BAŞKENT UNIVERSITY
INSTITUTE OF SCIENCE AND ENGINEERING

DEVELOPMENT OF A FRAMEWORK FOR SURGERY ROBOTICS

MURAT BİLEN

MSc. Thesis
2012

DEVELOPMENT OF A FRAMEWORK FOR SURGERY ROBOTICS

AMELİYAT ROBOTLARI İÇİN YAZILIM İSKELETİ GELİŞTİRME

MURAT BİLEN

Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Department of Electrical and Electronics Engineering at Başkent University

2012

“Development of a framework for surgery robotics.” başlıklı bu çalışma, jürimiz tarafından, 06/06/2012 tarihinde, **Elektirik-Elektronik MÜHENDİSLİĞİ ANABİLİM DALI 'nda YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Başkan	:Prof. Dr. Mitat Uysal
Üye (Danışman)	:Yrd. Doç. Dr. Mustafa Doğan
Üye	: Yrd. Doç. Dr. E. İlhan Konukseven

ONAY

…/06/2012

Prof. Dr. Emin AKATA

Fen Bilimleri Enstitüsü Müdürü

To my Wife (Derya Kekeç Bilen).

ACKNOWLEDGMENTS

I would like to thank my supervisor Assist.Prof.Dr. Mustafa DOĞAN for his invaluable contributions, support, patience and encouragement throughout my dissertation. I would not be able to complete this dissertation without his guidance and recommendations. I would also like specially to thank to Assist. Prof. Dr. E. İlhan Konukseven for giving me inspiration and for his useful advices and contributions.

I would like to thank GATE Elektronik A.Ş. and TR-Teknoloji Ltd. Şti. for their patience throughout my postgraduate study. I am really thankful to my family, Haydar, Gülser and İrem Bilen who gave their endless love and support.

Finally, I thank to my wife Derya Kekeç Bilen, my greatest supporter, who has shared my feelings and experienced this thesis everyday as much as myself.

ABSTRACT

Robotic Surgery is the use of robots in the surgery. It is still a new technology and it is rapidly evolving. The main advantage of using robots in surgery is the high precision of the robots. Robots provide minimal invasions thus this helps patients to recover quicker, reduces the infection risks, reduces the pain and decreases the blood loss. They help surgeons to operate in smaller regions with higher sensitivity and to reduce the effort. Robotic surgery can be classified as model based and non-model based.

Model based robotic surgery uses a mathematical model which is generated from patients by MRI (Magnetic resonance Imaging) or CT (Computed Tomography) before surgery. The modeling process is done as a pre-processing operation before the surgery. In model based surgery the most important thing is the accuracy of the mathematical model and the registration process. The registration process is the process which matches the model in computer space with the patient in real world.

In non-model based surgery, the human machine interface plays a key role. Because of the non existence of a model, the surgeon navigates with the aid of the vision system typically by using a 3D stereo endoscopes or multi view Cameras. Sometimes the model cannot be created because of the large deformations or instability of the mathematical model thus the model becomes useless. Due to the lack of model in non-model based surgery, data acquisition and sensor fusion play an important role. Therefore, an identification model should be created from acquired data generally by using pattern recognition. On both methods - model or non-model based - force-feedback and the haptic devices play an important role in the success of the operation.

Building a Surgery robotics system is a very complex and difficult task for both Surgeon Side Network and Patient side network. The tasks need to be handled are controlling and preventing collisions of the robots, tools and manipulators attached to the robots, acquiring data from various sensors by DAQ Systems, filtering and processing this acquired data, applying pattern recognition techniques, or feeding the acquired data to fuse the sensors. Besides,

communicating with the FPGA, DSP boards, acquiring data from imaging devices such as endoscopes, image processing with multi view cameras and registration of the patient, displaying and analyzing the Medical Imaging Data, driving the controllers for surgeon, running a 3D vision system, creating a mathematical model for the force feed-back for the haptic devices, running physics processing for that model, rendering the processed results, driving the haptic devices, creating a network communication, and processing a huge amount of data and more.

There is no framework for doing all these tasks. Therefore, throughout this thesis a powerful platform that can handle all of these tasks is created from the scratch and named DMPlatform. In this thesis the architecture of the platform is examined and some applications done with the platform is presented.

Keywords: Soft-tissue simulation, physics engine, collision detection, rendering engine, machine vision, stereo vision, mri, surgery robotics, haptic devices, portability, platform independency, plug-in architecture.

Supervisor: Assist. Prof. Dr. Mustafa DOĞAN – Doğuş University – Control Engineering Dept.

ÖZ

Robotik Ameliyat, ameliyatlarda robotların kullanımı demektir. Henüz yeni bir teknoloji olsa da hızla gelişmektedir. Robotların ameliyatlarda kullanılmasının en büyük avantajı robotların yüksek hassasiyetidir. Robotların minimal kesiklerle operasyon yapabilmesi, enfeksiyon riskini, kan kaybını, kesiklerden kaynaklanan acıyı, hastanın toparlanma süresini ve hastane masraflarını azaltır. Robotlar cerrahların çok küçük bölgelerde çok yüksek hassasiyetlerde operasyon yapmasına izin verir ve cerrahın yorulmadan uzun süre operasyon yapabilmesine olanak tanır. Robotik ameliyatlar model tabanlı ve model tabanlı olmayan olarak ikiye ayrılır.

Model tabanlı robotik ameliyat, hastanın ameliyat öncesi çekilen MRI (Magnetic Resonance Imaging) yada CT (Computed Tomograpy) verisini kullanarak oluşturulan matematiksel modeli kullanır. Model çıkarma işlemi ameliyattan önce ön işlem olarak yapılır. Model tabanlı ameliyatlarda en önemli şey matematiksel modelin doğruluğu, hassaslığı ve hastanın koordinat uzayının model koordinat uzayı ile doğru birşekilde çakıştırılmasıdır.

Model tabanlı olmayan ameliyatta, insan makine arayüzü anahtar rol oynamaktadır. Model olmamasından dolayı, cerrah operasyonu endoskop veya çoklu görüşlü kameralar yardımı ile yapmaktadır. Bazen modelleme yapabilmek için veri bulunmasına rağmen, büyük defarmosyanların modellenememesi ya da modelin kararlı olmamasından dolayı model kullanılamaz hale gelir. Model olmamasından dolayı, model tabanlı olmayan ameliyatta veri yakalamak ve sensör füzyonu önemli bir rol oynar. Bu yüzden, genellikle örüntü tanıma yöntemleri kullanılarak, yakalanan verilerden tanımlama modeli oluşturulmalıdır. Haptic aygıtlar ve kuvvet geri beslemesi, her iki metod da -hem model tabanlı hemde model tabanlı olmayan- operasyonun başarısı için önemli bir rol oynar.

