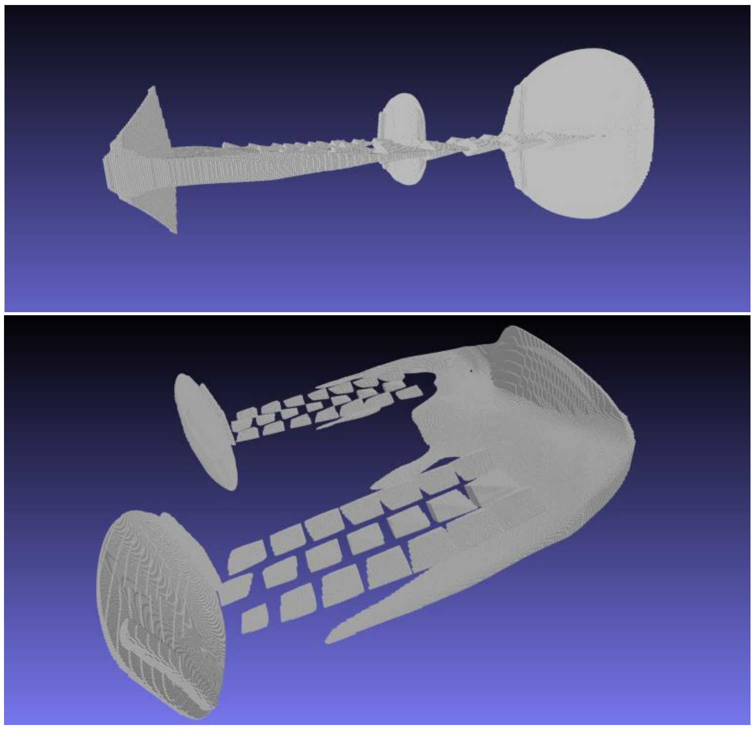


Figure 6-6: Total knee femur - incorrect intersection volume.

A symmetry is observed around the XY-Plane, this is because z or height information is lost when subtracting the femur depth map from the implant depth map. The height difference buffer only contains the cuboid z height information. To get the z-translation we need to offset by the femur depth buffer. As seen in Figure 6-7 the expected shape of the depth intersection volume is observed.

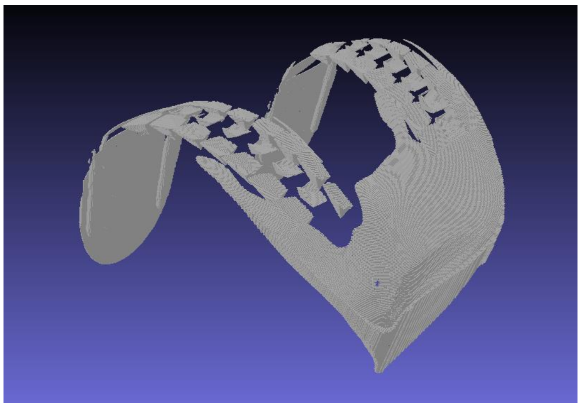


Figure 6-7: Total knee femur - correctly offset intersection volume.

A cuboid intersection volume tests were performed with the following results.

Distal camera resolution: height: 200 pixels, width: 277 pixels.

Pixel width: 0.28135368mm

Calculated volume: 9185.152mm^3

Intersection calculations: 85.1ms - cubes: 28206, triangles: 338472

Created K-D Tree with 225648 vertices **in:** 14.0133ms

Total cuboid intersection volume creation: 279.8946ms

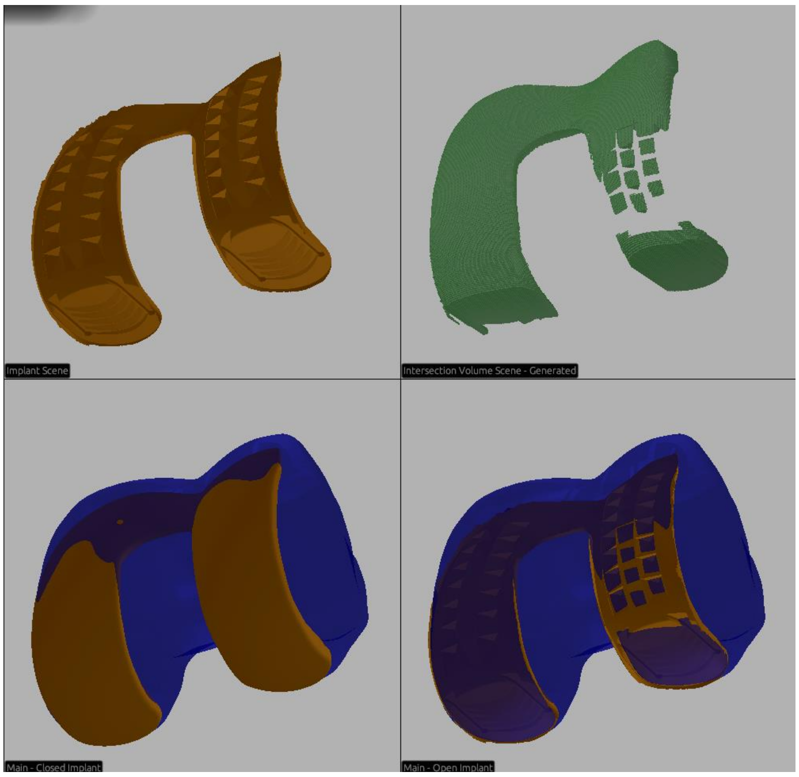


Figure 6-8: Femoral total knee - cuboid intersection volume.

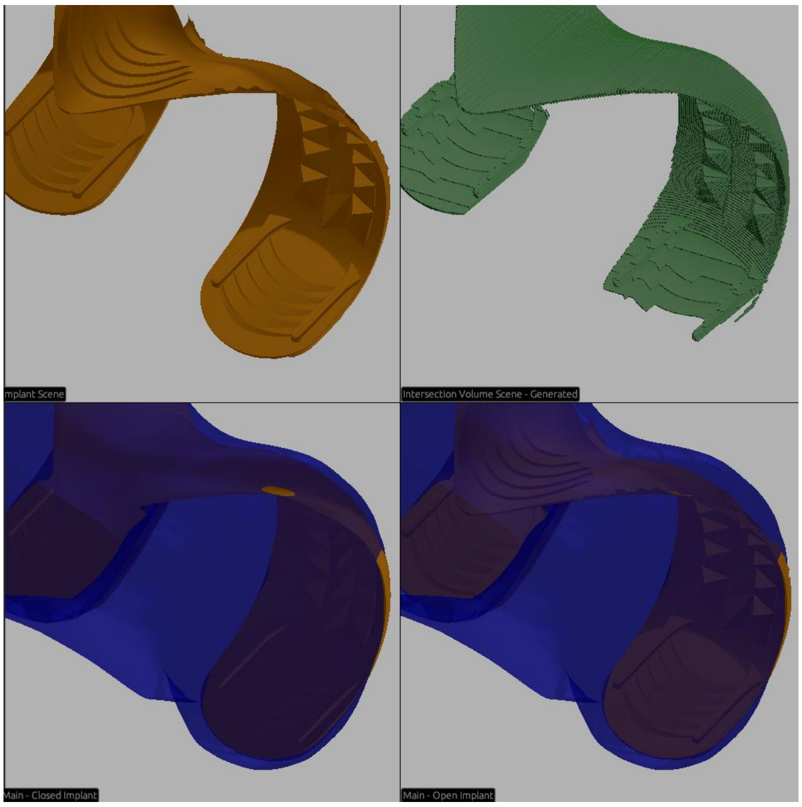


Figure 6-9: Femoral total knee - cuboid intersection volume.

Distal camera resolution: height: 1200 pixels, width: 1013 pixels.

Pixel width: 0.07689228mm

Calculated volume: 9809.835mm^3

Intersection calculations: 1270ms **-** cubes: 383493, triangles: 4601916

Created K-D Tree with 3067926 vertices **in:** 279.9476ms

Total cuboid intersection volume creation: 5199.4176ms

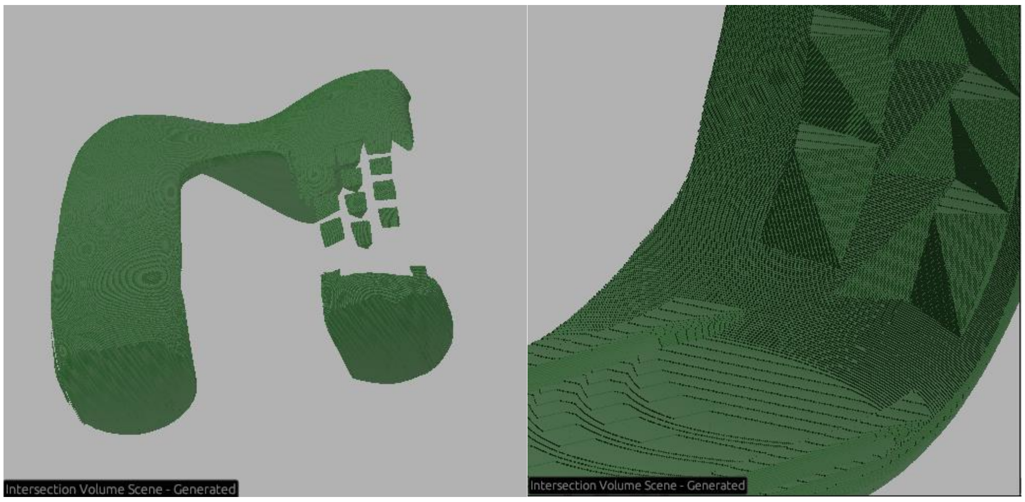

Intersection Volume Scene - Generated
Intersection Volume Scene - Generated

Figure 6-10: Femoral total knee - cuboid intersection volume.

6.2.3.2 Unilateral

Distal camera resolution: height: 1200 pixels, width: 890 pixels.

Pixel width: 0.09801464mm

Calculated volume: 2083.398mm^3

Intersection calculations: 231.5616ms - cubes: 73708, triangles: 884496

Created K-D Tree with 589652 vertices **in:** 38.3798ms

Total cuboid intersection volume creation: 734.0091ms

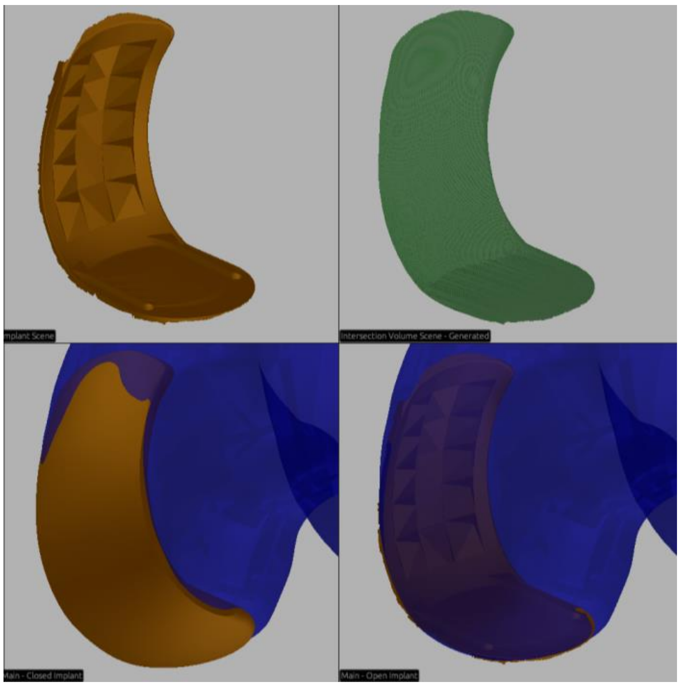


Figure 6-11: Femoral uni knee - cuboid intersection volume.

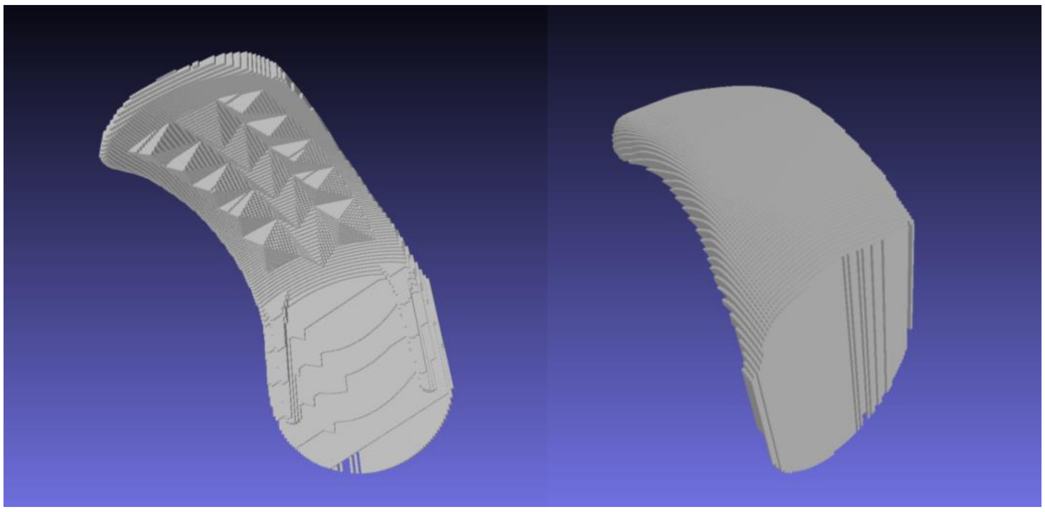


Figure 6-12: Femoral uni knee - cuboid intersection volume.

The information loss from one camera view is pronouced here, we expect to see the curve of the posterior section of the femur in the intersection volume. But there is just a straight drop off.

This is an inhererent fault with the depth intersection method. Because the implant is not visable in that section from a distal view. There for there is no difference between the femur and implant heights. So there is no intersection depth.

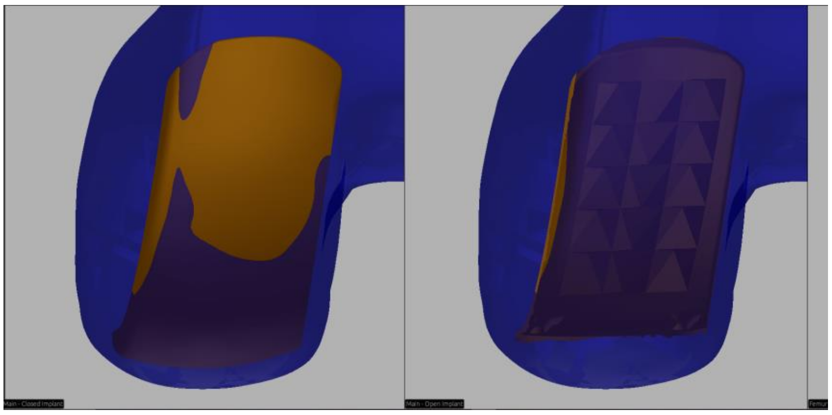


Figure 6-13: Femoral uni knee – femur and implant.

6.2.4 Tibia Test

Distal camera resolution: height: 1200 pixels, width: 1034 pixels.

Pixel width: 0.10457094mm

Calculated volume: 6589.126mm^3

Intersection calculations: 712.8163ms - cubes: 223929, triangles: 2687148

Created K-D Tree with 1791432 vertices **in:** 175.0044ms

Total cuboid intersection volume creation: 2527.6489ms

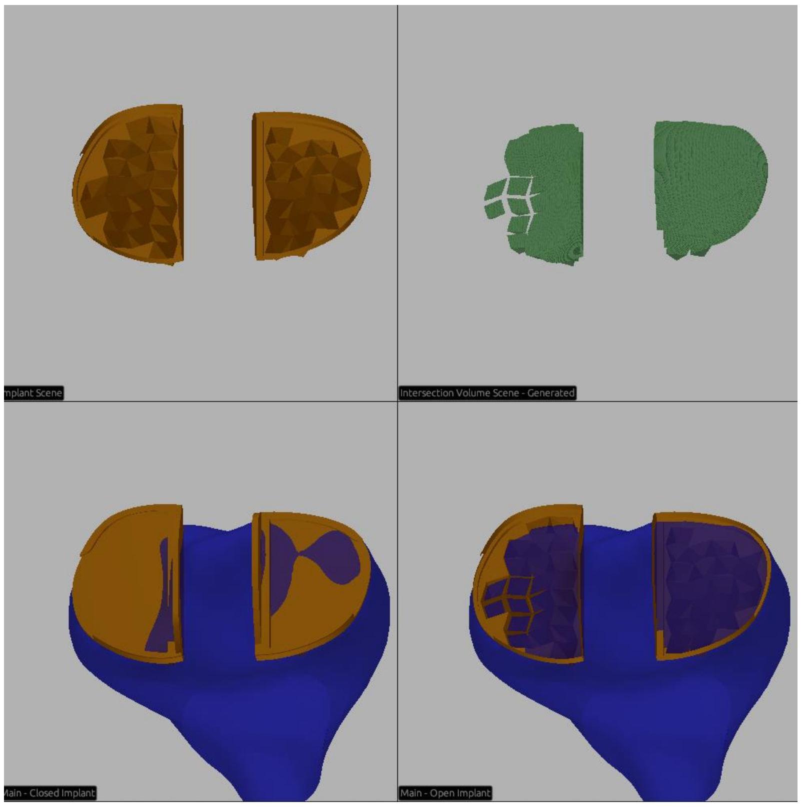


Figure 6-14: Tibial full knee - cuboid intersection volume.

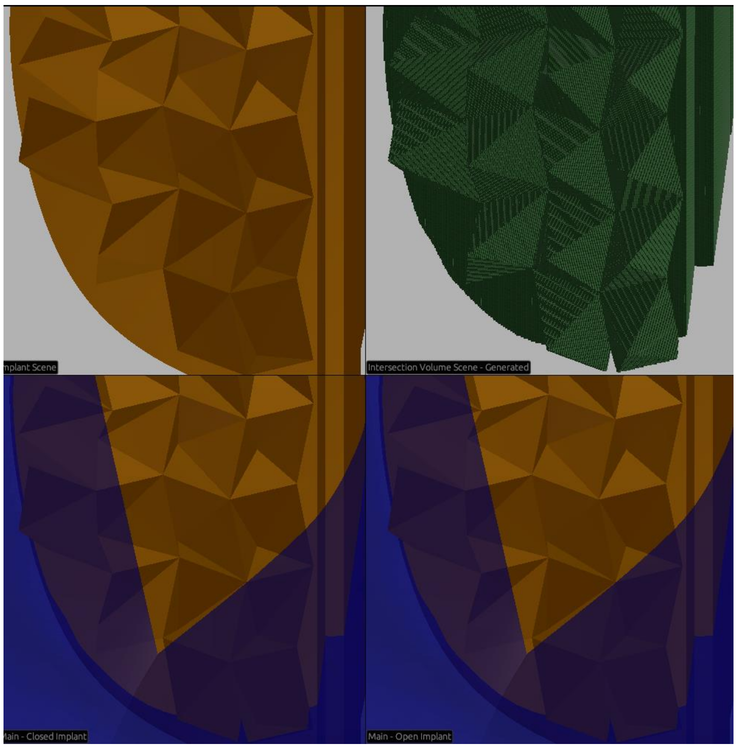


Figure 6-15: Tibial full knee - cuboid intersection volume.

6.2.4.1 CAD test

To determine the accuracy of the depth intersection method quantitatively and qualitatively a comparison was made between the generated cuboid tibial intersection volume and the intersection volume generated by Autodesk Inventor, a high-level CAD program.

Figure 6-16 shows the intersection volume in green and tibia in blue rendered at their positions in space. The intersection volume correctly follows the contour of the tibia.

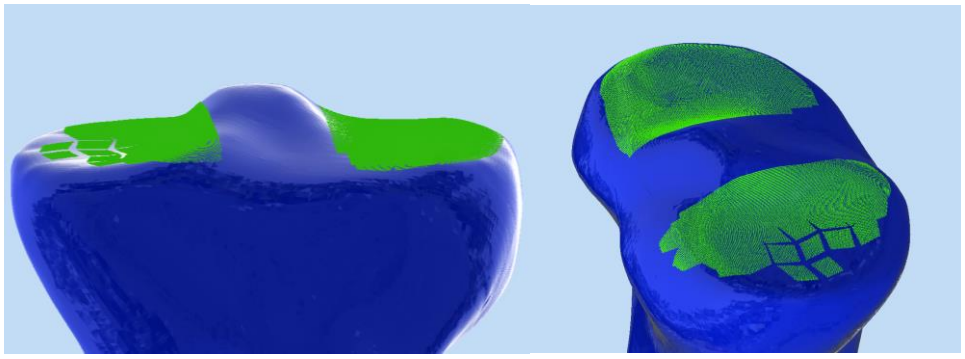


Figure 6-16: Visual comparison of the tibia and tibial intersection volume.

Checking the intersection volume against the tibial implant also verifies the correctness of the volume. Figure 6-16 displays the implant in yellow and the intersection volume in green.

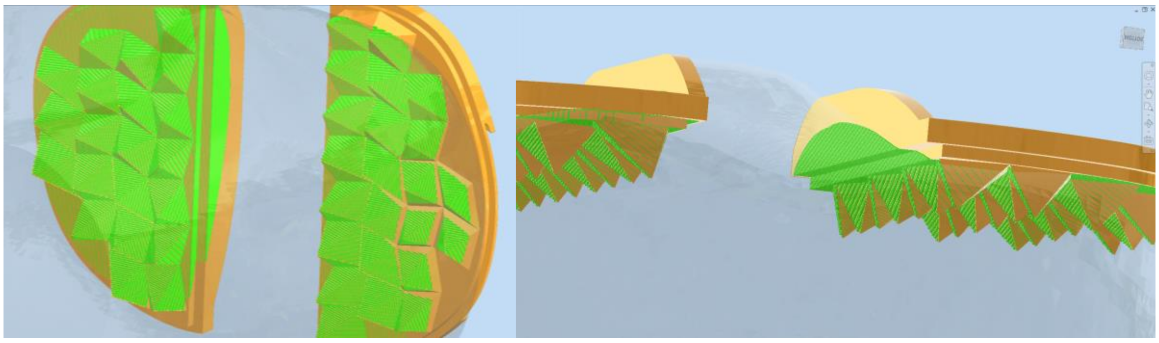


Figure 6-17: Visual comparison of the tibial implant and tibial intersection volume.

Rendering the tibia, implant, and intersection all at once in Figure 6-18 the intersection volume can be seen correctly outlining the bone that would be in the way for implant placement.

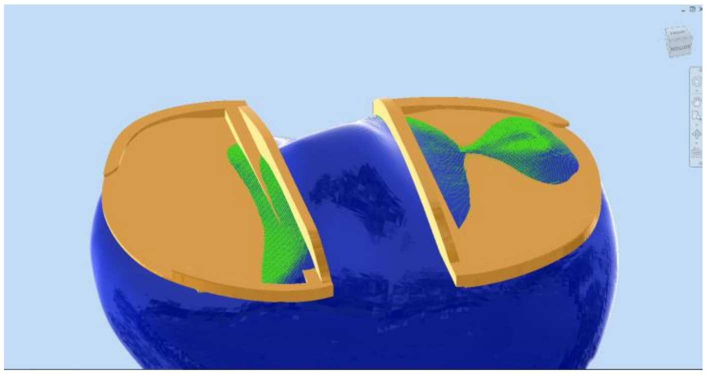


Figure 6-18: Visual comparison of the resected tibia and tibial intersection volume.