Ameliyat robotu tasarlamak, hem cerrah tarafındaki sistem hemde hasta tarafındaki sistem için çok karmaşık ve zor bir iştir. Yapılması gereken görevler, robotların, araçların ve robotlara bağlı olan manipulatörlerin denetimi, çarpışmalarının engellenmesi, birçok sensörden veri toplanması, toplanan verilerin filtrelenmesi, işlenmesi, sensör füzyonu amaçlı diğer sensörlerin

beslenmesi,örüntü tanıma tekniklerinin uygulanmasıyla başlar. Ayrıca, FPGA ve DSP kartları ile haberleşme, endoskop ve çok görüşlü görüntüleme sistemlerinden veri yakalanması, çok görüşlü kameralar ile görüntü işleme tekniklerinin uygulanması, hastanın ve bilgisayar modelinin koordinat sistemlerinin gerçek zamanlı çakıştırılması, tıbbi görüntü verilerinin gösterilmesi ve analizi, cerrahın kullandığı kontrolcülerin sürülmesi, 3B görüş sisteminin kurulması, kuvvet geribeslemesi yapabilmek için matematiksel modelin oluşturulması, matematiksel modelin fizik kurallarına uygun olarak simulasyonunun yapılması, simulasyon sonuçlarının fotogerçekçi şekilde görselinin oluşturulması (rendering), haptic aygıtların sürülmesi, ağ iletişiminin yapılması ve büyük boyuttaki verilerin işlenmesidir.

Tüm bu görevleri yapabilmek için bir tümleşik yazılım sistemi bulunmamaktadır. Bu nedenle bu tezde tüm bu görevleri yapacak tümleşik ve güçlü bir platform sıfırdan geliştirilmiş, DMPlatform olarak adlandırılmıştır. Bu tezde platformun mimarisi ve kullanılan teknikler anlatılmış ve platformla yapılan bazı uygulamalar sunulmuştur.

Anahtar Kelimeler: Yumuşak doku simulasyonu, fizik motoru, çarpişma tespiti, grafik motoru, stereo görüntü işleme, makine görüş sistemleri, mri, ameliyat robotlari, haptic aygitlar, taşinabilirlik, platformdan bağimsizlik, tak ve çaliştir mimari.

Danışman: Yrd. Doç. Dr. Mustafa DOĞAN – Doğuş Üniversitesi – Kontrol Mühendisliği Bölümü.

TABLE OF CONTENTS

ABSTRACT i
ÖZ iii
LIST OF FIGURES viii
LIST OF TABLES xiii
LIST OF EQUATIONS xiv
ABBREVATIONS xv
1. INTRODUCTION 1
1.1. Background and Motivation 1
1.1.1 Haptic devices 1
1.1.2 Medical scanners 2
1.1.3 Evaluating MRI data 5
1.1.4 Surgery robots 5
1.1.5 An example surgery robotic system 7
1.1.6 Types of robotic surgery 11
1.2 Problem Statement 14
1.3 Contributions 17
1.4 Outline 18
2. LITERATURE REVIEW 19
2.1 Robotic Devices In Surgical Applications 19
2.2 Haptic Devices in Surgery Robotics 19
2.3 Modelling The Force 20
3. DMPLATFORM 21
3.1 Design Criteria 22
3.2 Platform Components 24
3.2.1 Libraries: 24
3.2.2 Engines: 26
3.3 DMPhysics – Physics Engine 28
3.3.1 Simulators: 30
3.3.2 Force accumulator base object 30
3.3.3 Integrator 34
3.3.4 Physics base object 35
3.3.5 Particle 36

3.3.6 Rigid body object 37
3.3.7 Maxwell-mesh 39
3.3.8 Soft body 39
3.3.9 Grids 40
3.3.10 Hierarchical grids 41
3.4 DMCollide – Collision Detection Engine 42
3.4.1 Fundamental collision tests 42
3.4.2 Bounding volumes 43
3.4.3 Advantages & disadvantages of the bounding volumes 44
3.4.4 Compound 46
3.4.5 Convex hull 47
3.4.5 Bounding volume hierarchies 51
3.4.6 Oc-tree mesh hierarchy 52
3.4.7 Coarse collision detector 56
3.4.8 Narrow-phase collision detector 58
3.4.9 Collision resolver 59
3.4.10 Collision Response Solver 62
3.5 DMRenderer – Rendering Engine 63
3.5.1 Transformation 64
3.5.2 Lighting 66
3.5.3 GPU architecture 66
3.5.4 Architecture of DMRenderer 68
3.5.5 DMShader – shader engine 77
3.5.6 Volume rendering techniques 78
3.5.7 Tessellation 85
3.5.8 Culling 88
3.5.9 Per-pixel lighting techniques 92
3.6 DMIO 104
3.6.1 Controller interface 105
3.6.2 Haptic device interface 105
3.6.3 DAQ interface 105
3.6.4 Sensor interface 105
3.6.5 Serial line interface 105

3.6 DMVision....106
3.7 DMAI – AI / ML Engine....107
3.8 DMNetwork – Network Engine....108
3.8.1 TCP/IP....108
3.8.2 UDP....108
3.8.3 SLIP....108
3.9 DMCore....109
3.9.1 Resource manager....109
3.9.2 Scripting system....109
3.9.3 Linker system....109
3.9.4 Memory manager....110
3.9.5 Ram disk....110
3.9.6 Streaming system....110
3.9.7 File system....110
3.9.8 Thread manager....110
4. APPLICATIONS....111
4.1 Cancer Identification....111
4.2 Soft-Tissue Sampling....118
4.3 MRI – Haptic Segmentation....121
4.3.2 Results....127
5. CONCLUSIONS....128
REFERENCES....132
APPENDIX....136
APPENDIX A – ABBREVIATIONS....137
APPENDIX B - TRANSFORMATION EQUATIONS....140
APPENDIX C - PHYSICS EQUATIONS....147

LIST OF FIGURES

Figure 1- Reaction forces of a soft-body to the penetrating object 1
Figure 2 - Force feedback gaming wheel and joystick..... 2
Figure 3 - Sensable Omni and Phantom 6-DOF Haptic Devices..... 2
Figure 4 - An MRI slice of head 2
Figure 5 - MRI scanner and MRI data 3
Figure 6 - MRI scanner Internals 3
Figure 7 - CT scanner and CT data..... 4
Figure 8 - PET scanner and PET data..... 4
Figure 9 - A voxel from MRI data..... 5
Figure 10 - Six tetrahedrons constructing a voxel 5
Figure 11 - Open vs minimally invasive surgery 6
Figure 12 - Example surgery robotic system 7
Figure 13 - Example tele-surgery system..... 8
Figure 14 - Stereo endoscope 9
Figure 15 - Surgery simulation 12
Figure 16 - Shared surgery..... 12
Figure 17 - Pre-programmed, machine controlled surgery 13
Figure 18 - On-site surgery (with or without haptic)..... 13
Figure 19 - Tele-surgery..... 14
Figure 20 - ZEUS surgery robot 14
Figure 21 - Da Vinci surgery robot..... 15
Figure 22 - DLR-micro Surge and SOPHIE surgery robots 15
Figure 23 - Five layers of the platform..... 21
Figure 24 - Detailed architecture of DMPlatform 22
Figure 25 - Physics loop..... 28
Figure 26 - DMPhysics architecture 28
Figure 27 - Sample accumulators..... 31
Figure 28 - Linking objects and simulators with accumulators..... 31
Figure 29 - Objects are linked with simulators and accumulators..... 31
Figure 30 - Relationship between physics objects..... 35
Figure 31 - Base physics object container..... 35
Figure 32 - Particle object..... 36
Figure 33 - On particle update..... 36
Figure 34 - Rigid body object..... 38
Figure 35 - Rigid body update 38
Figure-36 - Maxwell element 39
Figure 37 - Structural (left) and shear (right) Maxwell mesh..... 39
Figure 38 - Bend Maxwell mesh (left) and Maxwell mesh in 3D (right)..... 40
Figure 39 - Example grids Maxwell grid (left) fluid grid (right)..... 40
Figure 40 - Hierarchical Maxwell grid 41
Figure 41 - Collision detection loop 42