To test the capabilities of CAD programs for determining intersection volume the tibial implant model was manually extended upwards. This would allow a Boolean intersection to capture all the necessary tibial bone to be removed. Using the Autodesk Inventor interference analysis tool, the interference volume between the implant and tibia was calculated to be 8154.861mm^3, which took a few seconds to compute.

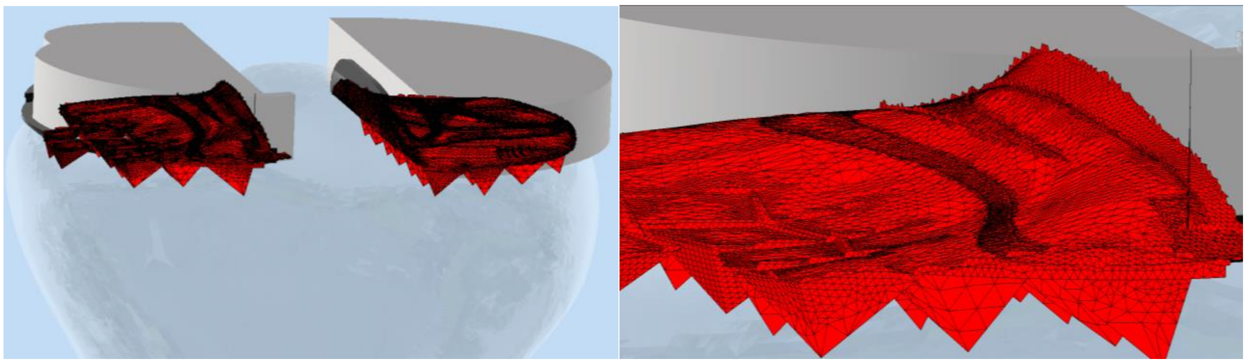
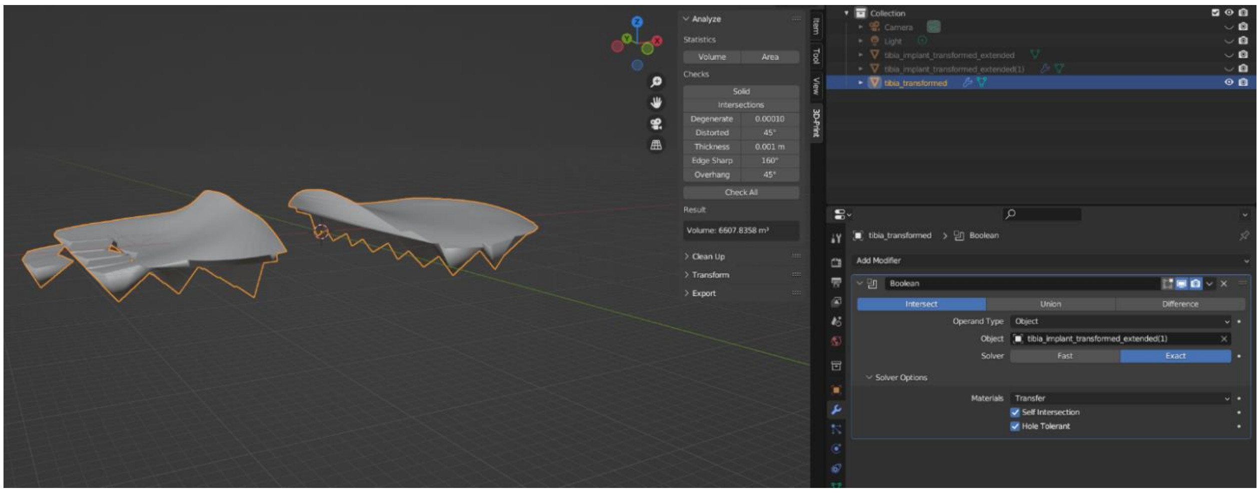


Figure 6-19: CAD program tibial interference analysis.

The calculated volume of 8154.861mm^3 is 24% larger than the cuboid depth-intersection volume calculation of 6589.126mm^3. Visually inspecting the CAD generated intersection volume shows a number of erroneous triangles, spikes and non-manifold edges. Comparing this to the visual accuracy of the cuboid depth-intersection volume, the CAD result seemed to be inaccurate. To verify the correct result a third calculation was computed in Blender. Blender is a 3D rendering and editing toolset. Using

Blender, a Boolean intersection on the tibia and extended tibial implant was created, then the volume of that intersection calculated, leading to a computed volume of 6607.836mm^3. This result again appeared visually accurate and was 0.28% off the computed cuboid volume, verifying the accuracy of the cuboid depth-intersection volume.



Analyze

Statistics: Volume | Area

Checks: Solid, Intersections

Degenerate	0.00010
Distorted	45°
Thickness	0.001 m
Edge Sharp	160°
Overhang	45°

Check All

Result: Volume: 6607.8358 m³

Collection
- Camera
- Light
- tibia_implant_transformed_extended
- tibia_implant_transformed_extended(1)
- tibia_transformed

tibia_transformed > Boolean

Boolean: [✓ Intersect] | Union | Difference

Operand Type: Object

Object: tibia_implant_transformed_extended(1)

Solver: Fast | [✓ Exact]

Solver Options

Materials: Transfer

✓ Self Intersection

✓ Hole Tolerant

Figure 6-20: Intersection volume calculation in blender.

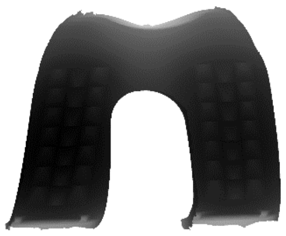
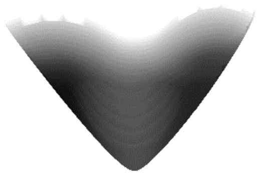
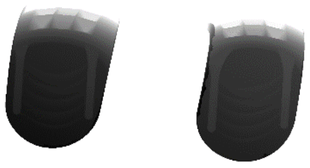
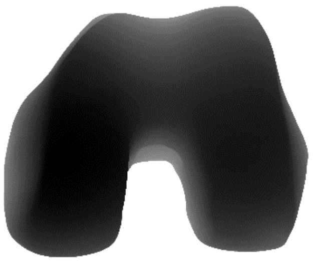
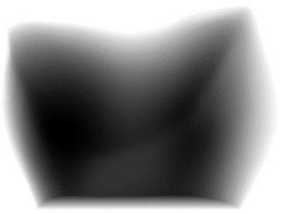
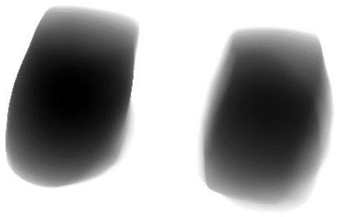
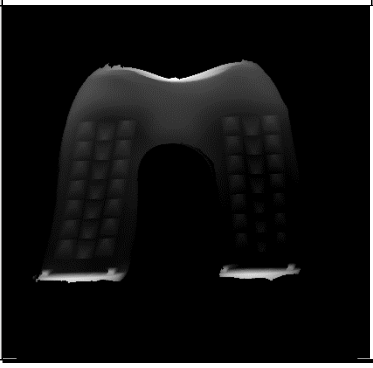
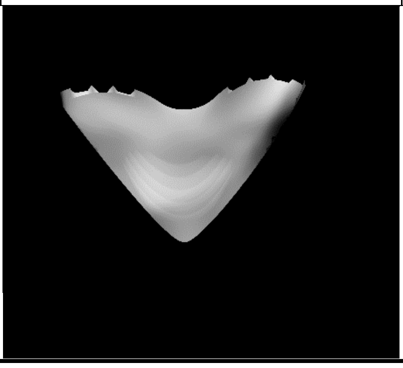
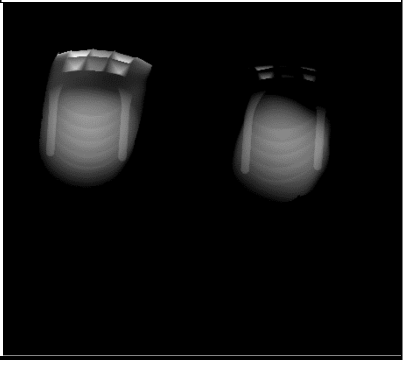
The slow speed and inaccuracy of Autodesk Inventor shows the difficulty CAD programs have when working with mesh objects. While extending the tibial implant to allow a Boolean mesh intersection to work was possible due to the implant's 'flat' shape this method is not practical for other implant shapes.

6.2.5 Multiple Views

To counteract the loss of information from one camera view a distal, anterior, and posterior view of the rendered scene were used. These are medical terminology for sides of the human body. The three cameras were generated mathematically and a depth map for each view created.

Table 6-1 shows the generated depth maps from each view for the implant and femur. These have been normalised for visual clarity and exported as a grayscale image. Lighter values represent being further away from the Z_NEAR plane of the camera frustum, and darker values are closer. The last row shows the difference between the two depth maps. Subtracting the femur depth from the implant depth.

Table 6-1: Depth buffers from each camera view for Implant, Femur and Subtracted Height Difference.

	Distal	**Anterior**	**Posterior**
Implant			
Femur			
Height Difference			

6.2.5.1 Positioning Cameras

The distal, posterior, and anterior cameras are dynamically generated in software depending on the model of bone and implant used. Firstly, the axis aligned bounding box (AABB) is calculated from the mesh points of both the bone and implant, shown in Figure 6-21. This is box that contains all points of both meshes, stored as a min (x, y, z) and max (x, y, z) point in space.

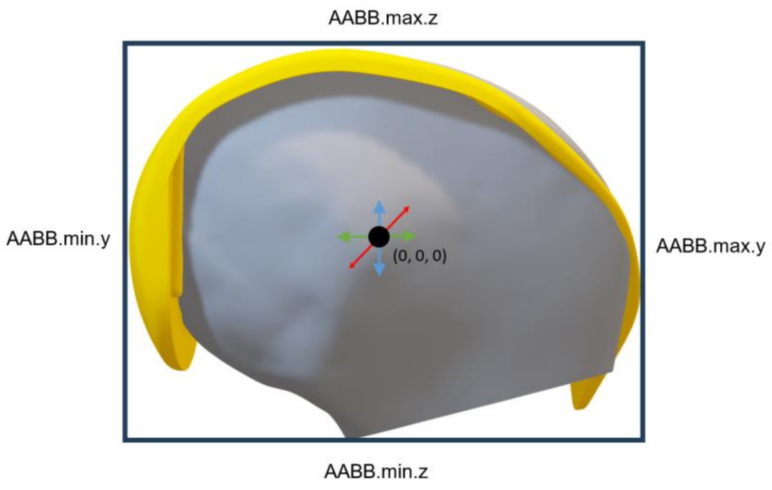


*Figure 6-21: **A**xis **A**ligned **B**ounding **B**ox of femur and implant.*

Then the three orthographic cameras are generated, all pointing towards the origin (0, 0, 0). The near plane of each camera is set to 0 (no offset from the camera position to the start of the frustum). Two user defined constants POSTERIOR_DISTANCE and ANTERIOR_DISTANCE in millimetres make up the depth of the posterior and anterior cameras. These values need to be changed depending on the bone type and implant design used. The height of the distal camera can be increased by a constant to force an overlap between the camera frustums. This is useful in determining correct alignment of the three generated intersection volumes. All other camera parameters are automatically calculated, shown in Figure 6-22.

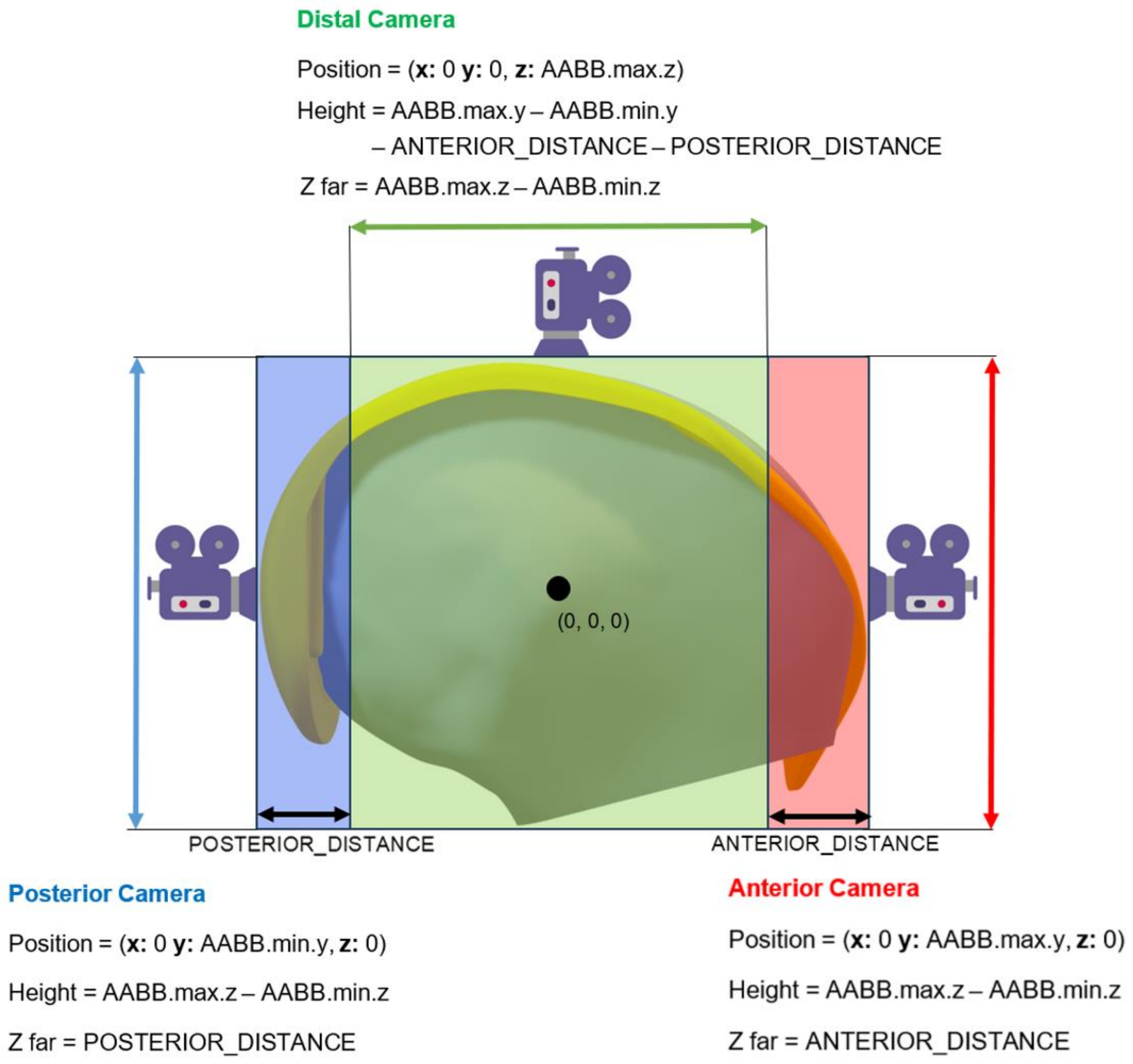

Distal Camera
Position = (**x:** 0 **y:** 0, **z:** AABB.max.z)
Height = AABB.max.y – AABB.min.y – ANTERIOR_DISTANCE – POSTERIOR_DISTANCE
Z far = AABB.max.z – AABB.min.z
(0, 0, 0)
POSTERIOR_DISTANCE
ANTERIOR_DISTANCE
Posterior Camera
Position = (**x:** 0 **y:** AABB.min.y, **z:** 0)
Height = AABB.max.z – AABB.min.z
Z far = POSTERIOR_DISTANCE
Anterior Camera
Position = (**x:** 0 **y:** AABB.max.y, **z:** 0)
Height = AABB.max.z – AABB.min.z
Z far = ANTERIOR_DISTANCE

Figure 6-22: Multi-view camera creation.

In this example the femur bone model has been shortened to remove unnecessary length in the z axis, when this isn't the case a Z_FAR_OFFSET constant could be subtracted from the distal Z_FAR parameter, to focus the frustum on the important section of the bone. The heights of the posterior and anterior cameras should then be adjusted accordingly.

6.2.5.2 Total Knee Multi View

Preforming a multi-camera view intersection test generates a more accurate intersection volume.

Distal camera resolution: height: 500 pixels, width: 732 pixels.

Distal pixel width: 0.11923511mm

Calculated distal volume: 9812.825mm^3

Distal intersection calculations: 1083.102ms - cubes: 159501, triangles: 1914012

Created distal K-D Tree with 1276002 vertices **in:** 107.8119ms

Anterior camera resolution: height: 500 pixels, width: 648 pixels.

Anterior pixel width: 0.13463803mm

Calculated anterior volume: 4562.182mm^3

Anterior intersection calculations: 422.5088ms - cubes: 58879, triangles: 706548

Created anterior K-D Tree with 471032 vertices **in:** 61.746ms

Posterior camera resolution: height: 500 pixels, width: 648 pixels.

Posterior pixel width: 0.13463803mm

Calculated posterior volume: 2668.258mm^3

Posterior intersection calculations: 353.475ms - cubes: 40797, triangles: 489564

Created posterior K-D Tree with 326376 vertices **in:** 69.8723ms

Total volume: 17043.265mm^3

Total cuboid intersection volume creation: 2893.2208ms

As the three camera view volumes were calculated in parallel the total computation time is not that much different to the single view femoral volume. The calculated total volume is much higher than the single view femoral volume, the multi-view volume is enlarged by the overlap between the sections. Without an overlap the shape of the intersection volume becomes inaccurate, as some of the implant features require multiple views to accurately map.

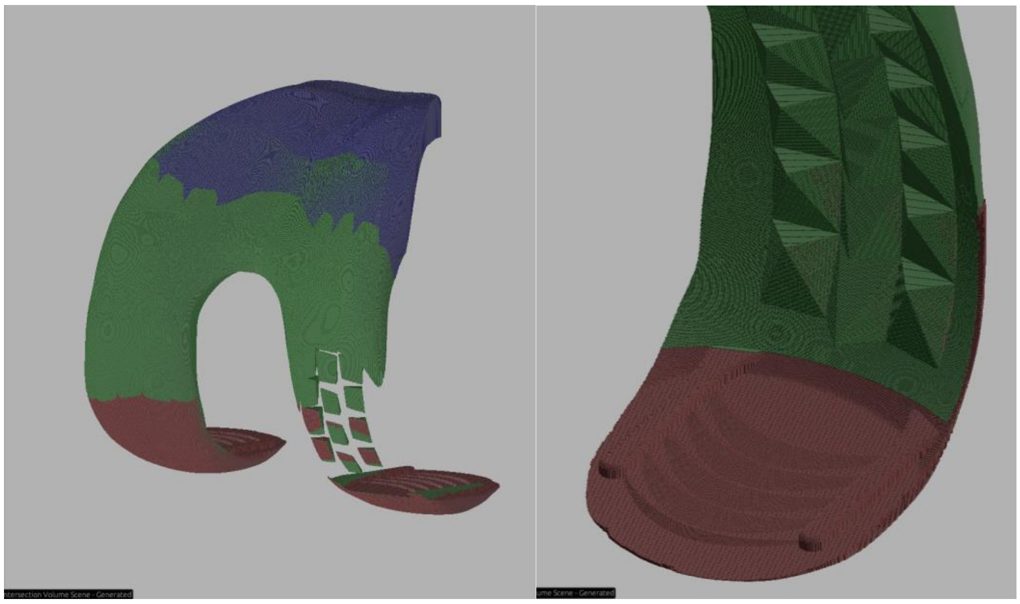


Figure 6-23: Femoral total knee - multi-view cuboid intersection volume.

The posterior and anterior sections of the intersection volume show much more detail and are visually more accurate to the expected intersection volume. Comparing the intersection volume against the implant shows, the implant face on the inner side of the volume and the contour of the femur on the outer side of the volume.

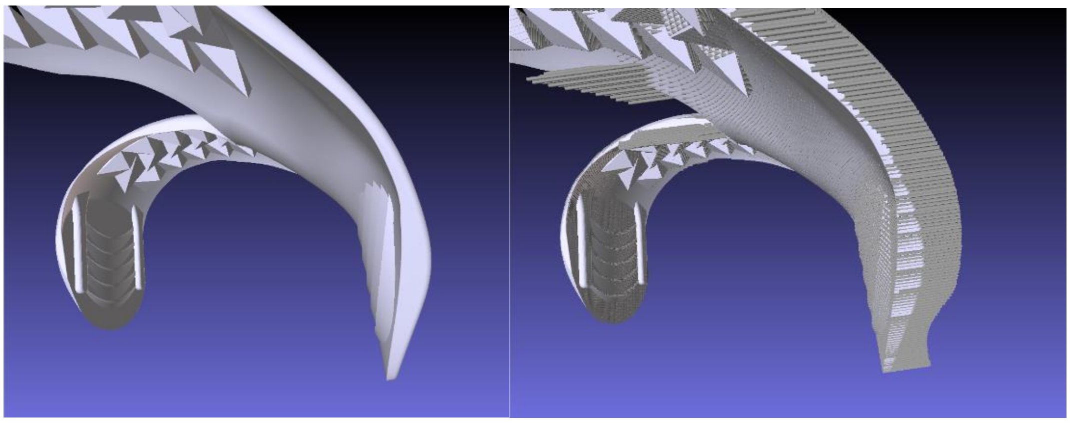


Figure 6-24: Femoral implant model (left) – with overlayed depth intersection volume (right).

6.2.5.3 Unilateral Multi View

Preforming a multi camera view intersection volume on a unilateral implant and femur we get a more accurate volume. The details on the posterior grooves can be seen and the posterior curve of the femur.

Distal camera resolution: height: 862 pixels, width: 732 pixels.

Distal pixel width: 0.10123512mm

Calculated distal volume: 1421.480mm^3

Distal intersection calculations: 288.6037ms - cubes: 61539, triangles: 738468

Created distal K-D Tree with 492310 vertices **in:** 34.5264ms

Anterior camera resolution: height: 500 pixels, width: 648 pixels.

Anterior pixel width: 0.13463803mm

Calculated anterior volume: 0.0mm^3

Anterior intersection calculations: 7.056ms - cubes: 0, triangles: 0

Created anterior K-D Tree with 0 vertices **in:** nil

Posterior camera resolution: height: 500 pixels, width: 648 pixels.

Posterior pixel width: 0.13463803mm

Calculated posterior volume: 1397.835mm^3

Posterior intersection calculations: 165.1304ms - cubes: 29063, triangles: 348756

Created posterior K-D Tree with 232504 vertices **in:** 29.6888ms

Total volume: 2819.315mm^3

Total cuboid intersection volume creation: 754.0052ms

The volume calculation is higher, as more femur has been determined to be intersecting implant placement. As with the total knee calculation this volume is inflated from the actual value due to the overlap of cuboids.