Figure 42 - Collision test on two triangles 43
Figure 43 - Some examples of bounding volumes 44
Figure 44 - Sphere: Initial state (left) and rotated (right) 45
Figure 45 - AABB : Initial state (left) and rotated (right) 45
Figure 46 - OBB : Initial state (left) and rotated (right) 46
Figure 47 - Ellipsoid, Initial state (left) and rotated (right) 46
Figure 48 - Compound generated from Axis Aligned Bounding Boxes 47
Figure 49 - Collision tree of the compound below 47
Figure 50 - Original model 48
Figure 51 - Sampled vertices 48
Figure 52 - Generated mesh from vertices 48
Figure 53 - Maximum points found in blue 48
Figure 54 - Connected to each other 49
Figure 55 - Deleted vertices inside this volume 49
Figure 56 - Maximum distanced vertices from planes found in blue 49
Figure 57 - These vertices connected with hull and vertices inside are cleared 49
Figure 58 - Algorithm continues with other planes 50
Figure 59 - Maximum points found again 50
Figure 60 - Maximum points added to the hull and inside the hull is cleared 50
Figure 61 - There is no vertices left the algorithm finished 50
Figure 62 - Convex parts are covered successfully but concave parts are not 51
Figure 63 - Example Hierarchy of a desk mesh 51
Figure 64 - Example hierarchy of a car mesh 52
Figure 65 - Mesh is placed into bounding sphere 52
Figure 66 - Each 4 vertices placed into a AABB 53
Figure 67 - Each 4 AABBs placed into an AABB 53
Figure 68 - Finally last 4 AABBs placed into one AABB 53
Figure 69 - Example collision scenario 54
Figure 70 - Example collision test with oc-tree bounding volume hierarchy 54
Figure 71 – Test objects in the scene 56
Figure 72 - Each object should be tested with others in the scene 57
Figure 73 - Total intersection count for 28 objects 57
Figure 74 - Applying the grid to the scene 57
Figure 75 - Objects are arranged into corresponding grid cell 58
Figure 76 - Oc-tree hierarchy of the grid 58
Figure 77 - Example of a false collision 59
Figure 78 - Detected collision at delta t 60
Figure 79 - Object positions at t=0 60
Figure 80 - Object positions at t = delta t / 2 ; 61
Figure 81 - Object positions at t = 3 delta T / 4; 61
Figure 82 - Object Positions in 9 delta T/8; there is collision go back in time 62
Figure 83 - Final collision positions found; Collision report will be created. 62
Figure 84 - Basic rendering pipeline 63
Figure 85 - Rendering Pipeline 63

Figure 86 - Object and camera in local coordinate system. 64
Figure 87 - Objects are transformed to the world coordinate 64
Figure 88 - Objects are transformed to camera space with view matrix 65
Figure 89 - The visible volume is highlighted by green (viewing frustrum) 65
Figure 90 - 6 planes of view frustrum. 65
Figure 91 - Normal lighting (left) and global lighting model (right) 66
Figure 92 - GPU architecture 67
Figure 93 - Computer architecture 68
Figure 94 - DMRenderer architecture 68
Figure 95 - Vertex system 69
Figure 96 - Viewport system 69
Figure 97 - Buffer system 70
Figure 98 - Primitive system 71
Figure 99 - Texture system 71
Figure 100 - Mesh system 72
Figure 101 - Material System 72
Figure 102 - Off secreen render target system 73
Figure 103 - Sprite system 73
Figure 104 - Camera system 74
Figure 105 - Light system 74
Figure 106 - Render cache system 75
Figure 107 - Pre processing system 76
Figure 108 - Post processor 76
Figure 109 - The Shader Engine 77
Figure 110 - Transfer function of an MRI image 78
Figure 111 - Marching squares configuration 79
Figure 112 - Selecting the correct mesh 80
Figure 113 - Look-up table generated for the possible 16 configuration 80
Figure 114 - First 16 of 256 configurations of marching cubes 81
Figure 115 - Six tetrahedrons of a voxel 81
Figure 116 - Cube rendering of brain MRI 82
Figure 117 - Cube rendering example for debugging 82
Figure 118 - Rays from eye to the back plane 83
Figure 119 - Rays From eye intersecting the front face to back face 83
Figure 120 - Volume ray casting to the brain MRI data 84
Figure 121 - Volume ray casting using smaller cubes 84
Figure 122 - Tessellation of a triangle 85
Figure 123 - Continuous tessellation steps 86
Figure 124 - The original mesh (left) and discrete tessellated mesh (right) 86
Figure 125 - speedup with tesselation in 2D 87
Figure 126 - speedup with tesselation in 3D 87
Figure 127 - Culling steps in the preprocessing system 88
Figure 128 - Frustrum (left) and AABB of the frustrum (right) 88
Figure 129 - Frustrum AABB is tested with first boxes in the scene 89