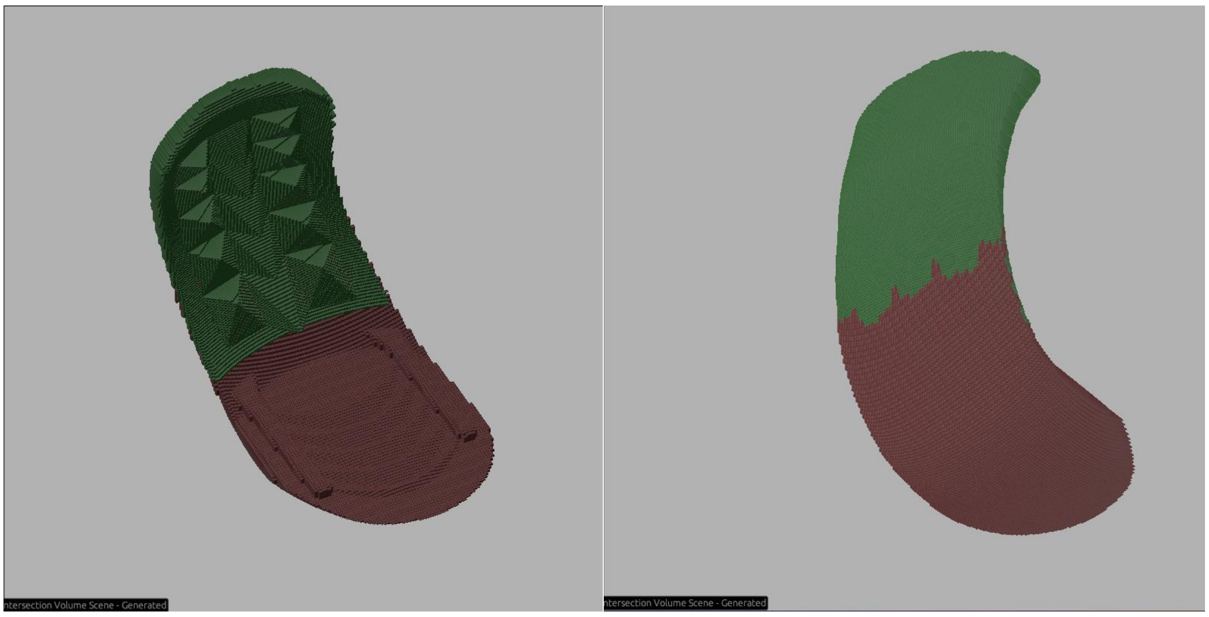

ntersection Volume Scene - Generated
ntersection Volume Scene - Generated

Figure 6-25: Femoral uni knee - multi-view cuboid intersection volume.

7 OCTREE STRUCTURE

To overcome the limitations of the cuboid data structure, a switch was made to an octree data structure. Octrees are tree data structures that can partition points in space by recursively subdividing a cube into eight smaller cubes called octants. In this case the octree is used to represent a volume. Each octant can either be empty, filled, or be further subdivided into eight smaller octants. This hierarchical structure allows octrees to efficiently represent both sparse and dense data.

The octree was generated using the graphics depth buffers with a desired resolution of 20 to 40 microns, meaning that the smallest cube in the octree represents a 20-micron cubed area.

Octrees have several advantages over other data structures, such as the cuboids, for representing and manipulating volumes. Octrees can reduce storage requirements, provide faster point lookups, and enable more efficient simulations. However, they also come with new problems to solve, such as handling edge cases and maintaining the octree structure during dynamic updates.

For example, point lookups in the octree are in the order of 10s of nanoseconds, compared to 10s of microseconds in the old cuboid data structure. This is a 1000x speed increase, which allows us to simulate laser ablation point by point quickly. To find a point in an octree, we simply start at the root node and traverse the tree until we reach an octant that contains the point. This process is very efficient because the octree is a hierarchical structure.

Figure 7-1 shows a simple octree volume, the empty cubes are “empty” leaf octants, and the green cubes are “filled” leaf octants.

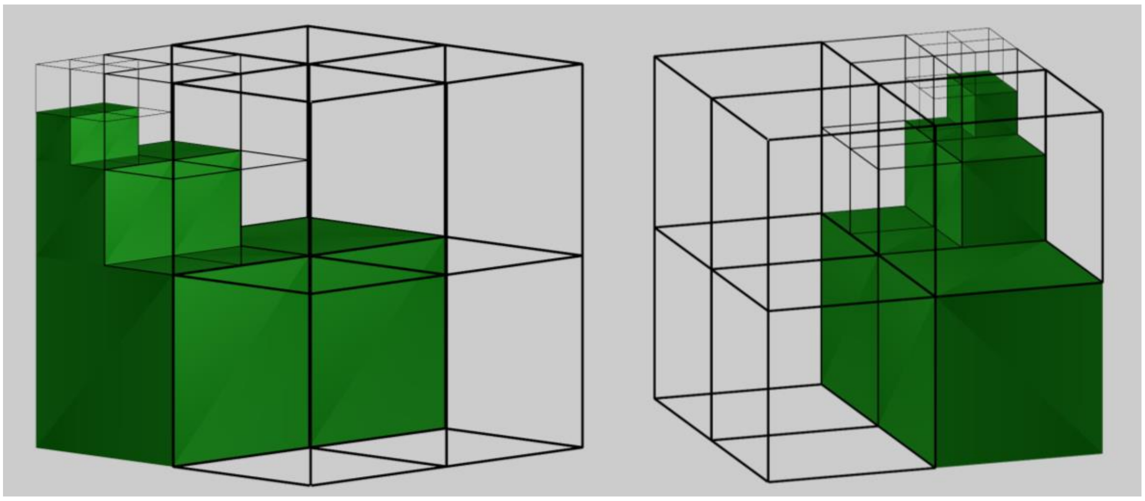


Figure 7-1: Simple octree volume

7.1 Creating Octree

To create the intersection volume represented as an octree the GPU depth buffers were again used.

- Calculate pixel width and height from camera.
- Map intersection depth map to a zmin and zmax.
- Calculate bounding box of depth map.
- Set resolution of octree = pixel width.
- Calculate centre and scale of octree, scale must be a power of 2. Smallest power of 2 that makes the octree larger than the bounding box of depth map.
- Offset octree centre so that depth map pixels line up with octree octants. (this adds 0.5 * resolution error)

```
Algorithm: create_octree(camera_info, depth_info) -> octree
    pixel_width = (camera.max.x – camera.min.x) / depth_info.width

    FOR EACH depth in depth_info:
        z_min = camera.pos.z – implant_depth
        z_max = camera.pos.z – bone_depth
    END FOR

    // Calculate intersection volume AABB from depth_info
    FOR EACH depth in depth_info:
        IF intersection_depth = 0 THEN
            CONTINUE
        END IF
        AABB.min = min(AABB.min, pixel_position)
        AABB.max = max(AABB.max, pixel_position)
    END FOR

    // Create octree
    octree.center = (AABB.min + AABB.max) / 2
    octree.resolution = pixel_width
    // Scale must be a power of 2
     scale = (AABB.max – AABB.min) / octree.resolution
     // Smallest integer power of 2 greater than scale
    scale = 2^(log$_2$(scale) + 1)
    octree.scale = scale

    // Split octree
    split_octree(octree, depth_info)
RETURN octree
END ALGORITHM
```

Algorithm 7-1: Octree creation from depth map.

```
Algorithm: split_octree(octree, depth_info) -> split_octree
      // Initialize a stack with the root octant, its center, and its scale
      PUSH (octree.root, octree.center, scale) to stack
      WHILE stack is not empty:
          POP (octant, center, scale) from stack
          // Calculate octant AABB
          octant_min.x = center.x - (scale * resolution / 2)
          octant_max.x = center.x + (scale * resolution / 2)
          . . . // Do same for y and z

          //Find depth buffer indices
          i_min = (octant_min.x - depth_info.AABB.min.x) / resolution
          i_max = (octant_max.x - depth_info.AABB.min.x) / resolution
          . . . // Do same for y_min, y_max

          FOR EACH i in i_min to i_max:
              FOR EACH j in y_min to y_max:
                  z_min, z_max = depth_info[j][i]
                  IF z_min or z_max intersect octant & scale != 1 THEN
                        split octant
                        FOR EACH child in octant.children:
                              PUSH child to stack
                        END FOR
                        check next octant
                  ELSE IF octant is within z_min to z_max THEN
                        Fill octant
                  ELSE
                        Empty octant
                  END IF
            END FOR
          END FOR
      END WHILE
RETURN split_octree
END ALGORITHM
```

Algorithm 7-2: Octree splitting.

After splitting the octree is potentially unoptimized so an optimisation algorithm run over the octree, as described in section 7.2.2. Making the octree the same resolution as the depth buffer and aligning them reduces artifacts and massively reduce octree creation time, as index lookups have a one-to-one mapping. Splitting time was reduced by splitting each root child octant in parallel, for a 2000px by 2800px depth map the total octree creation time was approximately 1 second.

7.2 Octree Operations

7.2.1 Sphere Lookup

```
Algorithm: within_sphere(octree, sphere) -> octants
      // Initialize a stack with the root octant, its center, and its scale
      PUSH (octree_root, octree_center, scale) to stack
      // Initialize an empty list to store octants within the sphere
      octants_within_sphere = empty list
      WHILE stack is not empty:
          POP (octant, center, scale) from stack
          IF octant is Filled THEN
              // Check if the octant is within the sphere
              IF center is in sphere THEN
                  ADD octant to octants_within_sphere
              END IF
          ELSE IF octant is a Branch THEN
              FOR EACH child in octant.children:
                  IF child intersects with sphere THEN
                      PUSH child to stack
                  END IF
              END FOR
          END IF
      END WHILE
RETURN octants_within_sphere
END ALGORITHM
```

Algorithm 7-3: Octree sphere lookup.

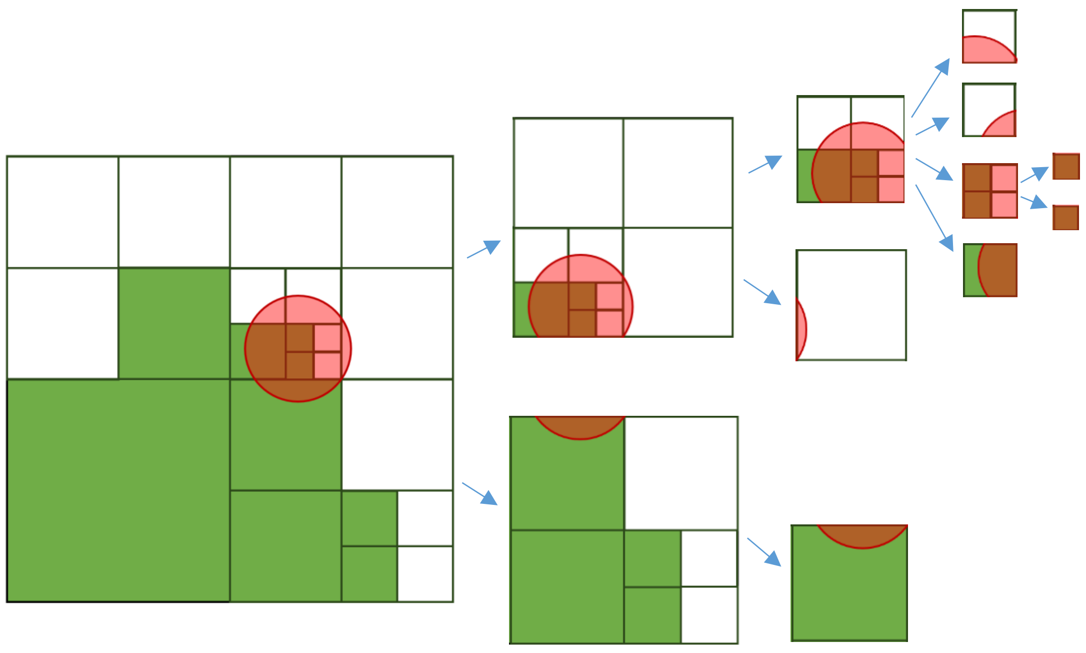


Figure 7-2: Octree sphere lookup.

7.2.2 Octree Optimisation

The creation of the octree and operations run on the octree such as ablation simulation can lead to an unoptimized octree. Meaning the octree contains octants that are a branch, but all 8 of its children are either all full or all empty. This means the octant can be collapsed down, reducing overall octree size, and reducing lookup times.

The optimisation algorithm starts with the root octant and then recursively calls the optimisation function on each child octant. Afterwards it checks if the octant can be collapsed down.

```
Algorithm: optimise(octant)
    IF octant is a Branch THEN
        FOR EACH child in octant.children DO
            optimise(child)
        END FOR
        IF octant children are all Filled THEN
             fill octant
         ELSE IF octant children are all Empty THEN
             empty octant
         END IF
    END IF
END ALGORITHM
```

Algorithm 7-4: Octree optimisation.

7.2.3 Volume Calculation

To calculate the volume of the octree, every octant is traversed, and all filled octants have their volumes added. The algorithm starts with the root octant and adds any child octants to a stack.

```
Algorithm: calculate_volume(octant, octree_resolution) -> float:
    volume = 0.0
    // Initialize a stack with the root octant and its scale
    PUSH (octant, scale) to stack

    WHILE stack is not empty:
        POP (octant, scale) from stack
        IF octant is Filled THEN
            volume = volume + (scale * octree_resolution)^3
        ELSE IF octant is a Branch THEN
            FOR EACH child in octant.children:
                PUSH (octant, scale/2) to stack
            END FOR
        END IF
    // Return the computed volume
    RETURN volume
END ALGORITHM
```

Algorithm 7-5: Octree volume calculation.

Large errors were found when using 32-bit floats for the volume calculations. Testing on a generated octree of size: 24,409,169 octants, the calculated volume using 32-bit floating point values was 8202.635mm^3 but with 64-bit floats the volume was calculated to be 8578.139mm^3 a 4.6% error. An inconsistency is observed when optimising the octree, this reduces the total number of octants, but volume of the octree should not change. After optimisation the new size was 18,028,457 octants, the volume calculated using 64-bit floats was again 8578.139mm^3 only differing at the 6th decimal place. The volume with 32-bit floats was 8314.06mm^3, different from the original 32-bit value.

This inconsistency is caused by the cumulation of floating-point errors when adding the volumes of millions of octants. Using 64-bit floats for the volume of each octant significantly reduces the floating-point error of each octant and hence the total volume error.

7.3 Octree Results

The experiments conducted aimed to evaluate the octree structure using models of an ideal femur, combined with both total and unilateral implants. Additionally, an ideal tibia and a real CT scan of a tibia were used. This section presents the generation time, resolution, scale, and octree size breakdown both before and after optimization for each model set.

7.3.1 Femoral Total

Using the ideal femur model from 6.2.3 and a total knee implant model, the following octree intersection volume was generated.

Distal camera resolution: height: 2000 pixels, width: 2769 pixels.

Octree centre point: (x: -35.911274, y: -29.805553, z: 22.475529)

Octree scale: 4096, **resolution**: 0.028135369mm

Intersection calculations: 216.142ms

Octree splitting: 753.2316ms

Distal octree created in 969.3736ms

Table 7-1: Femoral total knee – unoptimized octree breakdown.

Octant Scale	Filled Octants	Empty Octants
2048		4
1024		18
512		71
256		163
128		605
64	335	2,289
32	3,588	8,735
16	20,626	30,140
8	93,564	114,093
4	377,269	430,639
2	1,481,340	1,598,352
1	11,380,118	5,816,074

This led to a total of 24,409,169 octants (leaf + branch), the octree optimisation algorithm was run taking 0.071204 seconds, the updated tree breakdown is shown in Table 7-2.

Table 7-2: Femoral total knee – optimized octree breakdown.

Octant Scale	Filled Octants	Empty Octants
2048		4
1024		18
512		71
256		163
128		605
64	342	2,289
32	3,653	8,735
16	20,882	30,140
8	94,358	114,093
4	379,275	430,639
2	1,486,745	1,598,352
1	5,788,462	5,816,074

The optimised octree has a total of 18,028,457 octants (leaf + branch) and a calculated total filled volume of 8578.139mm^3.

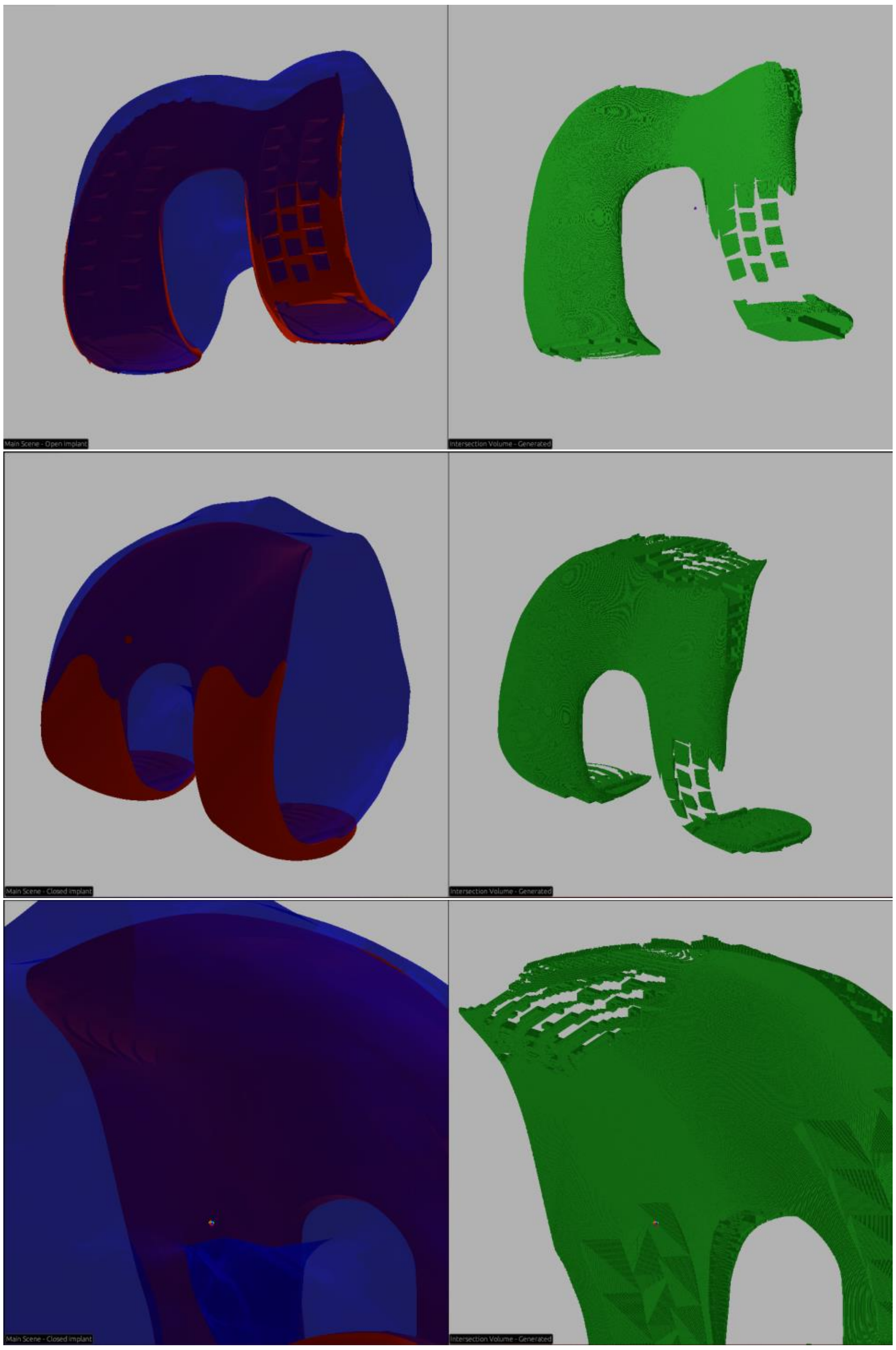

Main Scene - Open Implant
Intersection Volume - Generated
Main Scene - Closed Implant
Intersection Volume - Generated
Main Scene - Closed Implant
Intersection Volume - Generated

Figure 7-3: Femoral total knee - octree intersection volume.

The central area of the octree intersection volume is visually accurate and aligns with the femur and implant in CAD software. The vertical sections of the implant have poor octree quality due to the obfuscation and discretisation of information as described in 6.2.5.

7.3.1.1 Multi-View

While multiple camera views are necessary for an accurate intersection volume as outlined in section 6.2.5, stitching together multiple octrees with different translations and rotations that are not necessarily axis aligned, is a complex problem. Due to the timeframe of the paper the remainder of octree testing will be done only using a single distal camera view.

Stitching multiple octrees generated from different cameras would require transforming each octree to their correct relative positions in space and then having an algorithm combine all the octree information into a single octree. An unoptimized workaround could be to treat each octree as separate and loop through all octrees when preforming lookups and operations. This solution was believed to be not viable for the purposes of the JRPA and was not tested. A multi-view octree volume was generated, with the information from each view distinctly coloured. The artifacts and misalignment issues are heavily apparent.

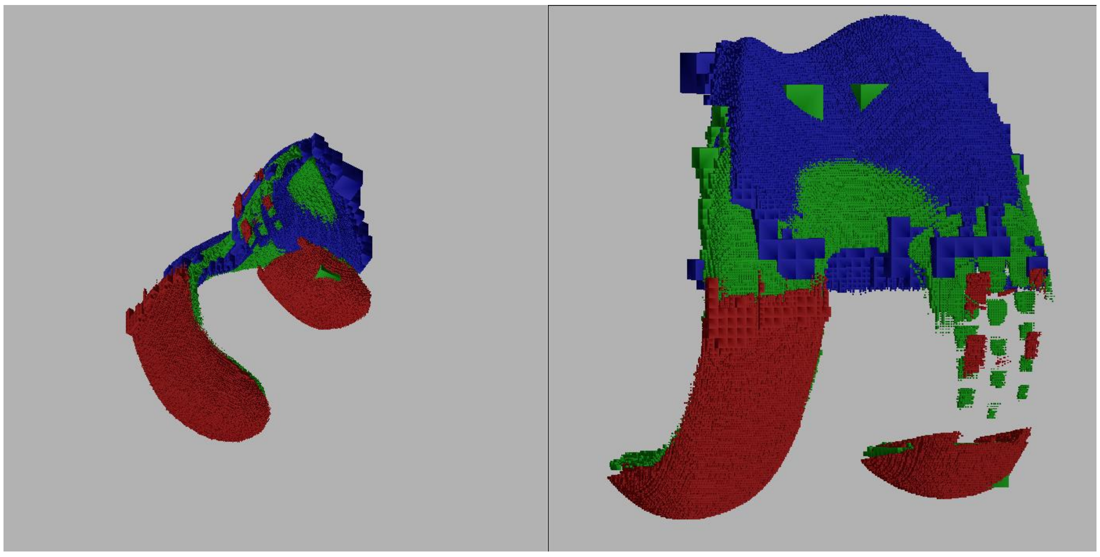


Figure 7-4: Multi-view octree intersection volume.

7.3.2 Femoral Uni

Using the same ideal femur model but with a unilateral implant model, the following octree intersection volume was generated.

Distal camera resolution: height: 2000 pixels, width: 2424 pixels.