Figure 130 - Oc-tree test continues 89
Figure 131 - The intersecting boxes are found 89
Figure 132 - Testing the remaining objects with the frustrum 90
Figure 133 - The outer volume of the objects are visible 90
Figure 134 - Self culling in volume ray casting 90
Figure 135 - Red boxes are occluded by the green ones. 91
Figure 136 - Red surfaces are back faces and not visible. 91
Figure 137 - Original mesh (Up) and down sampled mesh (Down) 92
Figure 138 - Texture channels; red, green, blue and alpha 92
Figure 139 - A 4 by 4 texture in memory 93
Figure 140 - Wall texture (left) and texture mapped to two by two mesh (right) ... 93
Figure 141 - Two by two mesh in Figure 140 is rendered. 94
Figure 142 - Example texture 95
Figure 143 - Mip maps of the example texture 95
Figure 144 - Tangent, normal, binormal of a surface 96
Figure 145 - Creating tangent and normal of the down sampled mesh 96
Figure 146 - Normals are converted to tangent space 97
Figure 147 - Normal map of the wall 97
Figure 148 - Rendering with texture 98
Figure 149 - Normal map applied 98
Figure 150 - Calculating the height offsets 99
Figure 151 - Height map of the wall (Stored in alpha channel) 99
Figure 152 - Calculation of normals and height offsets. 100
Figure 153 - Wall rendered with just using texture. 101
Figure 154 - Wall rendered with using normal maps 101
Figure 155 - Wall rendered with normal map and height offsets 102
Figure 156 - GPU softbody simulation example 103
Figure 157 - Displacement mapping 103
Figure 158 - I/O engine architecture 104
Figure 159 - Driver interface architecture 104
Figure 160 - Machine vision engine architecture 106
Figure 161 – An example filter graph 106
Figure 162 - Graph results stored in image slots 107
Figure 163 - Architecture of the DMNetwork 108
Figure 164 - Architecture of the DMCore 109
Figure 165 - 0 to 18 Khz spectrum of an instance 111
Figure 166 – Shadow of Arcene dataset 113
Figure 167 - Example LDA output 113
Figure 168 - Data acquisition. 119
Figure 169 – Path planning 119
Figure 170 - Tuning the PID controller coefficients 120
Figure 171 - Analyzing the material properties 120
Figure 172 - Section of a brain MRI Slice 121
Figure 173 - Gradients of the section marked in previous Figure 122

Figure 174 - Pre-processing of the MRI data.....123
Figure 175 - Intensity map (left); gradient map (Middle); edge map (right).....123
Figure 176 - effect of ray count.....124
Figure 177 - Active rays for 2D.....124
Figure 178 - Force calculation.....125
Figure 179 –CPU and GPU states on runtime.....125
Figure 180 - screenshot from the application 126
Figure 181 - Haptic Segmentation in action.....126

LIST OF TABLES

Table 1 - Constant gravity accumulator code 32
Table 2 - Example simulator running force accumulator object 32
Table 3 - Moderate Gravity force accumulator 33
Table 4 - Gravity calculation module 34
Table 5 - Advanced gravity accumulator 34
Table 6 - Oc-tree mesh hierarchy test table in 2D 55
Table 7 - Oc-tree mesh hierarchy test table in 3D 55
Table 8 - Tesselation speedup in 2D 87
Table 9 - Speedup in 3D 87
Table 10 - Arcene dataset features 112
Table 11 - LDA results for Arcene dataset 115
Table 12 - KNN results for Arcene dataset 116
Table 13 - Naive Bayes results for Arcene dataset 116
Table 14 - SVM (Linear) results for Arcene dataset 116
Table 15 - SVM (radial) results for Arcene dataset 117

LIST OF EQUATIONS

Equation 1 - Impulse 29
Equation 2 - Gravitation force 32
Equation 3 - Advanced gravity 33
Equation 4 - Newton 36
Equation 5 - Mass calculation of rigid body 37
Equation 6 - Center of mass calculation of rigid body 37
Equation 7 - Inertia calculation 37
Equation 8 - Inertia Tensor 37
Equation 9 - Torque 38
Equation 10 - Space needed for storing N mip-maps 95

ABBREVATIONS

GPU: Graphics Processing Unit

CPU: Central processing Unit

MRI: Magnetic Resonance Imaging

CT: Computed Tomography

PET: Positron Emission Tomography

Pixel: Picture element

Voxel: Volume element

1. INTRODUCTION

This chapter will try to summarize some background information about surgery robots, the tools and devices used for the robotic surgery.

1.1. Background and Motivation

Robots are mechanical devices that can perform tasks automatically by programming or with guidance by remote control. There are many applications such as industry, exploring deep seas, deep spaces and medical surgery. Their high precision movement capabilities are exploited in advanced processes.

1.1.1 Haptic devices

Haptic can be defined as the sense of touch. In physics, when our hand applies a force to an object, the object applies back a reaction force and our sensing system perceives this reaction force (Figure 1). Haptic devices are used to create this reaction forces in order to simulate a virtual touch sensing. In life there are lots of samples of haptic devices as seen on Figure 2(left), and Figure 2(right), the gaming wheel and gaming joysticks are used for increasing the virtual realism in the games with using force-feedback. In academic research more professional haptic equipments are used, like Sensable's Phantom Desktop Premium (Figure 3-right-) and Phantom Omni (Figure 3-left-).

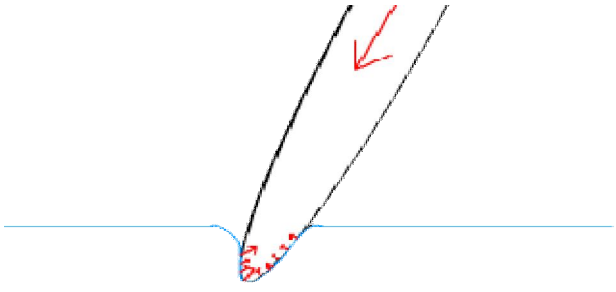


Figure 1-Reaction forces of a soft-body to the penetrating object

Figure 2 - force feedback gaming wheel and joystick

Figure 3 - Sensable Omni and Phantom 6-DOF Haptic Devices

1.1.2 Medical scanners

Medical Scanners help us to see behind the curtains. These scanners provide us a set of 2D medical imaging of the patient and the operation region. These 2D images give us a clue about how the interested region can be in 3D. Figure 4 shows a section of an MRI data of a patients head. Doctors examine these images one by one to collect information about the patient's disease.

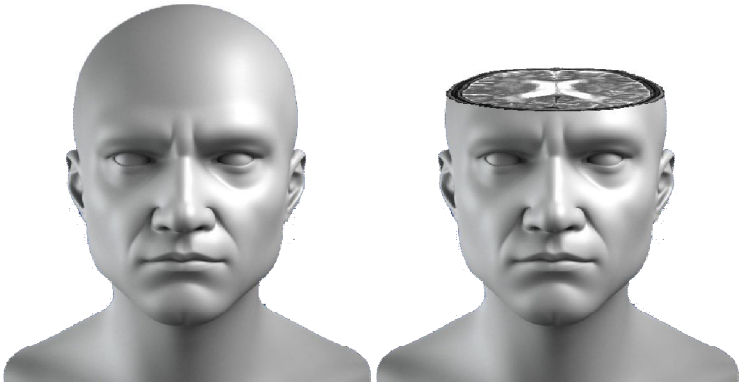


Figure 4 - An MRI slice of Head

1.1.2.1 MRI scanner

Radio waves are systematically used to alter the alignment of this magnetization, causing the hydrogen atoms to produce a rotating magnetic field detectable by the scanner. Figure 5 (left) shows an MRI scanner and Figure 5 (right) shows an example MRI data. Moreover, in Figure 6 internals of the MRI scanner is presented.