Octree centre point: (x: -22.221607, y: -11.391987, z: 36.084503)

Octree scale: 2048, **resolution:** 0.032135367mm

Intersection calculations: 266.5598ms

Octree splitting took: 252.2733ms

Distal octree created in 518.8331ms

Table 7-3: Femoral uni knee – unoptimized octree breakdown.

Octant Scale	Filled Octants	Empty Octants
1024		4
512		17
256		85
128		149

Octant Scale	Filled Octants	Empty Octants
64	2	441
32	395	1,797
16	4,055	6,362
8	20,811	24,718
4	84,604	94,638
2	328,329	348,288
1	2,619,795	1,286,565

This led to a total of 5,509,777 octants (leaf + branch), the octree optimisation algorithm was run taking 0.019779 seconds, the updated tree breakdown is shown in Table 7-4.

Table 7-4: Femoral uni knee – optimized octree breakdown.

Octant Scale	Filled Octants	Empty Octants
1024		4
512		17
256		85
128		149
64	3	441
32	411	1,797
16	4,167	6,362
8	21,040	24,718
4	84,310	94,638
2	327,901	348,288
1	1,279,603	1,286,565

The optimised octree has a total of 3,977,713 octants (leaf + branch) and a calculated total filled volume of 1705.516mm^3.

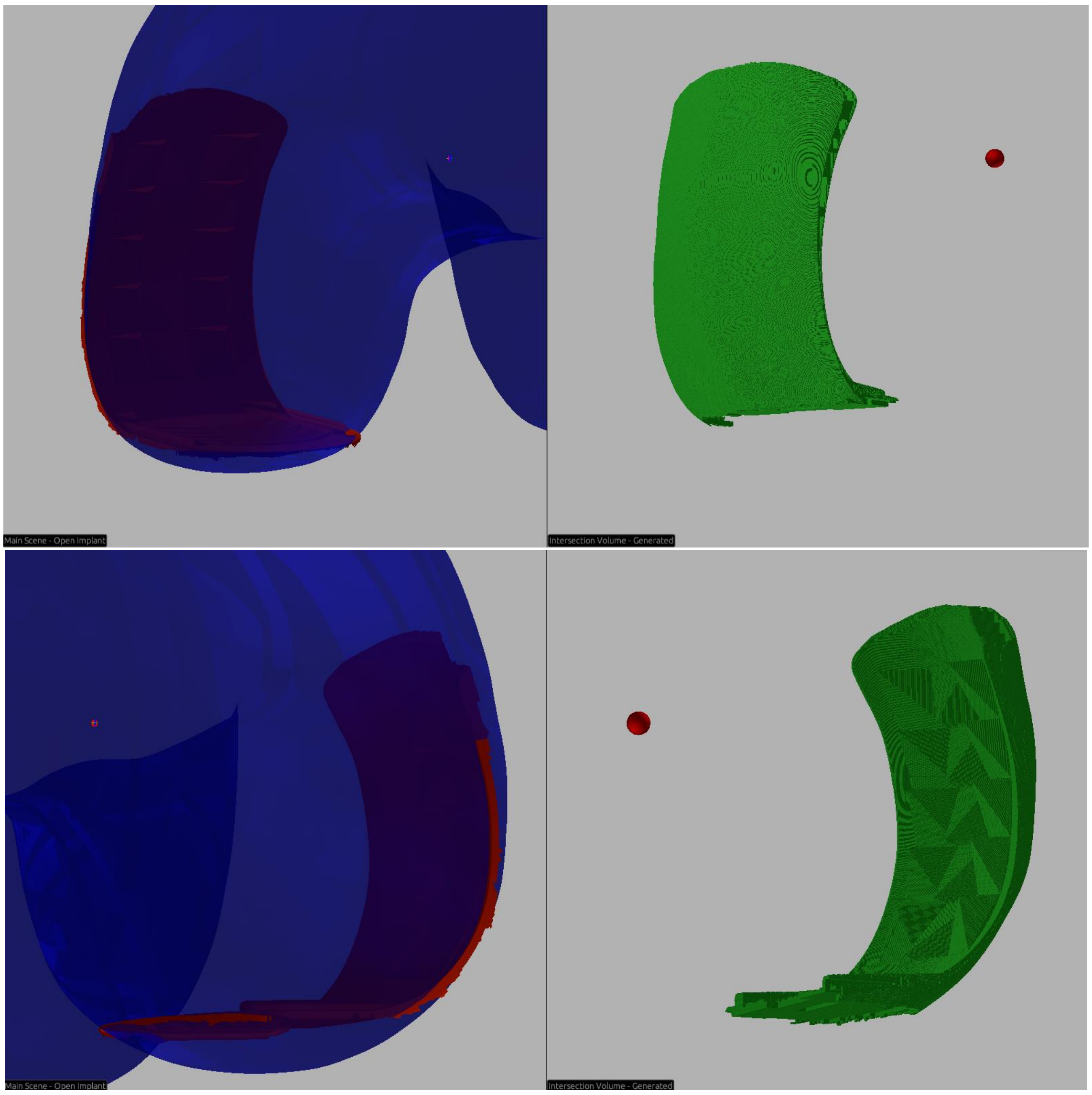

Main Scene - Open Implant
Intersection Volume - Generated
Main Scene - Open Implant
Intersection Volume - Generated

Figure 7-5: Femoral uni knee - octree intersection volume.

7.3.3 Ideal Tibial Total

Using the ideal tibia model from section 6.2.4 along with a total knee implant model, the following octree intersection volume was generated.

Distal camera resolution: height: 2000 pixels, width: 2974 pixels

Octree centre point: (x: -2.9597836, y: 4.26832, z: 17.246613)

Octree scale: 4096, **resolution:** 0.031155618mm

Intersection calculations: 297.0758ms

Octree splitting: 800.5891ms

Distal octree created in 1097.6649ms

Table 7-5: Tibial total knee – unoptimized octree breakdown.

Octant Scale	Filled Octants	Empty Octants
2048		4
1024		22
512		40
256		218
128		325
64	77	1,853
32	2,195	5,075
16	13,847	21,446
8	63,740	77,887
4	273,272	292,994
2	1,109,796	1,168,771
1	9,425,477	4,572,163

This led to a total of 19,461,945 octants (leaf + branch), the octree optimisation algorithm was run taking 0.049510 seconds, the updated tree breakdown is shown in Table 7-6.

Table 7-6: Tibial total knee – optimized octree breakdown.

Octant Scale	Filled Octants	Empty Octants
2048		4
1024		22
512		40
256		218
128		325
64	78	1,853
32	2,316	5,075
16	13,833	21,446
8	64,223	77,887
4	275,942	292,994
2	1,146,226	1,168,771
1	4,546,133	4,572,163

The optimised octree has a total of 13,930,913 octants (leaf + branch) and a calculated total filled volume of 6570.244mm^3. Compared to 6589.126mm^3 for the cuboid. This result also verifies the findings in section 6.2.4.1.

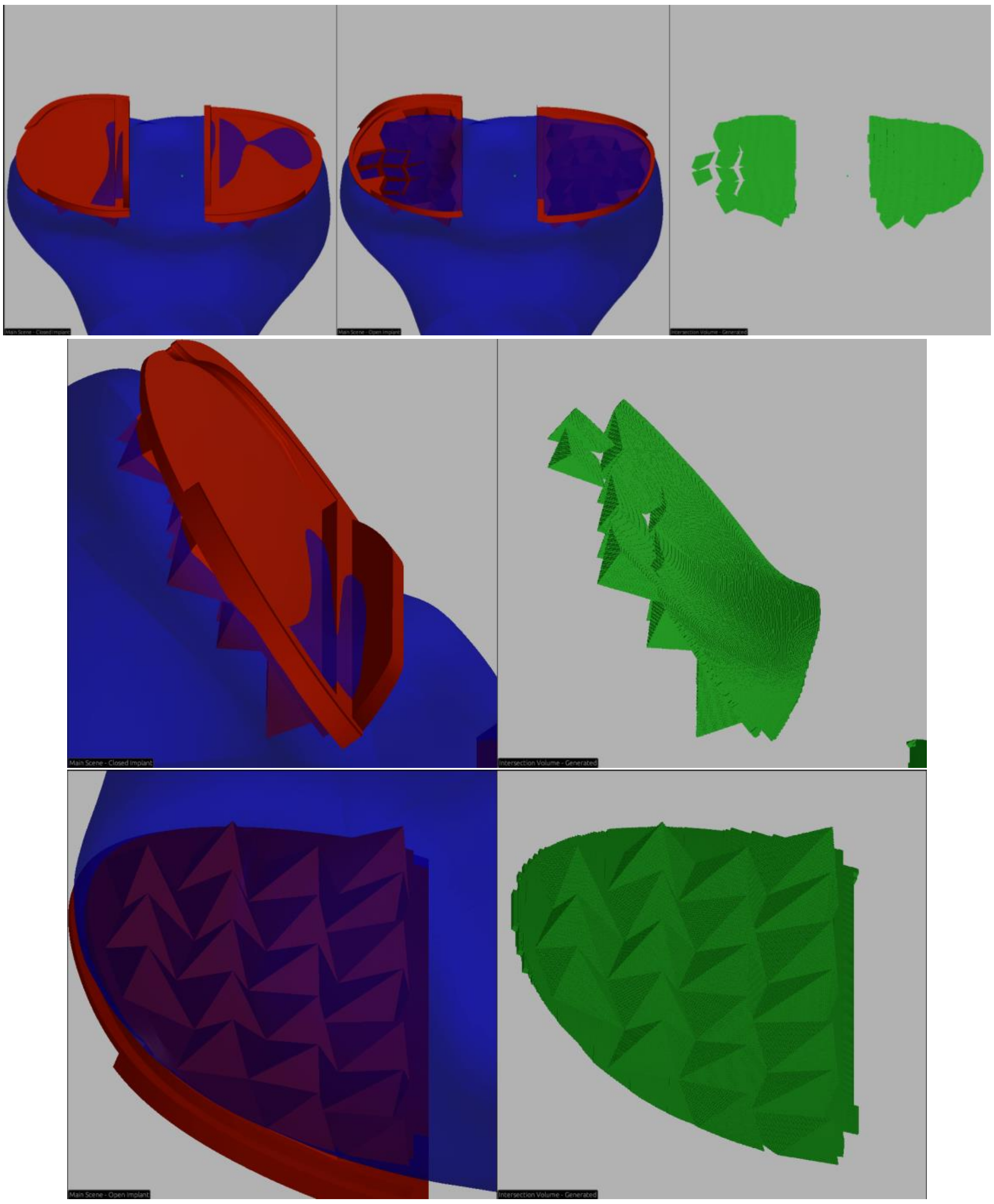

Main Scene - Closed Implant
Main Scene - Open Implant
Intersection Volume - Generated
Main Scene - Closed Implant
Intersection Volume - Generated
Main Scene - Open Implant
Intersection Volume - Generated

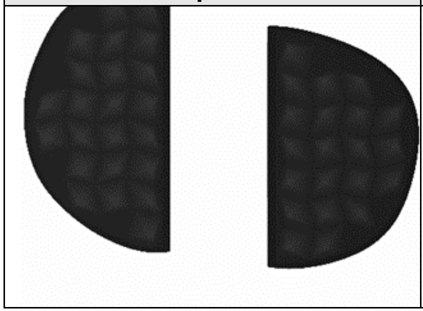
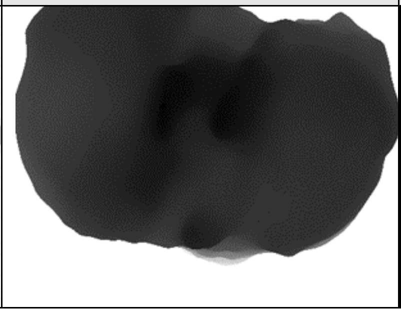
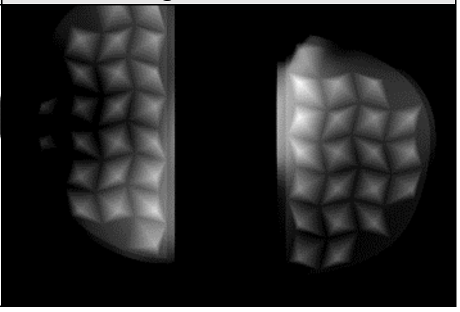
Figure 7-6: Tibial total knee - octree intersection volume.

Due to the flatter geometry of the tibial implant the distal camera is sufficient to capture the depth intersection volume. The result is much cleaner around the edges than the femoral octree volumes, as they are sharper.

7.3.4 Real Tibia Total

Next a CT scan of a real tibia was used along with a total knee implant, the implant was scaled down 10% from the ideal tibia test to accommodate a smaller tibia size.

Table 7-7: Tibia distal depth buffers.

Implant	Tibia	Height Difference
		

Distal camera resolution: height: 2000 pixels, width: 2932 pixels.

Octree centre point: (x: -1.832983, y: 4.3533344, z: 20.978971)

Octree scale: 4096, **resolution:** 0.030549727mm

Intersection calculations: 320.0779ms

Octree splitting: 719.1292ms

Distal octree created in 1039.2071ms

Table 7-8: Real tibial total knee – unoptimized octree breakdown.

Octant Scale	Filled Octants	Empty Octants
2048		4
1024		24
512		36
256		101
128	1	524
64	133	1,592
32	2,386	4,986
16	13,506	19,327
8	61,237	74,226
4	258,342	281,592
2	1,040,741	1,125,607
1	8,798,586	4,302,502

This led to a total of 18,269,089 octants (leaf + branch), the octree optimisation algorithm was run taking 0.046114 seconds, the updated tree breakdown is shown in Table 7-9.

Table 7-9: Real tibial total knee – optimized octree breakdown.

Octant Scale	Filled Octants	Empty Octants
2048		4
1024		24
512		36
256		101
128	1	524
64	141	1,592
32	2,428	4,986
16	13,605	19,327
8	61,854	74,226
4	261,680	281,592
2	1,054,755	1,125,607
1	4,278,026	4,302,502

The optimised octree has a total of 13,123,441 octants (leaf + branch) and a calculated total filled volume of 6713.892mm^3.

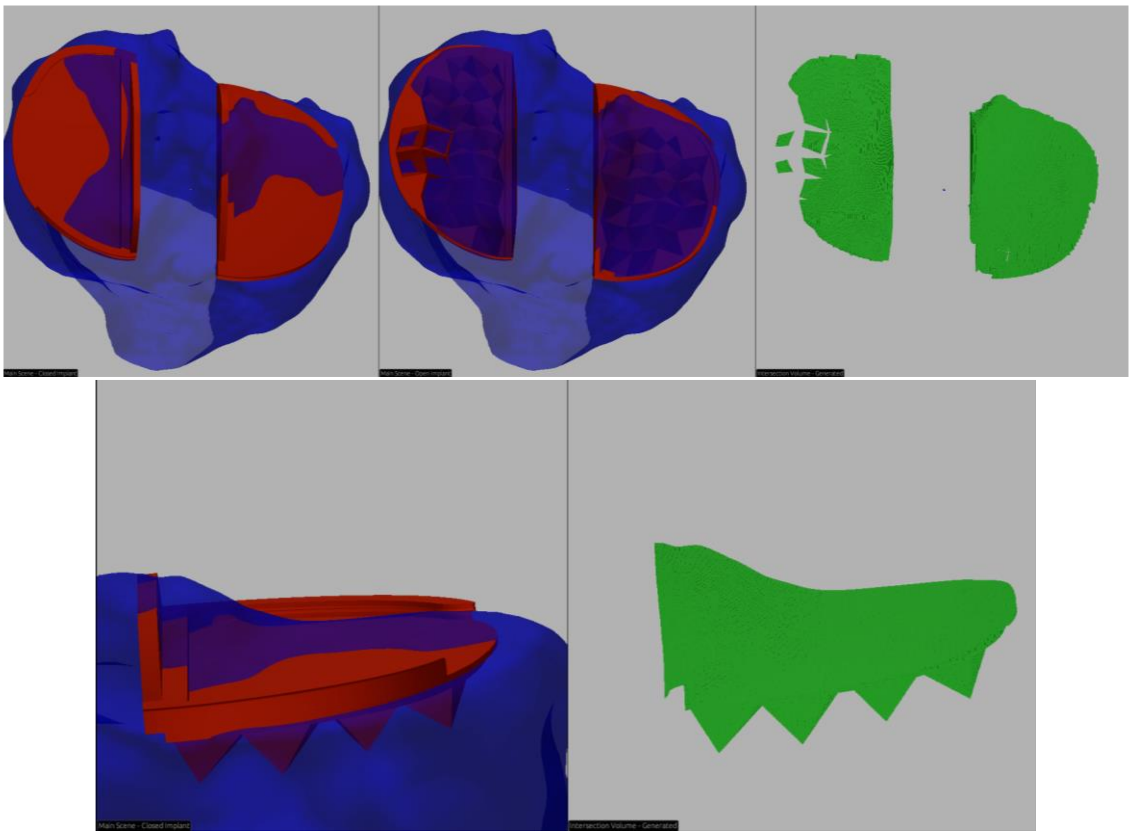


Figure 7-7: Real tibial total knee - octree intersection volume.

Despite implant scale being 10% smaller, there was a 2.1% increase in intersection volume. Likely a result of the more realistic rougher bone surface and different implant positioning.

8 BONE DEFORMATION

Due to the natural deformation and porosity of bone, a bone exactly cut to fit an implant will most likely result in a loose fit [14]. The AIRO-x implant is designed to press fit into the bone, with its grooves 'gripping' the bone [10]. The ideal scenario is a small 'undercut' of the bone, allowing the bone to deform slightly when clipping the implant in, to ensure it fits tightly and has good surface contact. A way for the JRPA to modify the output intersection volume to allow for deformation is required.

8.1.1 Implant Scaling

Implant scaling is not a viable method of accounting for bone deformation, As the implant features will not line up with their correct positions once scaled up. We need a way to enlarge the features but not change their relative positions.

8.1.2 Minkowski Sum

To account for bone deformation in a parametric and deterministic way a mathematical formula called the Minkowski sum was used. The Minkowski sum is the equivalent of running one shape over the surface of another and getting a new shape [40]. Figure 8-1 shows the Minkowski sum computed for a rabbit model and small sphere. The relative position of the rabbit's features remains consistent, but the scale of the features is enlarged by the sphere.

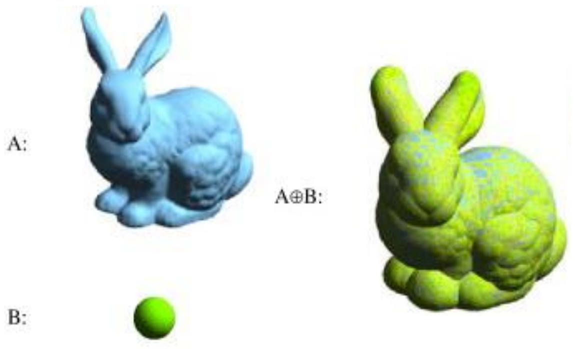


Figure 8-1: Minkowski sum example of a rabbit and sphere [1].

To compute the Minkowski sum of the two polyhedra the 3D mesh-based Minkowski sum software “m+3d” written by Jyh-Ming Lien was used [41]. This software takes two 3D ‘.obj’ files and computes the Minkowski sum of them. The Minkowski sum preformed on the original implant model led to an ‘over extruded’ model. The direction of all the implant mesh face normal had to be flipped. This was done by writing a program to reorder the vertices of each face to change the direction of the triangle normal via the ‘right hand rule’. The Minkowski sum of the implant face and various shapes was preformed to compare results.

8.1.2.1 0.6mm sphere

Figure 8-2 shows the Minkowski sum results of the total femoral implant and a sphere with 0.6mm diameter, compared to the original implant. The total computation time for the sum was 5421ms.

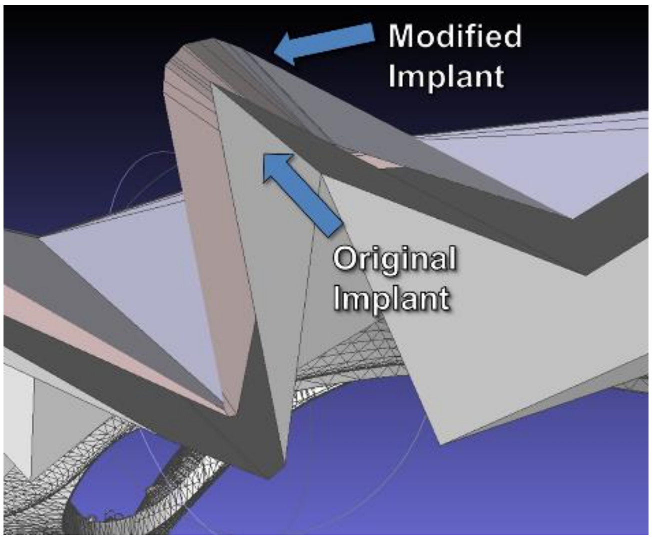


Figure 8-2: Minkowski sum of femoral implant and 0.6mm sphere.

The implant face is now shifted backwards, but its details and geometries are kept consistent. In comparison to if we just reduced the scale of the implant model. The implant model being ‘larger’ means the bone is slightly undercut, hopefully resulting in a tight fit for the actual implant. Figure 8-3 shows the Minkowski sum result in grey with the original implant shown as a wireframe. The relative position of the implant features has not changed but the scale of the features has.

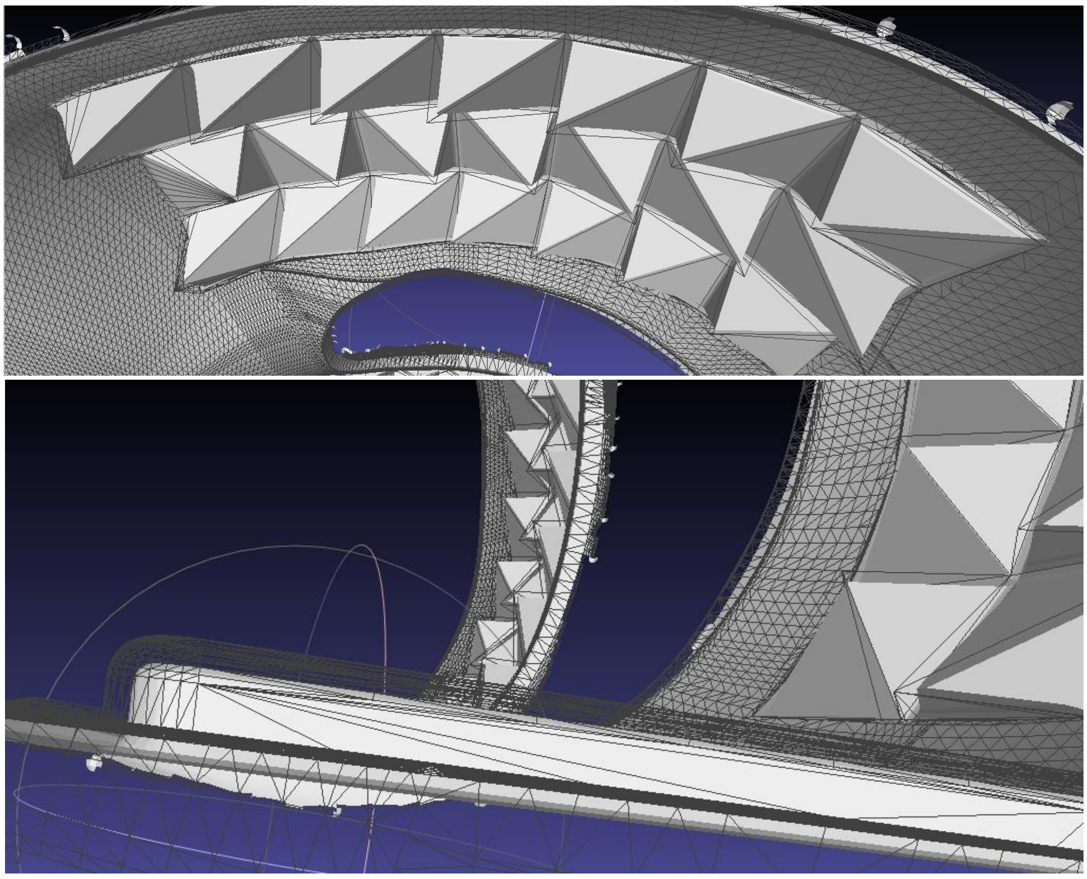


Figure 8-3: Minkowski sum of femoral implant and 0.6mm sphere.