Figure 5 - MRI scanner and MRI data

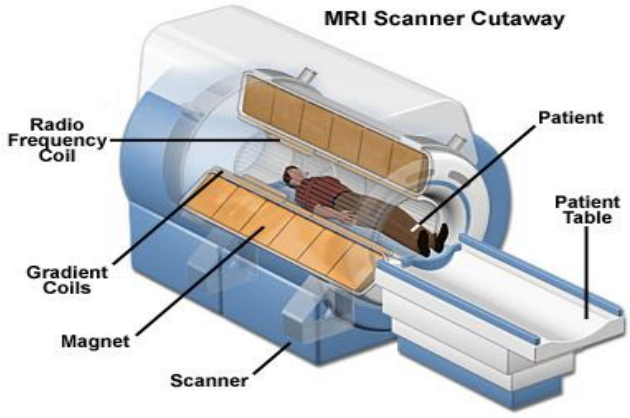


Figure 6 - MRI scanner Internals

1.1.2.2 CT scanner

CT scan is an x-ray imaging method where three-dimensional anatomy is deduced by geometric deduction from a large series of two-dimensional x-ray images taken around a single axis of rotation. An example of CT scanner and CT data can be seen below on Figure 7.

Figure 7 - CT Scanner and CT Data

1.1.2.3 PET scanner

Positron Emission Tomography is a method to measure Energy use in a body. Figure 8 is presented below as an example of PET scanner and PET data.

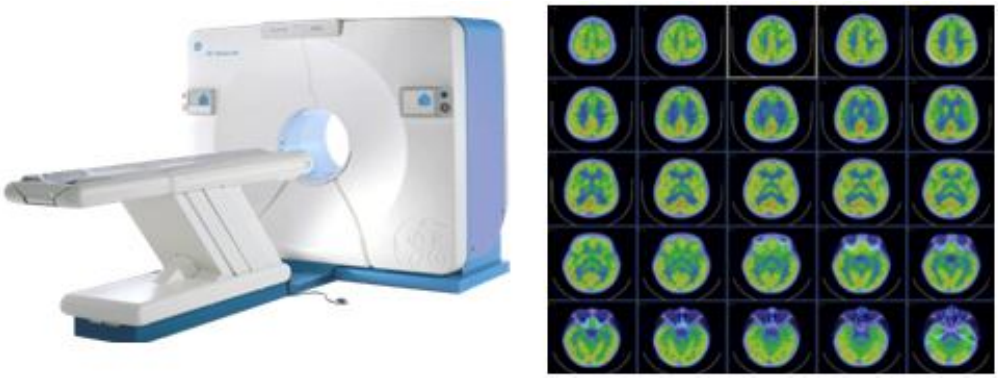


Figure 8 - PET Scanner and PET data

1.1.3 Evaluating MRI data

MRI Data is a set of 2D Medical Images constructed by pixels (picture element). These images are called slices. Patient's 3D Image is constructed by joining these 2D images as shown in Figure 9 and each neighboring 8 pixels creates a voxel (volume element) which is the smallest element of a volume. The mathematical model is created by these voxels. As seen on Figure 10, in order to achieve better precision, voxels can be divided into tetrahedrons. Each voxel contains 6 tetrahedrons.

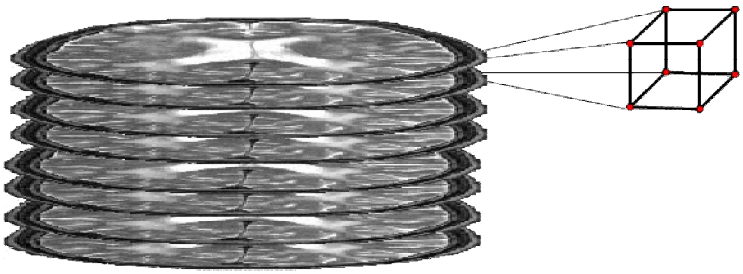


Figure 9 - A voxel from MRI data

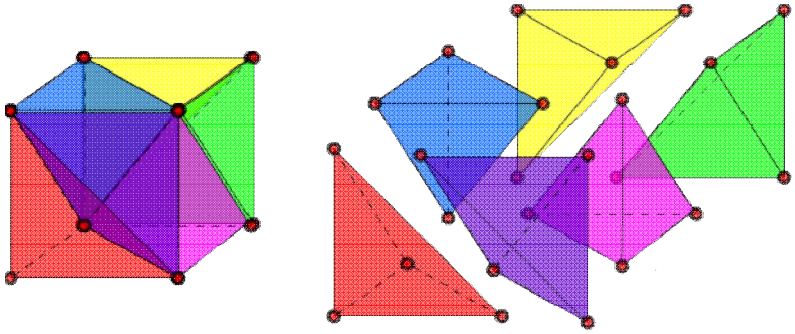


Figure 10 - Six tetrahedrons constructing a voxel

1.1.4 Surgery robots

Surgery methods are classified as open surgery and minimally invasive surgery. In open surgery in order to access internal tissues the surgeon has to make large incisions. This is necessary for the surgeon to see the operation site. But this

method delays patient recovery and causes a lot of pain. On the other hand, minimally invasive surgery the operations are performed by using special equipments that are designed to fit the body in small incisions. Minimizing the incisions decreases the patient's blood loss, reduces pain, the infection risks, recovery time, the hospital stay time and costs. Figure 11 shows the difference between open and minimally invasive surgery.

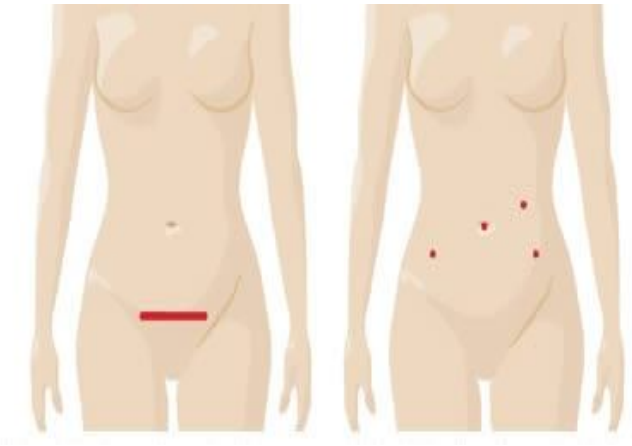


Figure 11 - Open Vs Minimally invasive surgery

Robotic Surgery is the use of robots in the surgery. It is still a new technology and its rapidly evolving. The main advantage of using robots in surgery is the high precision of the robots. Robots provide minimal invasions thus this helps patients to recover quicker, reduces the infection risks, reduces the pain and decreases blood loss. They help surgeons to operate in smaller regions with higher sensitivity and to reduce the effort. Robotic surgery can be classified as Model based and non-model based.

Model based robotic surgery uses a mathematical model which is generated from patients by MRI (Magnetic resonance Imaging) or CT (Computed Tomography) before surgery. The modeling process is done as a pre-processing operation before the surgery. In model based surgery the most important thing is the accuracy of the mathematical model and the registration process. The registration process is the process which matches the model in computer space with the patient in real world.

In non-model based surgery, the human machine interface plays a key role. Because of the non existence of a model, the surgeon navigates with the aid of the vision system typically by using a 3D stereo endoscopes or multi view Cameras. Sometimes the model cannot be created because of the large deformations or instability of the mathematical model thus the model becomes useless. Due to the lack of model in non-model based surgery, data acquisition and sensor fusion play an important role. Therefore, an identification model should be created from acquired data generally by using pattern recognition. On both methods - model or non-model based - force-feedback and the haptic devices play an important role in the success of the operation.

1.1.5 An example surgery robotic system

Building a Surgery robotics system is a very complex and difficult task both for Surgeon Side Network and the Patient side network. Figure 12 and Figure 13 show an example setup.

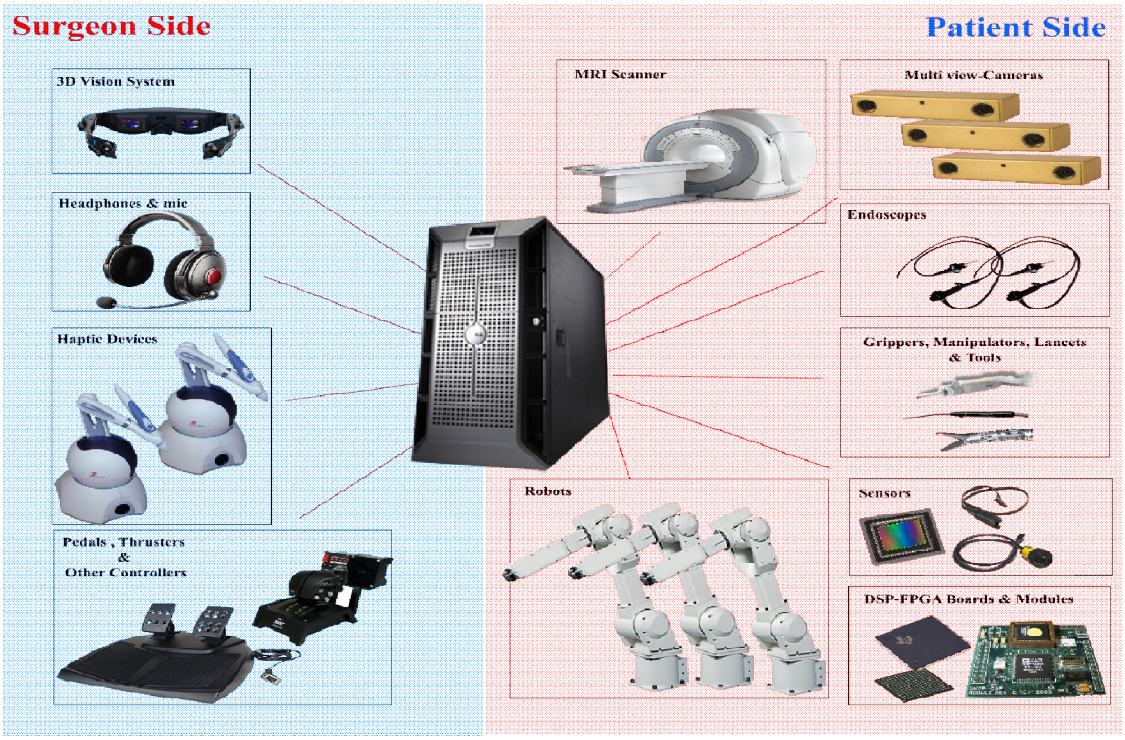


Figure 12 - Example surgery robotic system

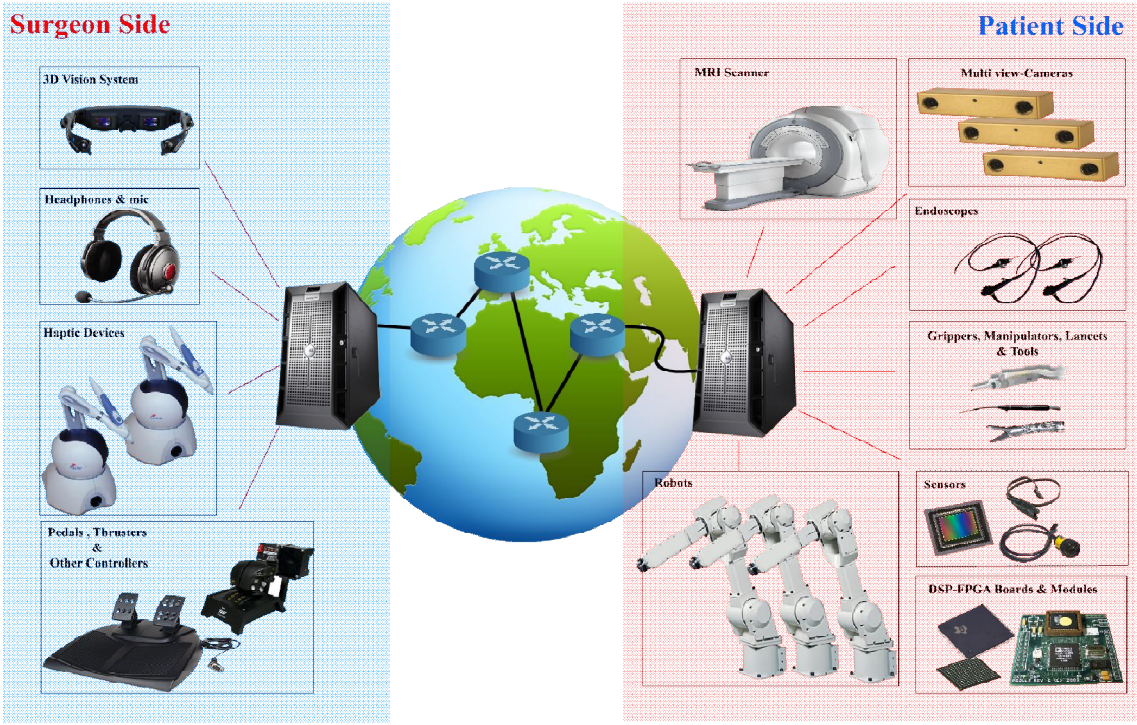


Figure 13 - Example Tele-Surgery System

1.1.5.1 Medical scanners

Medical Scanners provide a set of 2D medical images before surgery. These images are then processed by the processing system and the resulting 3D Image and the mathematical model are generated for exploring inside the patient and simulating the deformation caused by the operation.

1.1.5.2 Multi-view camera system

Multi-View camera system is responsible for acquisition of 3D Images of the patient and the surgery room for accurate 3D positioning and getting the initial data that is used for matching the model space and the patient space called registration process. The success of the operation depends on the registration of the patient accurately into the computer generated model. Therefore, before the surgery and the acquisition of MRI data, some markers are placed on the patient. These markers are highlighted in the MRI data which is associated with the mathematical model. Registration process transforms patient space into

computing space that mathematical model runs on and allows us to work in one space.