8.1.2.2 0.3mm sphere

With a 0.3mm diameter sphere the effect is similar just reduced. Again, the original femoral implant is shown with a wireframe, and the Minkowski sum result in grey. The total computation time for the sum was 5079ms.

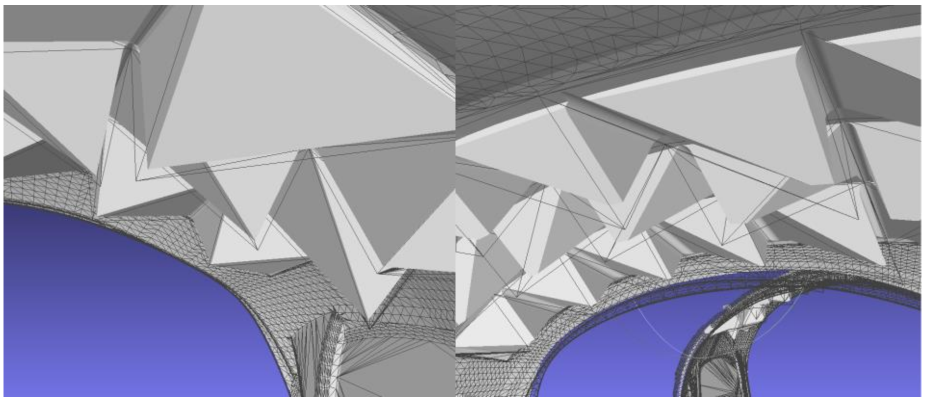


Figure 8-4: Minkowski sum of femoral implant and 0.3mm sphere.

8.1.2.3 0.6mm * 1mm cylinder

To test the Minkowski sum result with more complex geometries a test was done with the femoral implant and a 0.6mm diameter * 1mm height cylinder. The Minkowski sum result in grey shown in Figure 8-5 shows less rounded geometries. The total computation time for the sum was 6102ms.

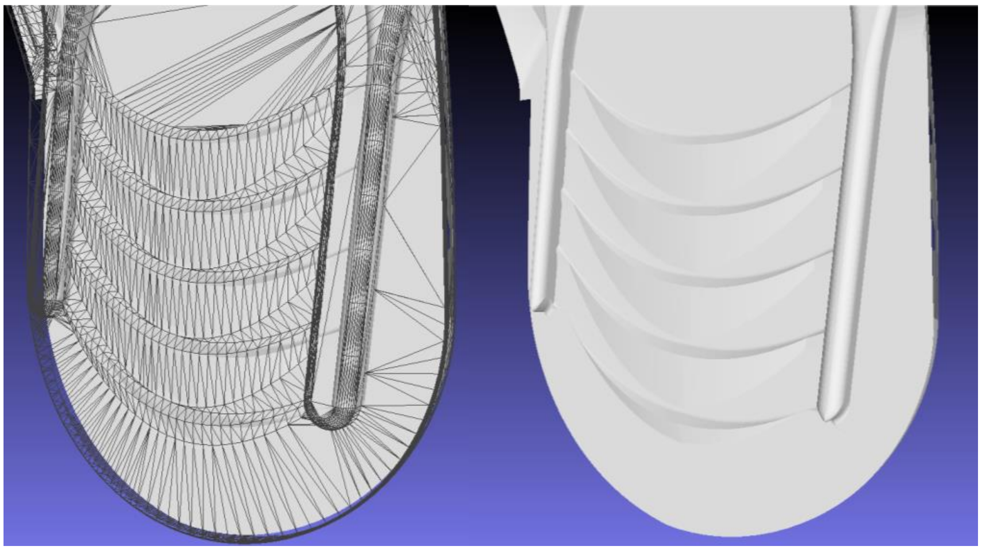


Figure 8-5: Minkowski sum of femoral implant and cylinder.

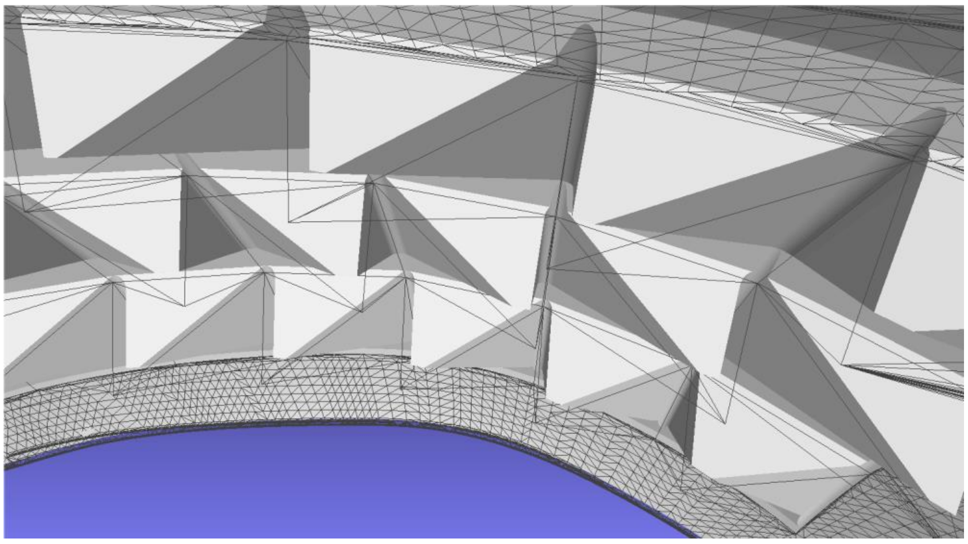


Figure 8-6: Minkowski sum of femoral implant and cylinder.

The Minkowski sum presents a way to parametrically edit the implant mesh fed into the JRPA to account for bone deformation and allow a tight press fit implant. Modifying the implant model instead of the volume intersection code of the JRPA simplifies computation. The Minkowski sum operation could be precomputed for each of the limited implant designs. Using bone-implant friction coefficient results as presented in [42] and [43] mechanical analysis could be done on the individual implant models to determine the desired bone displacement and then the desired ‘B’ shape.

9 ABLATION SIMULATION

The cuboid and octree intersection volume structures were both tested for simulating the process of laser ablating a bone.

9.1 Cuboid Structure

To simulate ablation the intersection model was converted to points and stored in a k-d tree, this led to 8 points per cuboid. The times to store and convert the intersection model points to a k-d tree are outlined in section 6.2.

A number of different methods for transforming the points on the mesh were attempted. For each ablation the k-d tree was queried with a sphere centre and radius, which returned the points within that sphere. Then these mesh points were all transformed according to the equations chosen.

The first thing tried was simply removing the points from the mesh, this did not work as it left holes in the mesh surface. As the mesh is hollow inside (no points) only the surface of the mesh would be ablated.

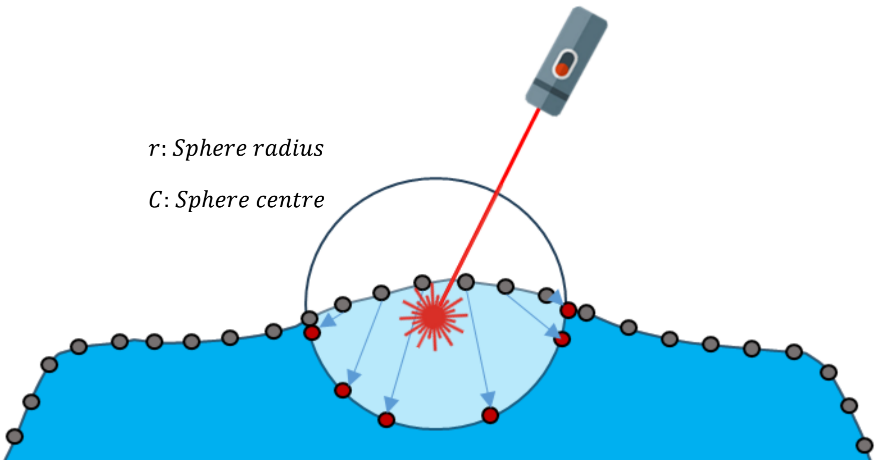


Figure 9-1: Point transformation for ablation simulation: surface projection.

```
**Algorithm:** Project point to nearest surface:
        $Center\ of\ sphere: C \quad Radius\ of\ sphere: r$
        $Point\ inside\ sphere: P$
**Direction from Centre to Point:** $D = \frac{P-C}{|P-C|}$
**Compute Point on Sphere's Surface:** $P' = C + D \times r$
***END ALGORITHM***
```

Algorithm 9-1: Point transformation: sphere surface projection.

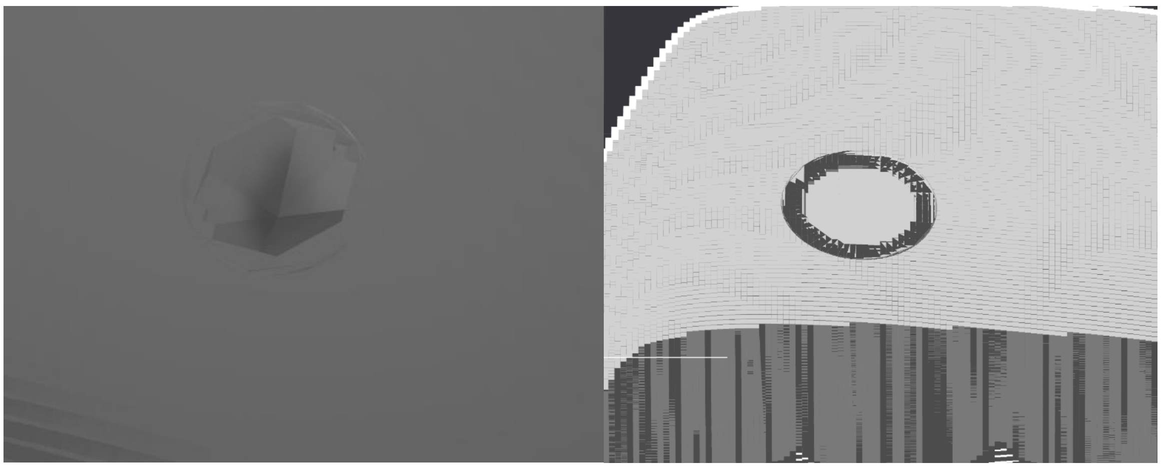


Figure 9-2: Surface projection - simulated ablation craters.

This produced a good result for singular ablation craters, but for multiple craters mesh points begun to overlap causing issues with the mesh faces, as seen in the right of Figure 9-2. This method of point transformation did not consider the direction of the laser pulse.

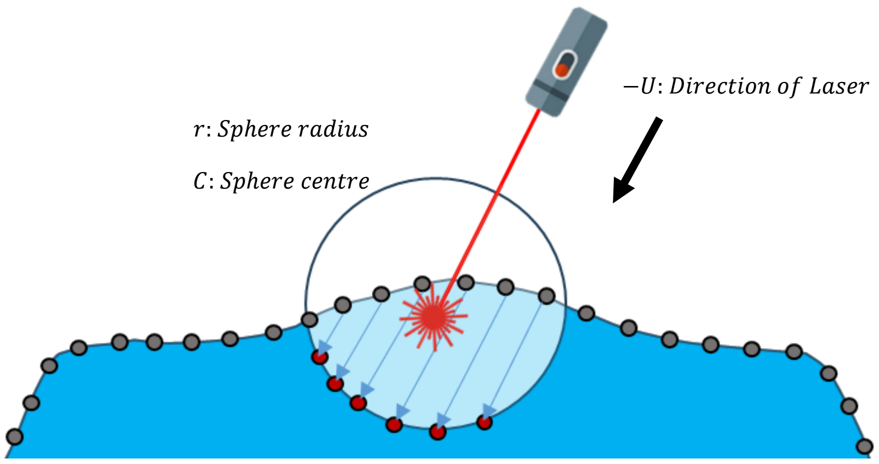


Figure 9-3: Point transformation for ablation simulation: downwards sphere projection.

Algorithm: Project point down to surface:

$$\mathit{Center\ of\ sphere}: C \quad \mathit{Radius\ of\ sphere}: r$$

$$\mathit{Point\ inside\ sphere}: P \quad \mathit{Up\ vector}: U$$

Check if point is in sphere: $|P - C| > r$ otherwise ignore.

Direction of projection: $D = -\frac{U}{|U|}$

Displacement from centre: $\Delta = P - C$

Quadratic Equation Coefficients: Using the equation of a line $P(t) = P + tD$ and the equation of a sphere $(x - Cx)^2 + (y - Cy)^2 + (z - Cz)^2 = r^2$, we can derive a quadratic equation in terms of t as:

$a = 1$ (since the direction vector D is normalized)

$$b = D \cdot \Delta$$

$$c = |\Delta|^2 - r^2$$

Solve for t: Using the quadratic formula, we get two solutions for t. We are interested in the smaller root $t = -b - \sqrt{b^2 - c}$

Compute the Point on the Sphere's Surface: $P' = P + tD$

END ALGORITHM

Algorithm 9-2: Point transformation: downwards sphere projection.

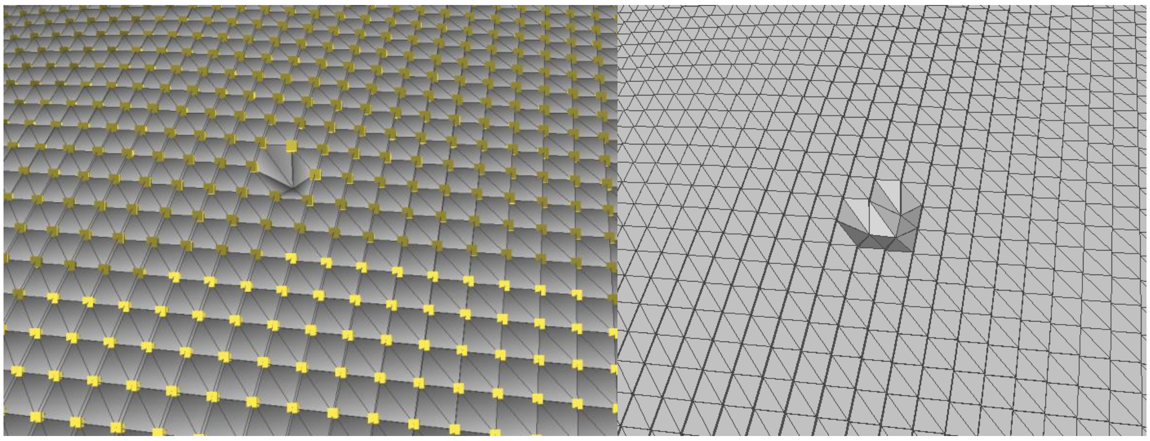


Figure 9-4: Directed surface projection - simulated ablation craters.

Directed surface projection of mesh points inside the ablation sphere produced good looking ablation craters in the cuboid depth intersection model.

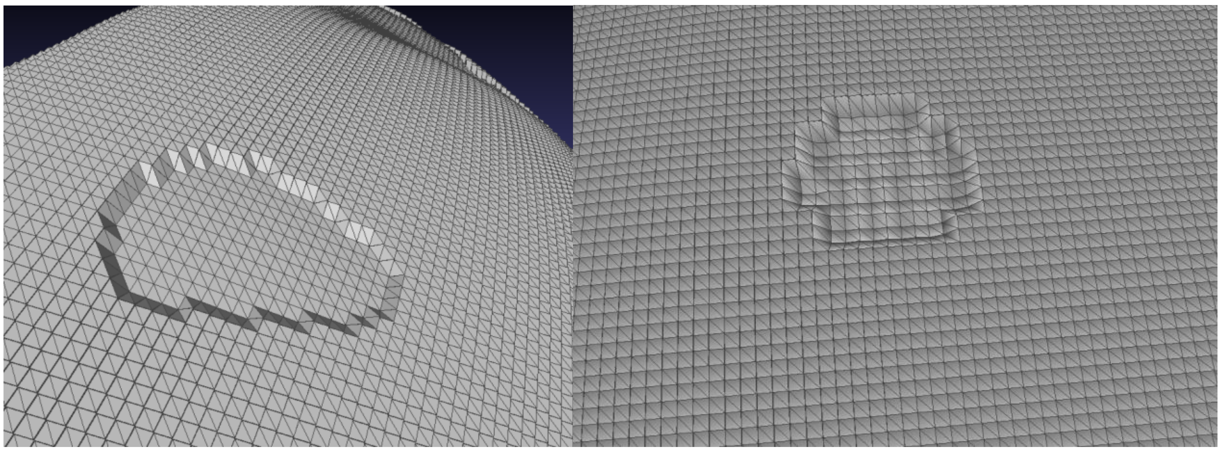


Figure 9-5: Directed surface projection - simulated ablation zones.

Ablating multiple craters in a zone produced nice flat surfaces, increasing the resolution of the cuboid depth intersection volume allowed for surface deviation to appear, see right of Figure 9-5.

9.1.1 Ablation Path Planning

Given the intersection volume a method for selecting and ordering the ablation points is required to ablate the volume. A simple approach is to choose an ablation direction and find the highest point in the intersection volume in that direction. A plane is then created slightly below the point perpendicular to the direction vector. All points above that plane are then ablated, this creates a thin zone of ablation like the green box seen in Figure 9-6.

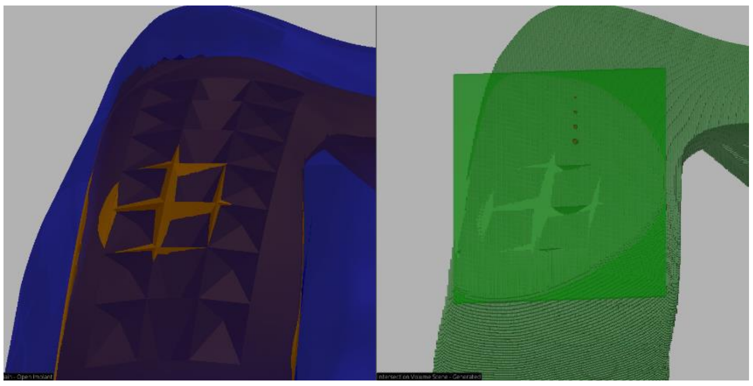


Figure 9-6: Ablation zoning.

9.1.2 Femoral Ablation Simulation

To test the viability of the cuboid intersection volume for simulating ablation, an ideal femur bone and total knee femoral implant were used for an ablation simulation. The intersection volume was generated at 400-pixel height with a resolution of 0.16829754mm.

Approximate time per ablation was 5000µs this included k-d tree lookup, mesh point transformation, and k-d tree editing. A sphere of 0.3mm radius was used as this is the approximate size of the ablation crater caused by a laser pulse during AIRO’s lab testing.

Figure 9-7 shows the ablation simulation in progress, the green intersection volume is queried for the next point to ablate, then both the blue femur and the intersection volume are edited.

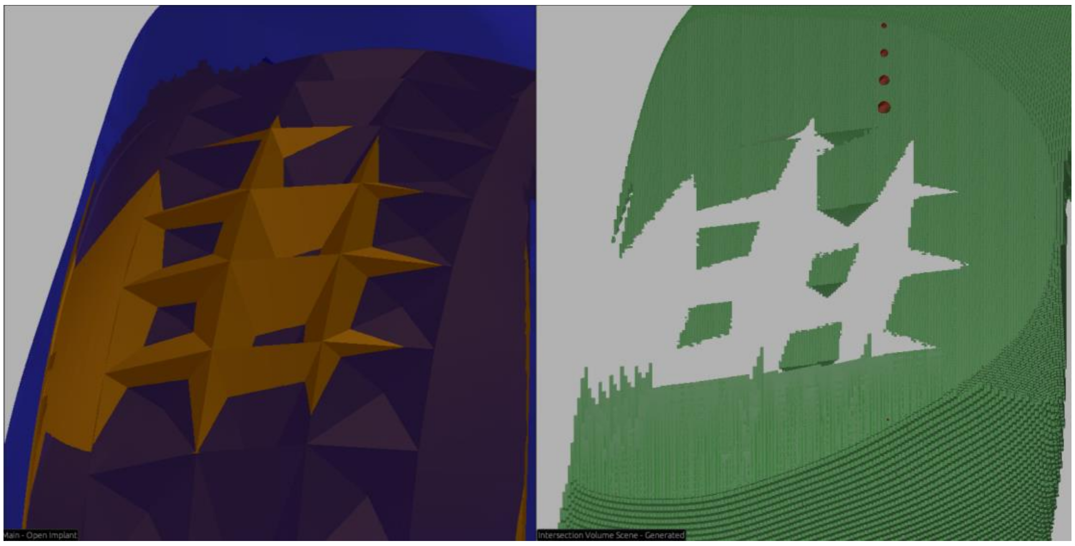


Figure 9-7: Femoral total knee - cuboid ablation simulation.

Highest point lookup for depth model k-d tree was 23.784ms. Trying to simulate a full femur ablation using the cuboid structure was too slow and could not be completed.

9.1.3 Simulating the Bone Ablation

A k-d tree was used to store the mesh vertices of the indexed bone model mesh. When a sphere position was removed from the depth intersection volume a 'within sphere' lookup was done on the bone model's k-d tree. These points were transformed using the same method outlined in

Figure 9-3. Creating the femur k-d tree with 189,064 vertices took 8.6048ms and the tibia k-d tree with 26,344 vertices took 3.9947ms. Figure 9-8 shows the result of simulating many ablations on a femur model.

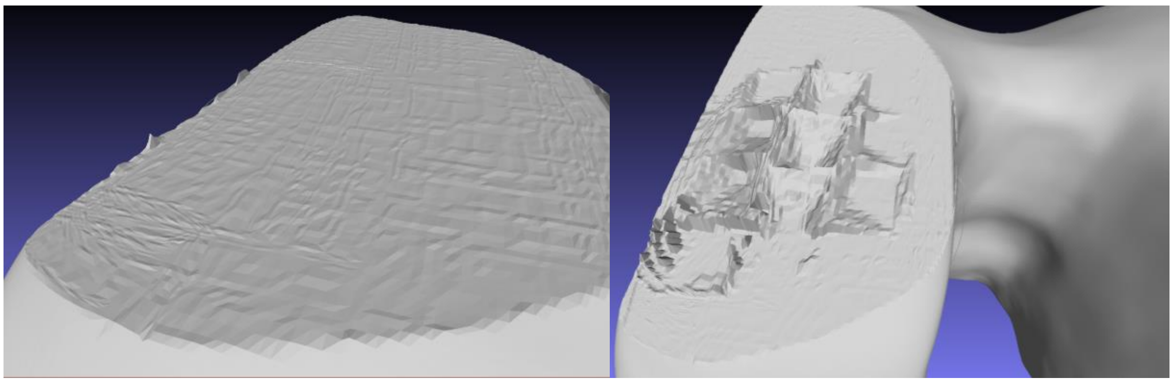


Figure 9-8: Femoral ablation simulation: flat plane (left), complex shape (right).

After each ablation the k-d tree is updated to reflect the transformed mesh points and check is preformed to remove any now duplicate vertices and remove any triangles with a zero area. Figure 9-9 is a flow diagram of the process.

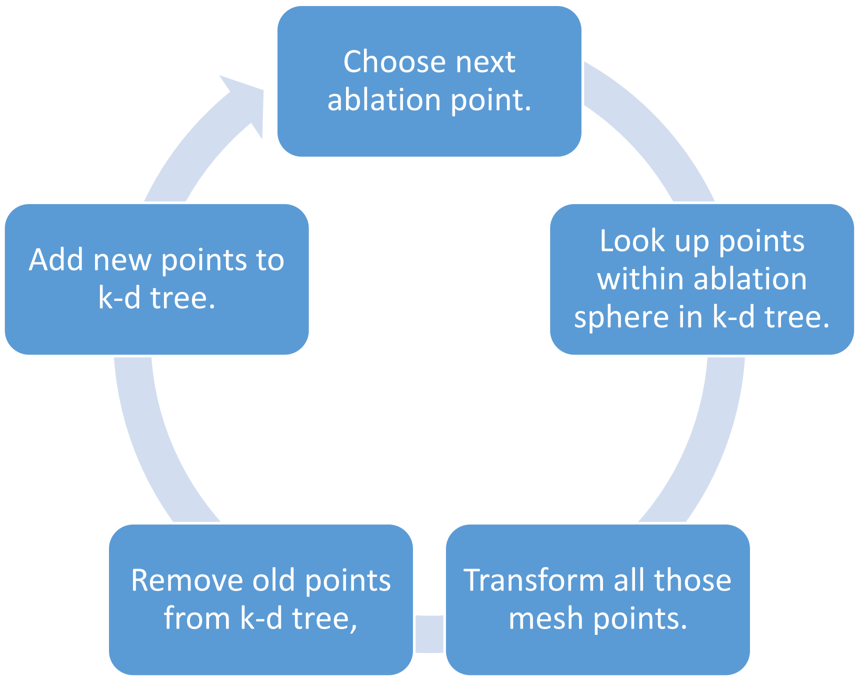

Choose next ablation point.
Look up points within ablation sphere in k-d tree.
Transform all those mesh points.
Remove old points from k-d tree,
Add new points to k-d tree.

Figure 9-9: Flow diagram of bone ablation simulation.

9.1.4 Directionality Issue

After an extended period of simulation with the cuboid structure an issue with the cuboid intersection volume becomes apparent. From certain directions the vertices of each cuboid are too far apart for the ablation simulation method to work.

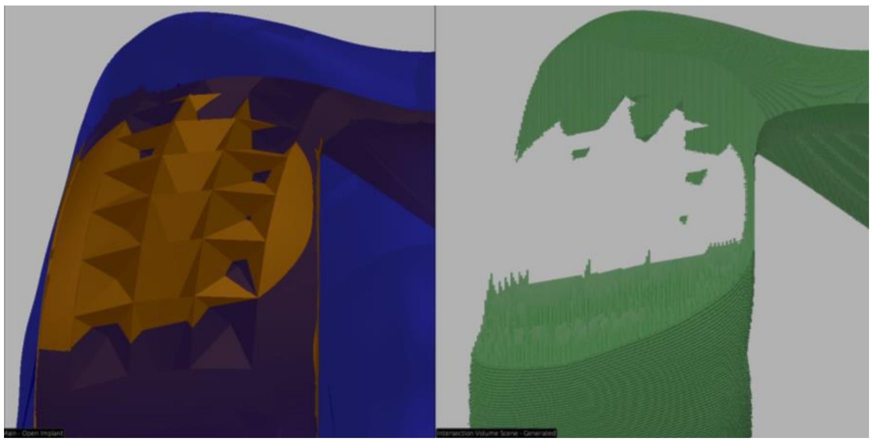


Figure 9-10: Cuboid ablation simulation gets stuck at this point.

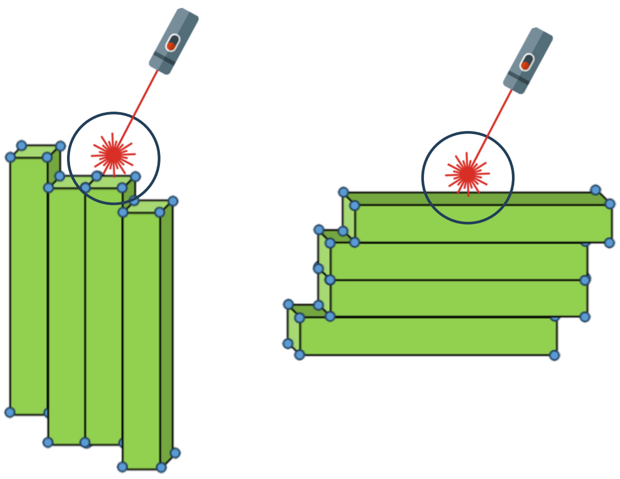


Figure 9-11: Directionality issue with ablation simulation of cuboids.

9.1.5 Subdividing Cuboids

To solve the issue with ablation simulation using the cuboid structure, subdividing the cuboid was tried. To attempt to get more points in and remove large gaps in points along the length of the cuboid.

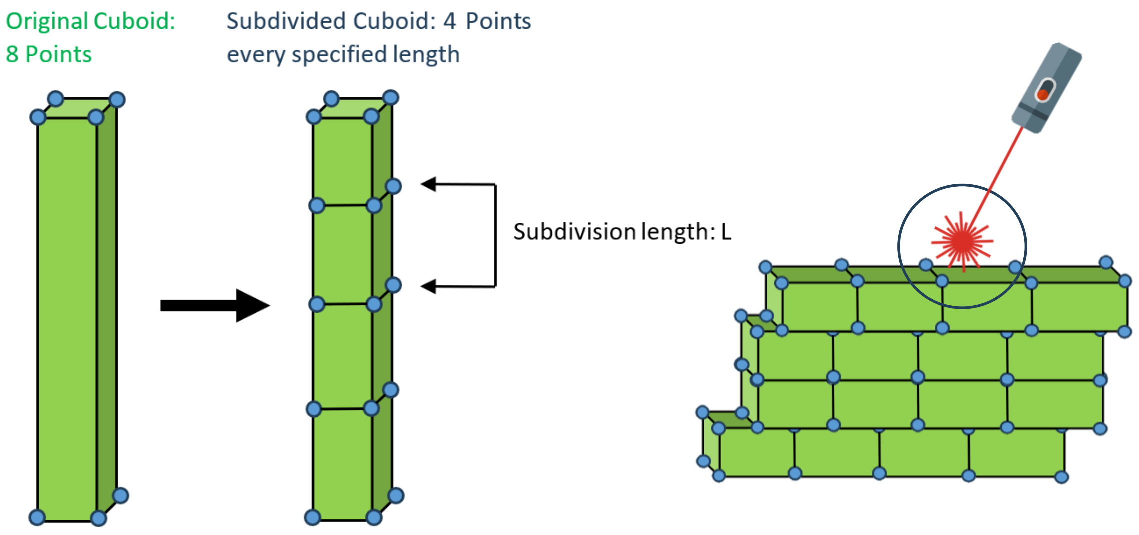
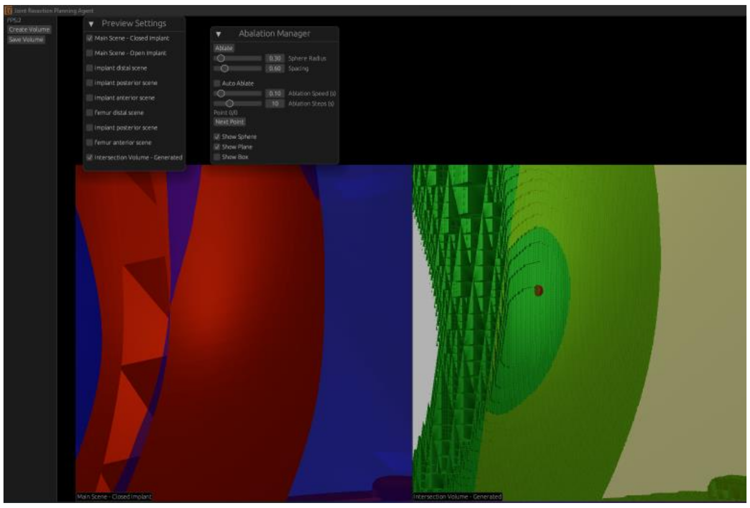


Figure 9-12: Cuboid subdivision.

This solved the directionality issue, however, was far too slow and memory inefficient. Leading to too many points for a high-resolution depth intersection volume. Used too much memory, really slow to generate and k-d tree point lookups became too slow. There is a lot of unnecessary information stored in the centre of the volume. Storing many points in the centre of the volume is unnecessary as we only ever need to look up the points on the surface. This is where the octree representation comes in.

9.2 Octree Structure

9.2.1 Simulating Ablation


Joint Resection Planning Agent
FPS:2
Create Volume
Save Volume
Preview Settings
Abalation Manager
Ablate
Next Point
Main Scene - Closed Implant
Intersection Volume - Generated

[✓] Main Scene - Closed Implant
[ ] Main Scene - Open Implant
[ ] Implant distal scene
[ ] Implant posterior scene
[ ] Implant anterior scene
[ ] Femur distal scene
[ ] Implant posterior scene
[ ] Femur anterior scene
[✓] Intersection Volume - Generated

0.30 Sphere Radius
0.60 Spacing
[ ] Auto Ablate
0.10 Ablation Speed (s)
10 Ablation Steps (s)
Point 0/0
[✓] Show Sphere
[✓] Show Plane
[ ] Show Box

Figure 9-13: Ablation of octree volume in GUI.

To simulate ablation with the octree data structure a recursive look up is used. First the root octant is added then the child octants that are intersecting the sphere are checked. Then each child octant is checked. If an octant is empty, it is ignored. If an octant is filled, it is subdivided until the minimum octant size is reached. In which case it is set to empty if still touching the sphere.

```
Algorithm: SimulateAblation
Input: rootOctant, sphereCenter, sphereRadius, minOctantSize

1. Initialize a stack and push the rootOctant onto it.
2. WHILE the stack is not empty DO
    2.1. Pop an octant from the stack to become the currentOctant.

    2.2. IF currentOctant is Empty THEN
        2.2.1. CONTINUE to the next iteration.
    END IF

    2.3. Compute the intersection between currentOctant and the sphere.

    2.4. IF currentOctant does not intersect with the sphere THEN
        2.4.1. CONTINUE to the next iteration.
    END IF

    2.5. IF currentOctant is Filled THEN
        2.5.1. IF size of currentOctant <= minOctantSize THEN
            2.5.1.1. Set currentOctant to Empty.
            2.5.1.2. CONTINUE to the next iteration.
        END IF
        2.5.2. Subdivide currentOctant into child octants.
        2.5.3. Push each child octant onto the stack.
    ELSE IF currentOctant is a Branch THEN
        2.5.4. Push each child octant onto the stack.
    END IF
END WHILE
```

Algorithm 9-3: Octree ablation.

Figure 9-15 shows a 2D representation of the octree ablation method. All the octants touching the sphere are subdivided until they do not touch the sphere or are at the minimum size and cannot be split further. Any octants left touching the sphere are emptied.

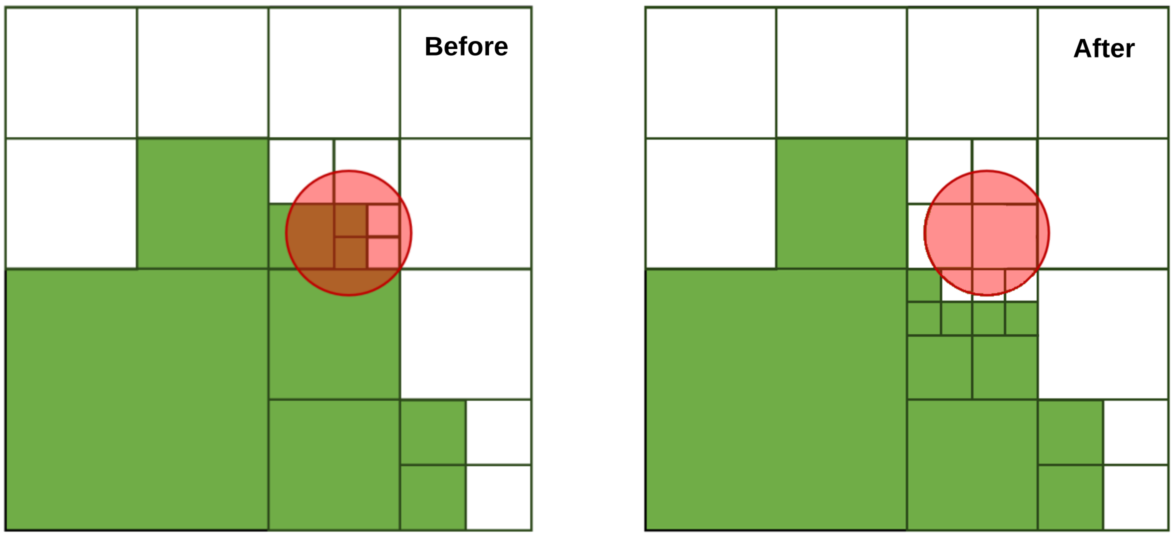


Figure 9-14: 2D diagram of octree ablation.

Higher resolution octrees get more accurate ablation results as shown by Figure 9-17. The smaller cube size allows the octree to match the spherical shape more closely, compared to Figure 9-15.

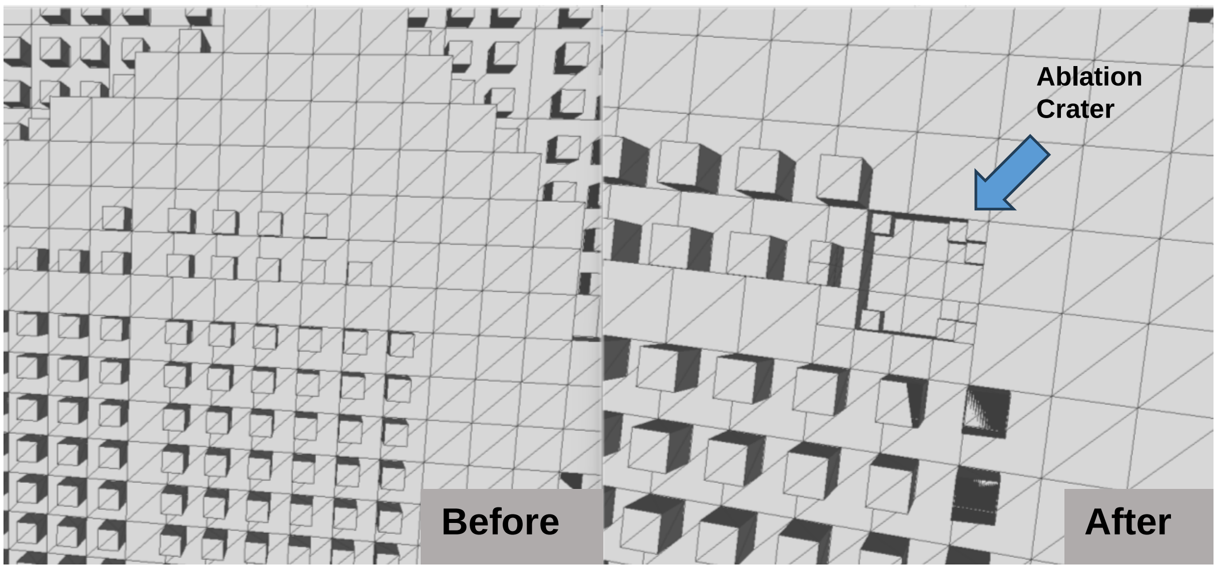


Figure 9-15: Octree ablation on generated low-res octree.

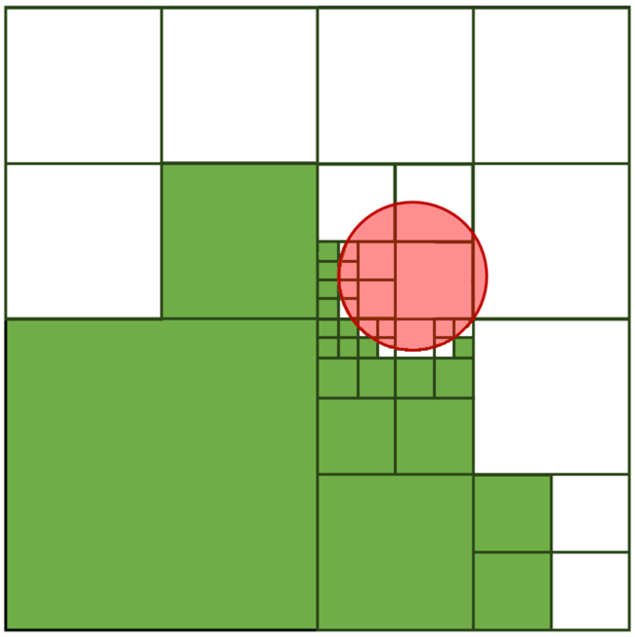


Figure 9-16: 2D diagram of high-resolution octree ablation.

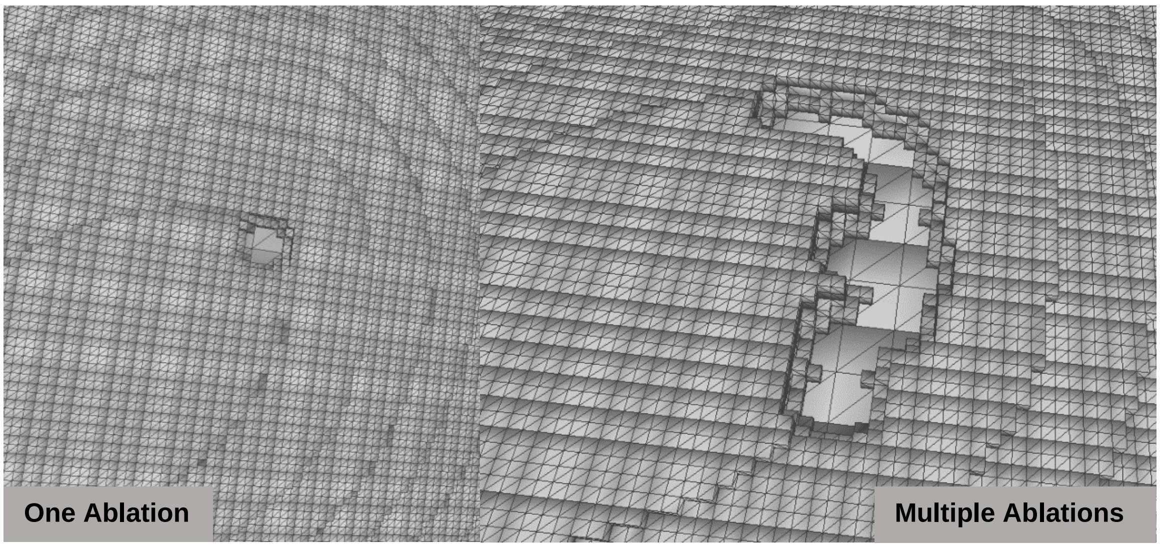


Figure 9-17: Octree ablation on high res generated octree.

Originally the octree ablation time was on average approximately 50ms per ablation. 20 ablations per second. With incremental code and efficiency improvements, primarily octree optimisation and removing excess debugging code ablation times were improved to approximately 400us per ablation. Or 2500 ablations per second.

9.2.1.1 Ablation time creep

This method leads to unoptimized octants, as octants are split then ablated a split octant could potentially contain all filled or all empty children, meaning the parent octant can just be collapsed down to an empty or filled leaf rather than a branch. The ablation times increase as simulation continues, due to the octree getting more complex as octants split. This time increase can be seen in the simulation data shown in Figure 9-18. An example of ablation lookup times is shown in Table 9-1.

Table 9-1: Ablation process timing breakdown.

Process	10% into ablation simulation	70% into ablation simulation
Filled octants	1,457,165	2,585,604
Highest octant lookup	147.4 µs	2213 µs
Octree ablation	141.9 µs	153.7 µs
Bone ablation	92.4 µs	398.4 µs

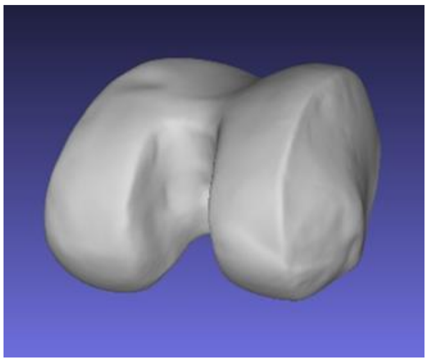
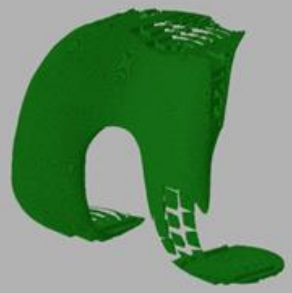
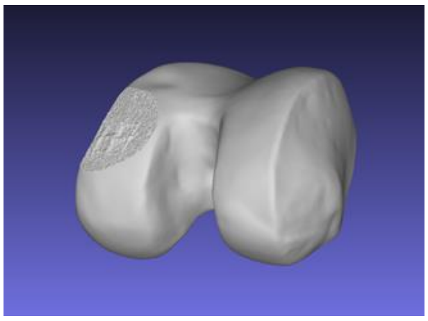
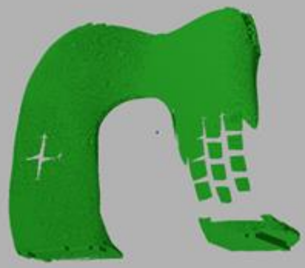
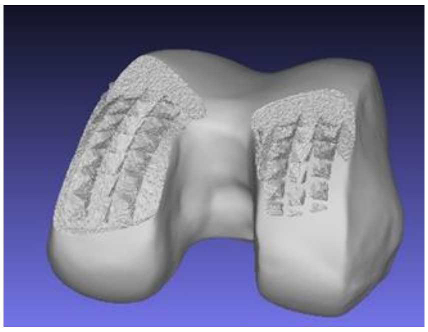
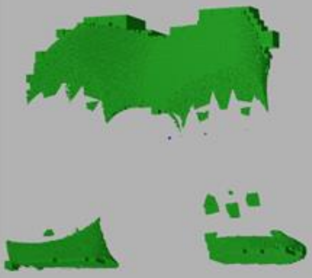
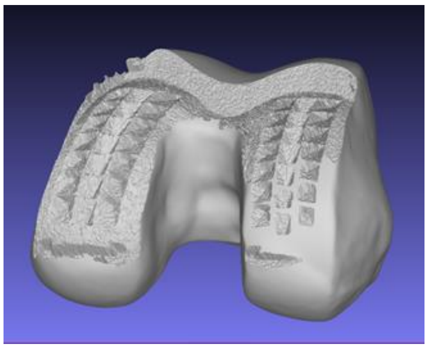
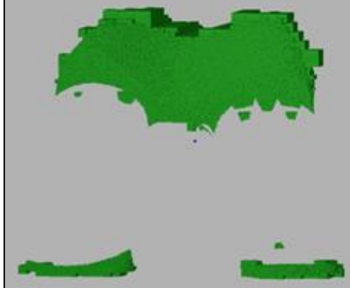
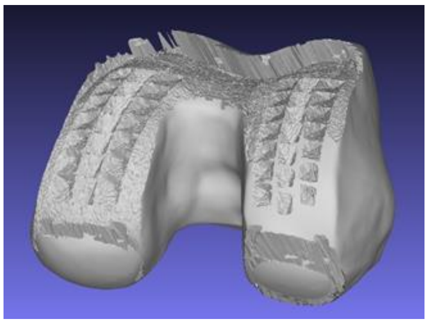
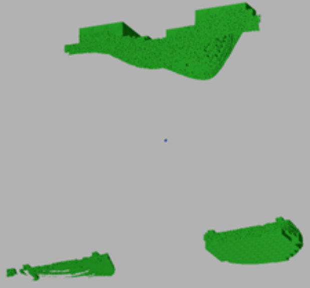
The octree optimisation algorithm from section 7.2.2 was run periodically as the simulation was running to avoid ablation time creep. With optimisation the ablation lookup times stay consistent and even decrease as the intersection volume decreases as bone is removed. This greatly improved simulation performance keeping ablation times much lower throughout the simulation.