1.1.5.3 Grippers, manipulators, lancets and tools

Grippers, manipulators, lancets, tools can be defined as the tools attached to the robots that surgeon uses in operation. There are various types of tools and the usage of these tools varies depending on the type of operation.

1.1.5.4 Endoscopes

Endoscopes are used to get image stream while working inside the body. These micro cameras are put into the patient from small (0.5cm – 1.0 cm) holes attached to another robot. The most used endoscope in robotic surgery is the stereo endoscope, which has two micro cameras as seen from the Figure 14.

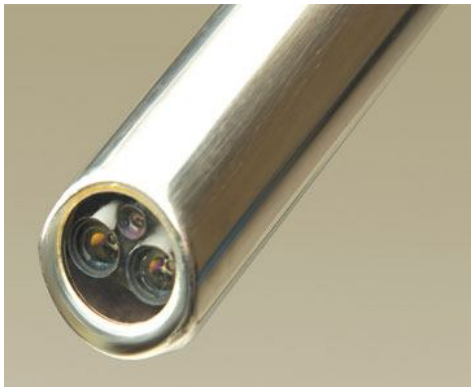


Figure 14 - Stereo endoscope

1.1.5.5 Robots

Robots are the hands of the surgeon at patient side which mimics the surgeon's movement. The key for the success is the good calibration, using precise positioning equipment, solid mechanical design and a good control system which is not in the scope of this thesis.

1.1.5.6 Sensors

Sensors or detectors are used to measure physical quantities. The use and type of the sensors change depending on the type of the operation. During the operation there is a huge amount of data that should be collected for identification, calibration and fusion purposes. The success of the operation relies on identification of the working environment accurately. For instance, if we are trying to remove a tumor from the brain which every smallest piece of the brain tissue has a responsibility for controlling the patient's body function, accidently removing a healthy brain tissue has a hazardous effect for the patient's body functions thus the healthy brain tissue and the tumor tissue should be carefully separated from each other. Moreover, environmental factors, such as conductivity, salinity, temperature, should be more accurately collected for fusing the main sensors.

1.1.5.7 DSP-FPGA-DAQ boards

Digital sensors are compatible with the computers but analog sensors are not. Analog data should be acquired by DAQ (Data Acquisition) systems which are designed for fast data acquisition. Neither DAQs nor sensors are perfect, they all subject to noises. Therefore, collected sensor data should be filtered, interpreted and analyzed as fastest as possible. But performing these operations on software is slow for real-time thus we use hardware processing via DSPs (Digital Signal Processors) designed for hardware digital processing or FPGA (Field Programmable Gate Array) boards used for building application specific integrated circuit chips.

1.1.5.8 Haptic devices

As described in the above sections, these devices are responsible for transmitting surgeon commands to the robots and creating virtual force feedback for the surgeon.

1.1.5.9 Pedals, thrusters, touch pads and other controllers

Humans only have two hands which are occupied by two haptic devices. However, the surgeon has to command not only robots but also other devices such as endoscopes, manipulators mounted on the robots etc. In order to give more control to the surgeon we use pedals, thrusters, touch pads and more. These controllers provide more control and sensitive adjustments to the surgeon.

1.1.5.10 Microphone – headphone

As stated above surgeon works two hands occupied. Thus, controlling the devices with speech processing helps him to get more control over the overall system.

1.1.5.11 3D- vision system

One of the most important thing in virtual realism is 3D Vision. Humans natively see the world in 3D for that reason a 3D viewing system should be provided to the surgeon. This system generates two images simultaneously for the surgeon one for the left eye and other for the right eye.

1.1.5.12 Computing system

All the collected data, surgeon commands, the physics calculations of the model and rendering calculations should be processed in a powerful computing system.

1.1.5.13 Network system

On separated systems, surgeon and the patient are in different network. Data and commands are transferred through Network System.

1.1.6 Types of robotic surgery

Depending on the use of devices or location, surgery robots can be classified into 6 classes.

1.1.6.1 Haptic training simulation

As illustrated in Figure 15, in simulation phase the aim is to train surgeons with different surgical experiments in a virtual reality environment in order to increase their experience or allow them to practice the operation before entering the surgery.

Figure 15 - Surgery Simulation

1.1.6.2 Shared surgery

Shared surgery is a type of surgery in which the surgeon controls the robot by bare hands as it is shown in Figure 16. The main aim of this type of surgery is to reduce the tremor effects and therefore provides more precise control to the surgeon.

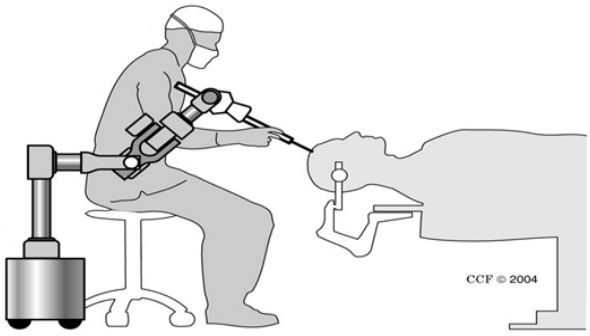


Figure 16 - Shared surgery

1.1.6.3 Pre-planned machine controlled

As shown in Figure 17, in this type of surgery the first step is the planning of the surgery in mission planning computer by the surgeon and then the next step is uploading the planned mission into the robot. In other words, robot does the surgery in line with the plan designed by the surgeon.

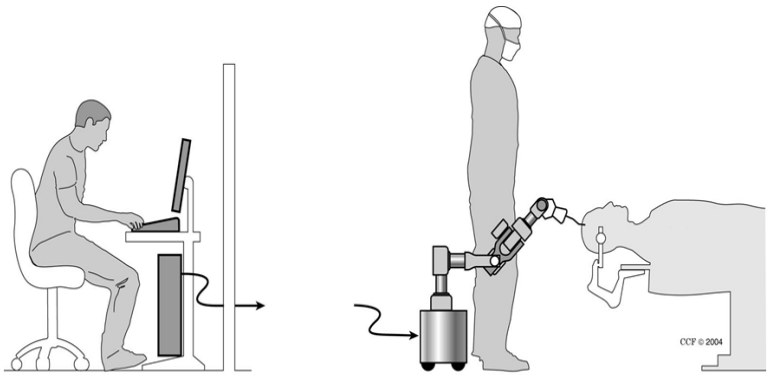


Figure 17 - Pre-programmed, machine controlled surgery

1.1.6.4 On-site surgery

As shown in Figure 18, the surgeon controls the robot from a command center which he/she can access all the sensor data.