9.2.2 Femoral Ablation Simulation

9.2.2.1 Ideal Femur

Femur Ideal: faces: 378,124

Table 9-2: Femoral ablation simulation progress.

Simulation Progress	Femur Output	Intersection Volume
0%		
15%		
50%		
70%		
85%		

Simulation femur output shows great results, towards the end of ablation the poor quality of the single view intersection volume becomes apparent. Outlining the need for multiple view octree creation.

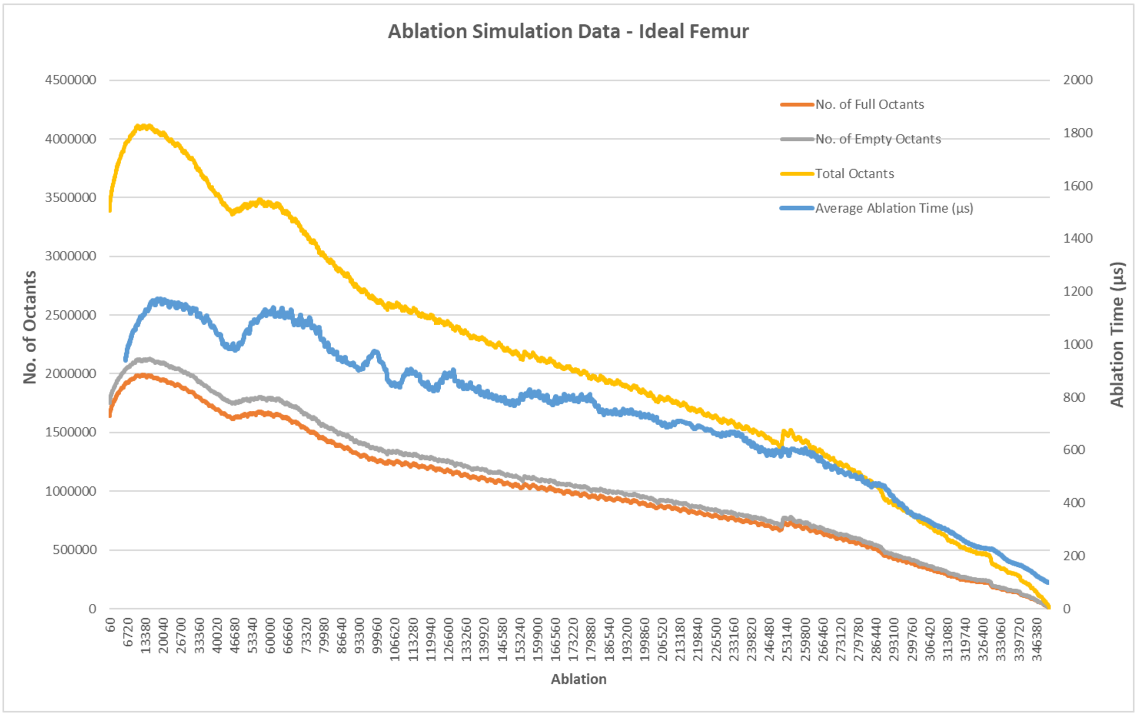


Figure 9-18: Ablation simulation data – ideal total femur.

Total time: 254.852968 seconds

Ablation Sphere Radius: 0.3mm

Total ablations: 351,420

Ablation maxed out at an average of 1200us per ablation or 833 ablations per second.

The ablation time tended to increase throughout the simulation as the octree grew more complex and lookup times got longer. Then towards the end the lookup times decreased as the octree grew smaller and octants were simplified to their larger counterparts.

9.2.3 Tibial Ablation Simulation

9.2.3.1 Ideal Tibia

Using a model of an ideal tibia shown in Figure 9-19 with 52,684 faces a total knee ablation simulation was run.

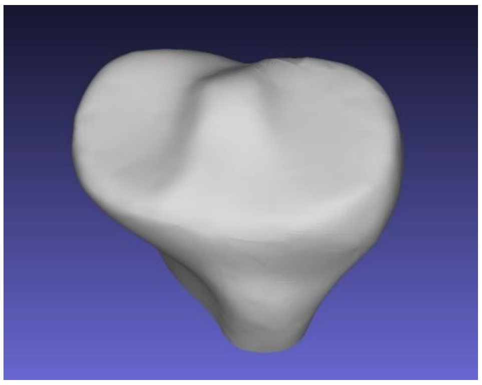


Figure 9-19: Ideal tibia model.

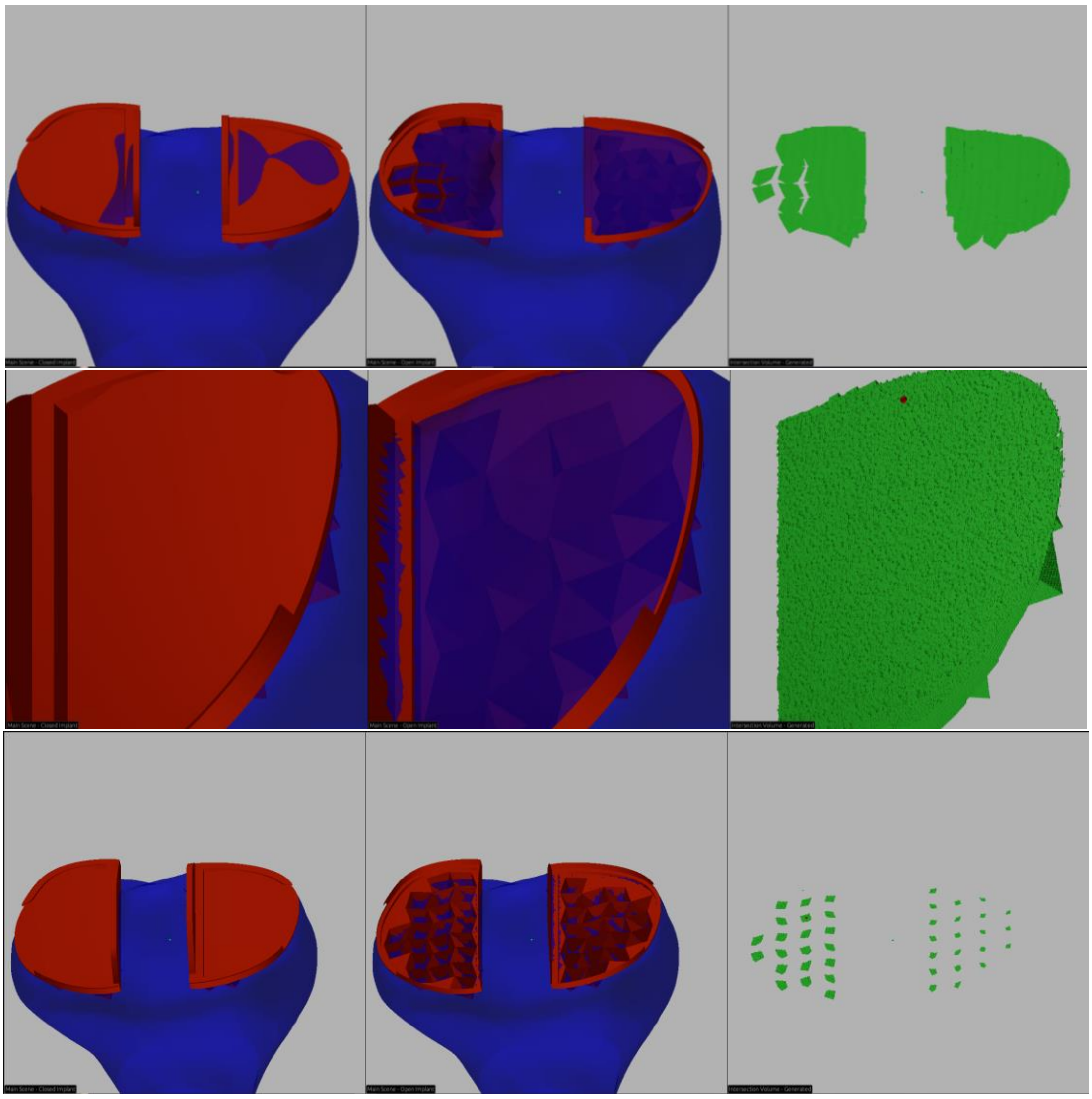


Figure 9-20: Tibial ablation simulation.

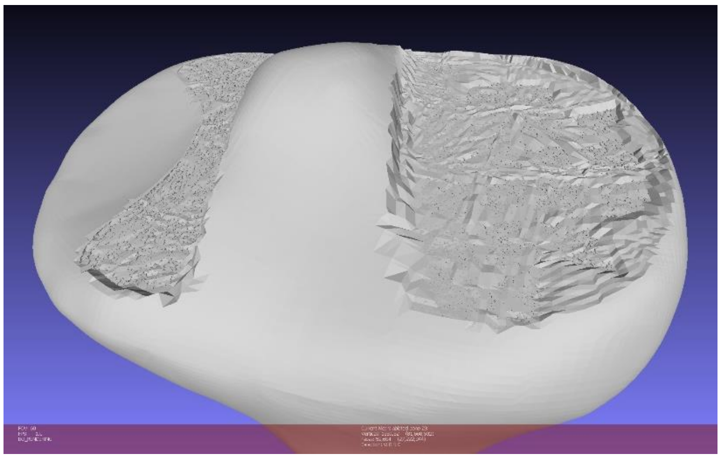


Figure 9-21: 50% ablated tibia model.

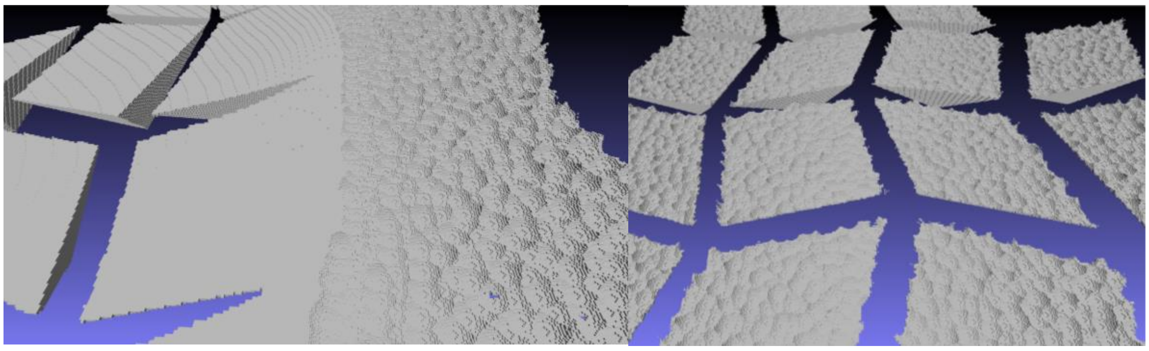


Figure 9-22: Close up of ablation craters in octree intersection volume.

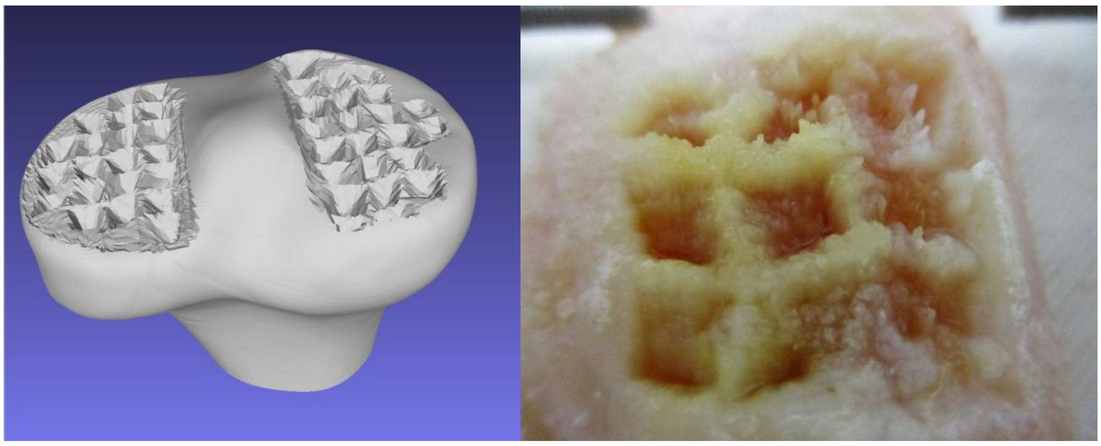


Figure 9-23: Simulated ablated tibia (left), real ablated bone (right).

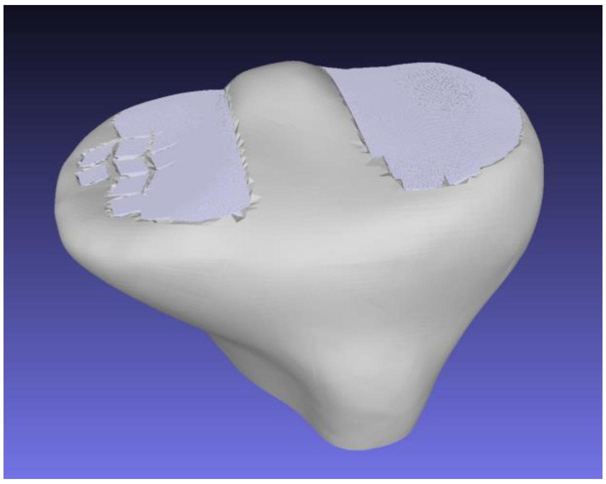


Figure 9-24: Check of intersection volume (light blue) in ablated tibia (grey).

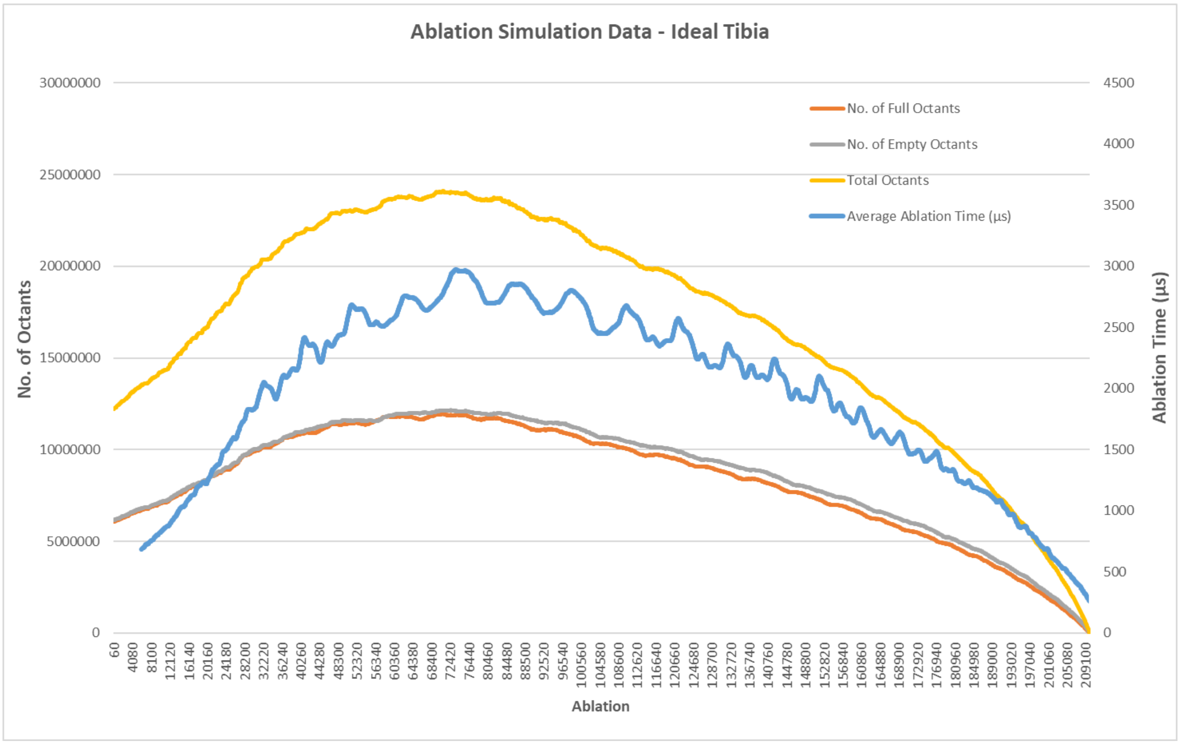


Figure 9-25: Ablation simulation data – ideal total tibia.

Average ablation time is roughly proportional to the number of octants in the octree. Even with octree optimisation the ablation time peaks due to the increased surface information during simulation.

Total time: 403.44197 seconds

Ablation Sphere Radius: 0.3mm

Total ablations: 210,000

These times would be improved with removing the graphical rendering steps + parallel computing.

9.2.3.2 Real Tibia

Using a CT-scan of a real Tibia - SKG2325343. Smoothed with a median filter.

Faces: 124,164

Severe osteoarthritis, lots of Tibial spiking and osteophytes.

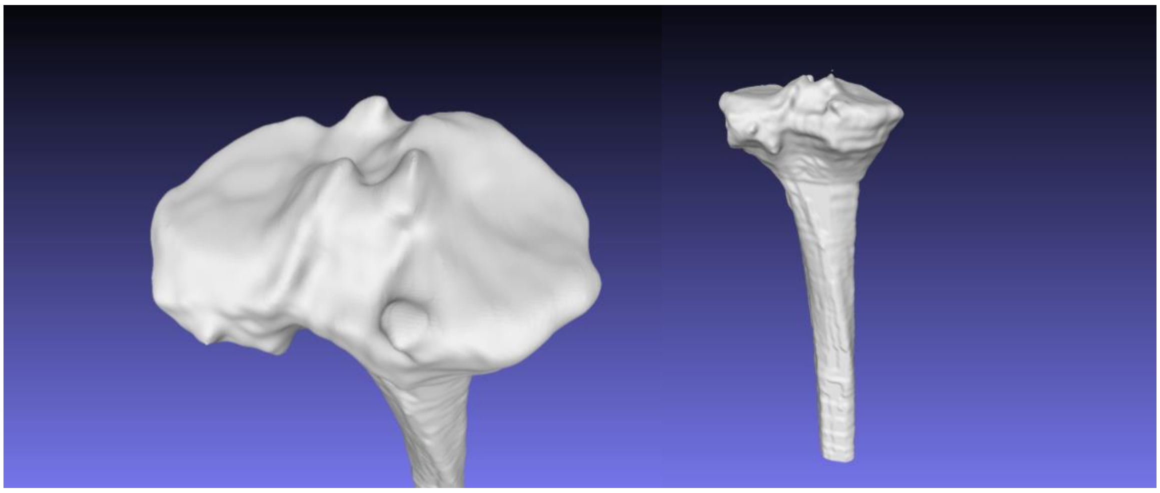
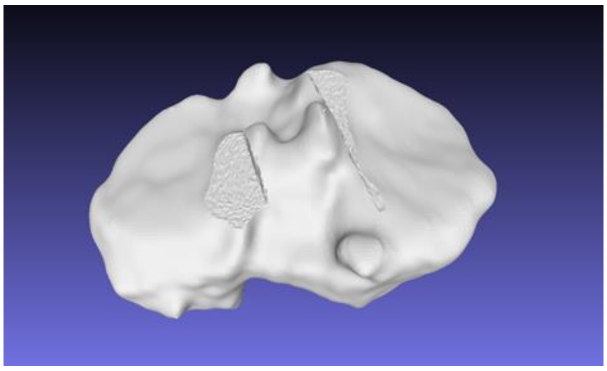
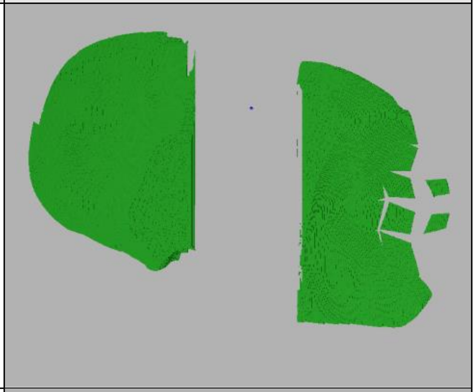
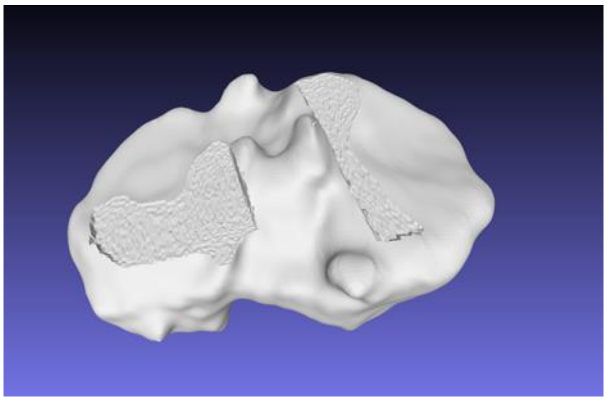
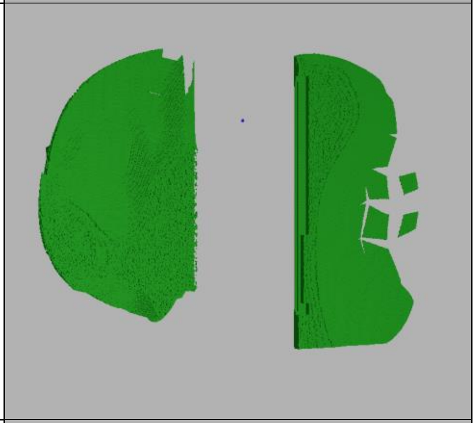
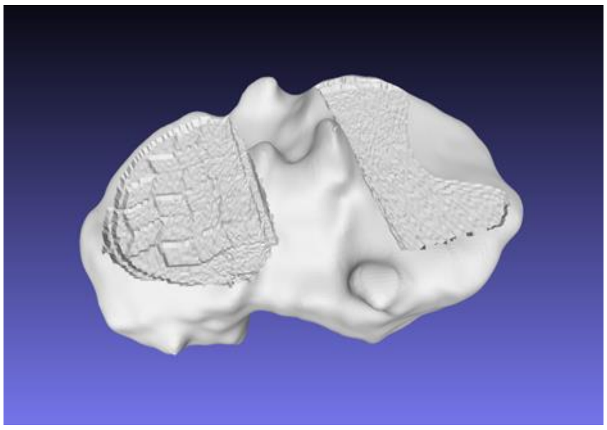
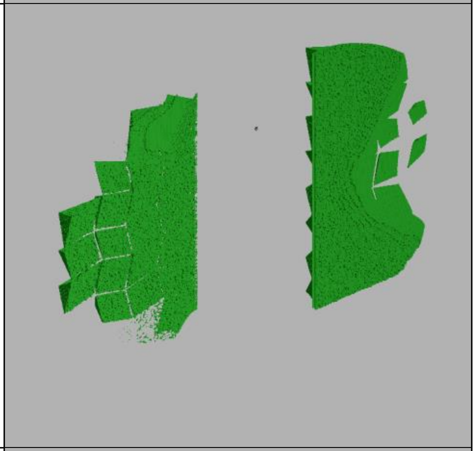
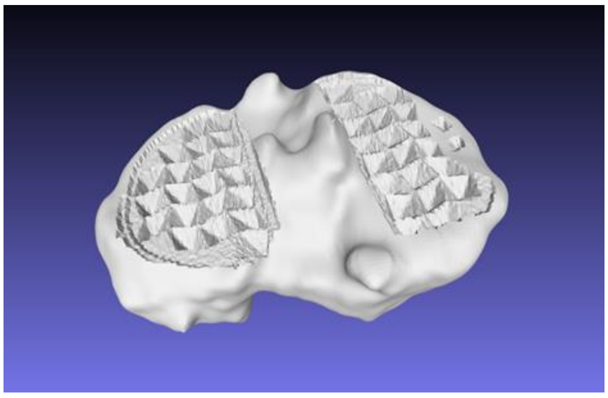


Figure 9-26 : CT scan of tibia with aggressive osteoarthritis.

Table 9-3: Tibial ablation simulation progress.

Simulation Progress	Femur Output	Intersection Volume
0%		
15%		
50%		
100%		

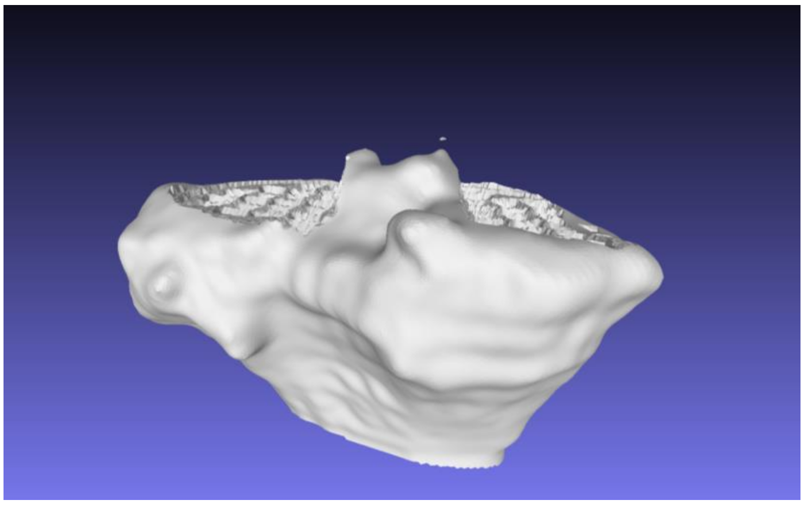


Figure 9-27: Final ablated tibia.

The final ablated tibia model still while able to fit the implant successfully still has many of the tibial spikes and osteophytes preventing free knee joint movement. This outlines a short coming of the method for generating the intersection volume, as only the implant placement is considered and not the knee joint as whole. Allowing free rotation of the knee is the primary goal of knee joint arthroplasty.

The current operation pipeline could be easily adjusted to solve this problem. A two-pass approach firstly ablating the bone to match the desired femur shape more closely, then a second pass ablating in the implant shape. Alternatively modifying the implant mesh to span the medial and lateral sides of the tibia and wrap around the outsides so that osteophytes are seen as ‘’intersection’’ by the algorithm.

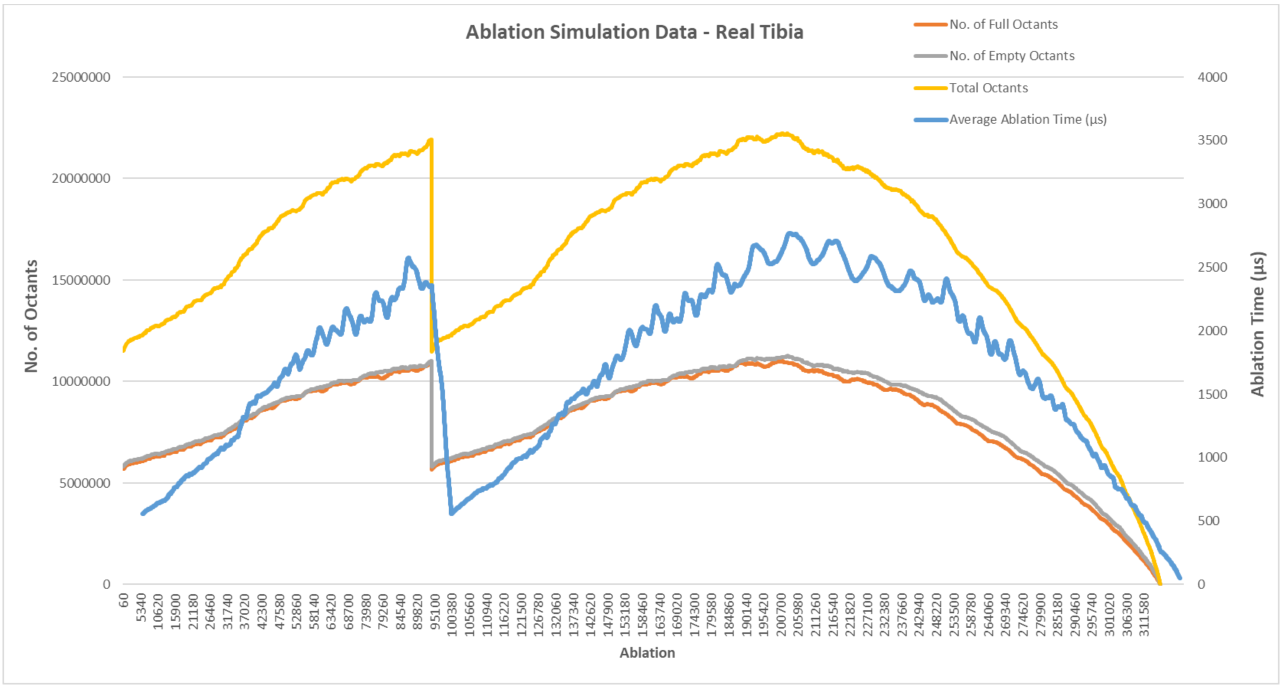


Figure 9-28: Ablation simulation data – real total tibia.

Total time: 536.88957 seconds

Ablation Sphere Radius: 0.3mm

Total ablations: 316,680

Ablation time is again proportional to the number of octants in the octree. In this case we see a sharp drop in octant numbers around the midway mark through the ablation simulation. This was because one of the large-scale octants was able to be collapsed as all children it contained became empty. Massively reducing octant numbers. This points towards another possible optimisation method of positioning the centre of the root octant of the octree in such a way that a minimal number of its children contain filled octants. Allowing for less octants overall but containing the same information.

The ‘spikey’ nature of the average ablation time is likely due to higher scale octants increasing in complexity as ablations are done within them, then being simplified as their contents are fully removed.

10 CONCLUSIONS

The JRPA is a promising new tool for knee replacement surgery for use with joint resection planning for the HAiLO robot. It can handle both unilateral and total knee implants, and the volume analysis works well, producing an accurate intersection volume. Graphics depth buffers are computed quickly at high resolution, and multiple camera views can be used to increase the accuracy of the intersection volume. Using the octree data structure for the intersection volume, a resolution within the target accuracy of 20-40 microns was achieved. The resulting volume correctness was confirmed visually and quantitatively in CAD. However, the octree multi-camera views are not yet working, and the time for volume generation, while within 0.5 to 2 seconds, could be further improved with more powerful dedicated computer hardware and parallel computing.

Another promising feature of the JRPA is the ablation simulation. Ablation is simulated using a top-down sectioned approach from a fixed robot position. Ablation simulation speed is significantly faster than real laser ablation, but there is still room for computational efficiency improvements. Depth intersection model updates are used to show bone that still needs to be removed, and the volume of bone to be removed can be easily calculated, enabling an estimation of surgery time. The JRPA also allows the state of the bone to be simulated during surgery. Simulated ablated bone is accurate to real lab tests, reducing the need for lab testing.

10.1 Future Work

There are several areas where the JRPA can be improved. One is to store bone as an octree to improve ablation simulation. Another is to implement parallel octree lookups and ablation simulation. The intersection volume algorithm does not currently account for osteophytes and bone growths, so it would be good to add that capability.

Multi-view octrees would also be an improvement. The current top-down ablation approach does not consider surgical access to the patient, so better ablation path planning is needed. This should consider macro/micro robot movements and the accuracy or focus point of the laser.

Computational efficiency improvements are also needed, as the current intersection volume generation is too slow. It would be good to take out the rendering steps for the volume, as the final agent would not need to render it.

Finally characterising bone surface, such as roughness, flatness, gradient, peaks/troughs. Would allow for better implant fit analysis, looking at bone surface and friction characteristics described in these papers: [13, 23, 31].

11 DEFINITIONS

Table 11-1: Acronym and jargon definitions.

Term	Definition
AIRO	Australian Institute of Robotic Orthopaedics
UWA	University of Western Australia
HAiLO™	High-speed Intelligent Laser Osteotomy System
AIRO-x	AIRO's custom knee joint implant
Osteotomy	Surgical cutting of bone
Knee Arthroplasty	Surgical procedure to resurface a damaged knee joint
TKA	Total Knee Arthroplasty
Laser Ablation	Removal of bone tissue through vaporisation and ejection via Laser treatment
CT-scan	A CT (computed tomography) scan is a type of X-ray that creates 3-dimensional images of your body
JRPA	Joint Resection Planning Agent
CAM	Computer-Aided Manufacturing
Mesh	A 3D surface or object made from a finite set of triangle faces
STL	Stereolithography file format that stores 3D object information as vertices and triangles.
Float or Floating-Point	The storage of decimal numbers on a computer, cannot exactly represent every number, so is an approximation.
GUI	Graphical User Interface

12 SOFTWARE PROGRAMS USED

Table 12-1: Web links to software programs and libraries used in project.

Software Program	Link
Autodesk Meshmixer	https://meshmixer.com/
m+3d.exe	http://masc.cs.gmu.edu/wiki/SimpleMsum
Autodesk Inventor	https://www.autodesk.com.au/products/inventor/
MeshLab	https://www.meshlab.net/
Visual Studio Code	https://code.visualstudio.com/
Rust Compiler	https://www.rust-lang.org/
v-hacd	https://github.com/kmammou/v-hacd
libigl	https://libigl.github.io/
Blender	https://www.blender.org/

13 RUST CRATES USED

Table 13-1: Rust crates (libraries) used in project.

Crate	Version
three-d	0.16.0
Log	0.4.20
Ncollide3D	0.33.0
Rayon	1.7.0
Kiddo	2.1.1
Dashmap	5.5.3
Tokio	1.32.0
Image	0.24.7
Egui	0.23.0

14 BIBLIOGRAPHY

[1] W. Li and S. McMains, "Voxelized Minkowski sum computation on the GPU with robust culling," *Computer-Aided Design,* vol. 43, no. 10, pp. 1270-1283, 2011/10/01/ 2011, doi: https://doi.org/10.1016/j.cad.2011.06.022.

[2] R. J. K. F. Khan, Daniel Paul ; Robertson, William Brett ; Sheh, Raymond Ka-Man ; Ironside, Charles ; Chipper, Richard, "Robot-assisted laser surgical system," Australia, 2017-08-09,

[3] J. H. Beaty, F. M. Azar, and S. T. Canale, *Campbell's Operative Orthopaedics*. Elsevier, 2020.

[4] O. A. Barrera, H. Haider, and K. L. Garvin, "Towards a standard in assessment of bone cutting for total knee replacement," *Proceedings of the Institution of Mechanical Engineers. Part H, Journal of engineering in medicine,* vol. 222, no. 1, pp. 63-74, 2008, doi: 10.1243/09544119JEIM286.

[5] S. Toksvig-Larsen and L. Ryd, "Surface flatness after bone cutting: A cadaver study of tibial condyles," *Acta orthopaedica,* vol. 62, no. 1, pp. 15-18, 1991, doi: 10.3109/17453679108993084.

[6] R. Vaishya, M. Chauhan, and A. Vaish, "Bone cement," *Journal of clinical orthopaedics and trauma,* vol. 4, no. 4, pp. 157-163, 2013, doi: 10.1016/j.jcot.2013.11.005.

[7] R. Breuer, "Market Requirements," Australian Institute of Robotic Orthopaedics, 2022, issue 2.

[8] K. Denis *et al.*, "Influence of bone milling parameters on the temperature rise, milling forces and surface flatness in view of robot-assisted total knee arthroplasty," *International Congress series,* vol. 1230, pp. 300-306, 2001, doi: 10.1016/S0531-5131(01)00067-X.

[9] D. G. Panduric, I. B. Juric, S. Music, K. Molčanov, M. Sušic, and I. Anic, "Morphological and Ultrastructural Comparative Analysis of Bone Tissue After Er:YAG Laser and Surgical Drill Osteotomy," *Photomedicine and laser surgery,* vol. 32, no. 7, pp. 41-408, 2014, doi: 10.1089/pho.2014.3711.

[10] R. J. K. F. Khan, Daniel Paul ; Robertson, William Brett ; Sheh, Raymond Ka-Man ; Ironside, Charles ; Chipper, Richard, "An Orthopaedic Implant and a Surgical Orthopaedic System Incorporating Same," Patent Appl. WO 2021/217207 A1, 2021.

[11] P. Honigmann *et al.*, "Cold ablation robot-guided laser osteotomy in hand, wrist and forearm surgery—A feasibility study," *The international journal of medical robotics + computer assisted surgery,* vol. 18, no. 5, pp. e2438-n/a, 2022, doi: 10.1002/rcs.2438.

[12] C. Li *et al.*, "Accuracies of bone resection, implant position, and limb alignment in robotic-arm-assisted total knee arthroplasty: a prospective single-centre study," *Journal of orthopaedic surgery and research,* vol. 17, no. 1, pp. 61-61, 2022, doi: 10.1186/s13018-022-02957-1.

[13] J. Goncalves, "Resection Surface Analysis Technical Report ", Australian Institute of Robotic Orthopaedics, FRM-110:1, 2017.

[14] L.-d. Wu, H. J. Hahne, and J. Hassenpflug, "The dimensional accuracy of preparation of femoral cavity in cementless total hip arthroplasty," *Journal of Zhejiang University. A. Science,* vol. 5, no. 10, pp. 1270-1278, 2004, doi: 10.1631/jzus.2004.1270.

[15] O. Günther and E. Wong, "A dual approach to detect polyhedral intersections in arbitrary dimensions," *BIT,* vol. 31, no. 1, pp. 2-14, 1991, doi: 10.1007/BF01952778.

[16] E. Technische, H. Zirich, B. Heidelberger, M. Teschner, and M. Gross, "Volumetric Collision Detection for Deformable Objects," 05/08 2003.

[17] K. G. Vince, A. Abdeen, and T. Sugimori, "The unstable total knee arthroplasty: causes and cures," *The Journal of arthroplasty,* vol. 21, no. 4 Suppl 1, pp. 44-49, 2006, doi: 10.1016/j.arth.2006.02.101.

[18] G. D. Rajitha Gunaratne, R. Khan, D. Fick, B. Robertson, N. Dahotre, and C. Ironside, "A review of the physiological and histological effects of laser osteotomy," *Journal of medical engineering & technology,* vol. 41, no. 1, pp. 1-12, 2017, doi: 10.1080/03091902.2016.1199743.

[19] N. Vaidya, T. N. Jaysingani, T. Panjwani, R. Patil, A. Deshpande, and A. Kesarkar, "Assessment of accuracy of an imageless hand-held robotic-assisted system in component positioning in total knee replacement: a prospective study," *Journal of robotic surgery,* vol. 16, no. 2, pp. 361-367, 2022, doi: 10.1007/s11701-021-01249-w.

[20] M. Fabrizio, B. Edoardo, G. Niccolò, C. Roberto, and I. Bernardo, "How reproducible are clinical measurements in robotic knee surgery?," *Journal of experimental orthopaedics,* vol. 10, no. 1, pp. 32-6, 2023, doi: 10.1186/s40634-023-00582-3.

[21] D. Kim, H. Owada, N. Hata, and T. Dohi, "An Er:YAG laser bone cutting manipulator for precise rotational acetabular osteotomy," 2004, vol. 1: IEEE, pp. 2750-2753, doi: 10.1109/IEMBS.2004.1403787.

[22] S. Rupprecht, K. Tangermann, P. Kessler, F. W. Neukam, and J. Wiltfang, "Er:YAG laser osteotomy directed by sensor controlled systems," *Journal of cranio-maxillo-facial surgery,* vol. 31, no. 6, pp. 337-342, 2003, doi: 10.1016/j.jcms.2003.07.007.

[23] A. Žemaitis, M. Gaidys, M. Brikas, P. Gečys, G. Račiukaitis, and M. Gedvilas, "Advanced laser scanning for highly-efficient ablation and ultrafast surface structuring: experiment and model," *Scientific reports,* vol. 8, no. 1, pp. 17376-14, 2018, doi: 10.1038/s41598-018-35604-z.

[24] B. Li, J. Wei, L. Wang, B. Ma, and M. Xu, "A comparative analysis of two point cloud volume calculation methods," *International journal of remote sensing,* vol. 40, no. 8, pp. 3227-3246, 2019, doi: 10.1080/01431161.2018.1541111.

[25] H. Noborio, S. Fukuda, and S. Arimoto, "Construction of the octree approximating three-dimensional objects by using multiple views," *IEEE transactions on pattern analysis and machine intelligence,* vol. 10, no. 6, pp. 769-782, 1988, doi: 10.1109/34.9101.

[26] L. Jeřábková, G. Bousquet, S. Barbier, F. Faure, and J. Allard, "Volumetric modeling and interactive cutting of deformable bodies," *Progress in biophysics and molecular biology,* vol. 103, no. 2, pp. 217-224, 2010, doi: 10.1016/j.pbiomolbio.2010.09.012.

[27] R. Dewil, P. Vansteenwegen, and D. Cattrysse, "Construction heuristics for generating tool paths for laser cutters," *International journal of production research,* vol. 52, no. 20, pp. 5965-5984, 2014, doi: 10.1080/00207543.2014.895064.

[28] B. Kainz, M. Grabner, A. Bornik, S. Hauswiesner, J. Muehl, and D. Schmalstieg, "Ray casting of multiple volumetric datasets with polyhedral boundaries on manycore GPUs," *ACM transactions on graphics,* vol. 28, no. 5, pp. 1-9, 2009, doi: 10.1145/1618452.1618498.

[29] *Geometrical product specifications (GPS) -- Flatness -- Part 1: Vocabulary and parameters of flatness (Vol. 1).* , I. O. f. Standards., London, United Kingdom, 2011.

[30] *Geometrical Product Specifications (GPS) -- Surface texture: Profile method -- Terms, definitions and surface texture parameters (Vol. 1).* I. O. f. Standards., London, United Kingdom, 1997.

[31] S. Toksvig-Larsen and L. Ryd, "Surface characteristics following tibial preparation during total knee arthroplasty," *The Journal of arthroplasty,* vol. 9, no. 1, pp. 63-66, 1994, doi: 10.1016/0883-5403(94)90138-4.

[32] A. P. Davies, "Rating systems for total knee replacement," *The knee,* vol. 9, no. 4, pp. 261-266, 2002, doi: 10.1016/S0968-0160(02)00095-9.

[33] T. Zhang *et al.*, "S 3 -Slicer: A General Slicing Framework for Multi-Axis 3D Printing," *ACM transactions on graphics,* vol. 41, no. 6, pp. 1-15, 2022, doi: 10.1145/3550454.3555516.

[34] R. Dewil, P. Vansteenwegen, and D. Cattrysse, "A review of cutting path algorithms for laser cutters," *International journal of advanced manufacturing technology,* vol. 87, no. 5-8, pp. 1865-1884, 2016, doi: 10.1007/s00170-016-8609-1.

[35] R. Breuer, "Laser Project v4 Overview," Australian Institute of Robotic Orthopaedics, DOC-PRO-101.1, 2022, vol. 4.

[36] S. Withers, "Laser Beam Propagation," Australian Institute of Robotic Orthopaedics, DOC-PRO-024.1, 2021.

[37] K. Mamou and F. Ghorbel, *A simple and efficient approach for 3D mesh approximate convex decomposition*. 2009, pp. 3501-3504.

[38] T. Möller and B. Trumbore, "Fast, Minimum Storage Ray-Triangle Intersection," *Journal of Graphics Tools,* vol. 2, no. 1, pp. 21-28, 1997/01/01 1997, doi: 10.1080/10867651.1997.10487468.

[39] OpenGL. "Graphics with OpenGL." https://opengl-notes.readthedocs.io/en/latest/topics/transforms/viewing.html (accessed 2023).

[40] W. Li, "Gpu-based computation of voxelized minkowski sums with applications," ProQuest Dissertations Publishing, 2011.

[41] J.-M. Lien, "A Simple Method for Computing Minkowski Sum Boundary in 3D Using Collision Detection," (Springer Tracts in Advanced Robotics. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 401-415.

[42] E. de Vries, E. Sánchez, D. Janssen, D. Matthews, and E. van der Heide, "Predicting friction at the bone – Implant interface in cementless total knee arthroplasty," *Journal of the mechanical behavior of biomedical materials,* vol. 128, pp. 105103-105103, 2022, doi: 10.1016/j.jmbbm.2022.105103.

[43] N. Kelly, D. T. Cawley, F. J. Shannon, and J. P. McGarry, "An investigation of the inelastic behaviour of trabecular bone during the press-fit implantation of a tibial component in total knee arthroplasty," *Medical engineering & physics,* vol. 35, no. 11, pp. 1599-1606, 2013, doi: 10.1016/j.medengphy.2013.05.007.