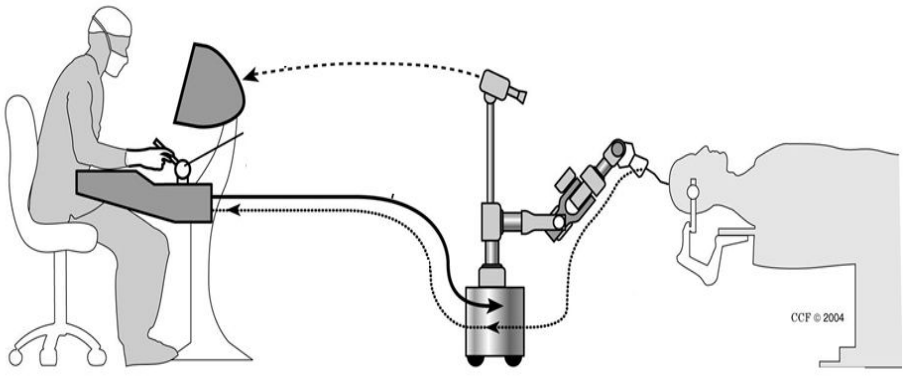


Figure 18 - On-site surgery (with or without haptic)

1.1.6.5 On site with haptic feedback

As shown in Figure 18, this type of surgery is very similar to the on-site surgery, only in this type the surgeon has the Haptic feedback.

1.1.6.6 Haptic tele-surgery

As shown in Figure 19, in this type surgery, surgeon is far away from the patient the surgery is performed by remote networking.

Figure 19 - Tele-surgery

1.2 Problem Statement

There are some commercial systems available on the market. Such as ZEUS (Computer Motion) (Figure 20), Da Vinci (Intuitive Surgical) (Figure 21), DLR-micro Surge (Figure 22) in all of these robots the surgeon commands are mimicked by robots.

Figure 20 - ZEUS surgery robot

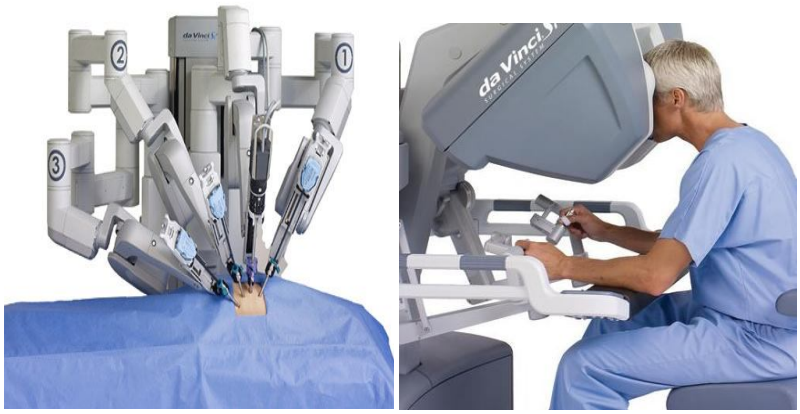


Figure 21 - Da Vinci surgery robot

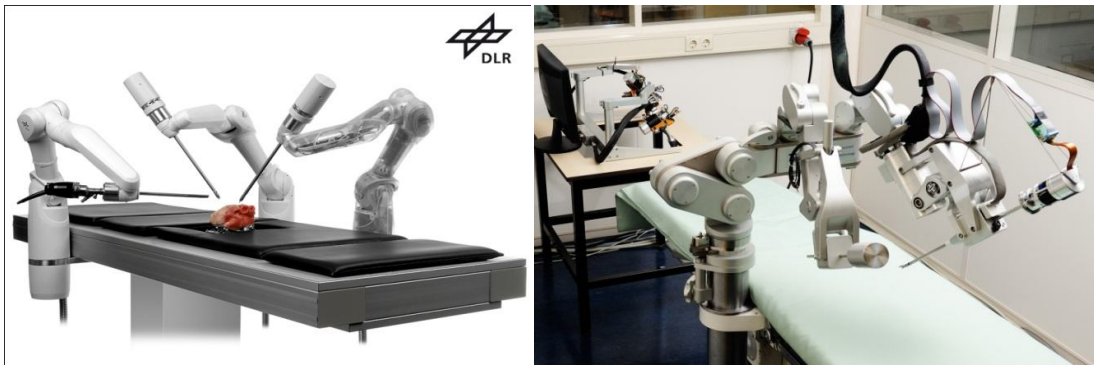


Figure 22 - DLR-micro Surge and SOPHIE Surgery Robots

First problem is the absence of haptic feedback which is crucial for surgeons used to operate with the presence of tissue forces. Moreover, as seen from the example surgery setup in Figure 13, second problem is the network and processing delays. In order to simulate forces and compensate the delays we need to create a mathematical model of the tissue and simulate it in a virtual reality environment. Since the size of the model data is huge, we need to process this data carefully.

For example in a 512*512*512 MRI data there are 511*511*511 = 133.432.831 voxels. Each voxel has 6 tetrahedrons, each tetrahedron has 4 triangles and each triangle has 3 vertices. The vertex count of this MRI data equals to 9.607.163.832. If size per vertex is 60Bytes then we need ≈ 600GB to store this mesh just for rendering, adding the physics mesh the size doubles to ≈1.2TB and there are lots of data to be processed in this time-delta like other meshes, textures, bounding

volumes, materials, maps , scripts, parameters, network messages, devices, device data, video images, signals and sensor data.

Human visual perception system is sensible to frame rates below 25Hz and cannot catch above 30Hz. Furthermore, touch system is sensible to update rates below 1000Hz. Thus we have to render for 2 eyes left and right. In short our system needs to be run at 60Hz which means in 16mS. All these data and all the surgery system should be processed in 16mS. More accurate timing can be approximated about 1mS to performing application logic, 1 mS to device handling, 1 ms to network, 1mS to resource transferring, 2mS to pattern recognition, 2 mS to image processing. In short, we have less time than we expected about 8mS to perform the physics simulations and rendering.

There are tools supplied by manufacturers like open haptics academic edition, ghost library, cyclops, sensor software. There are some open source projects like physics engines, rendering engines, machine vision libraries. But using such different software leads some incompatibilities listed below:

I. Different use of coordinate systems

For example, a physics engine can use left hand coordinate system but rendering engine can use right hand coordinate system. Thus, all of the transformation matrices, vectors, data, meshes calculated by physics engine should be converted to right hand coordinate system before rendering. This means we need some huge processing.

II. Different data types

For example, on the one hand physics engine can use extended precision on the other hand rendering engine can use double or float or half precision. This shows us the need to convert these between all the systems.

III. Different endienness

One can be written natively for Intel architecture or for the PowerPC or other architecture. Data endienness should be converted to each other.

IV. Different data packing

Each software package uses its own data packing mechanism. For example, physics engine can pack a vertex as Position, Normal; but rendering engine can pack as Normal, Position. Or one can pack vector as x, y, z and other one can pack as an array or as x, y, z, w. All these data should be converted before processing.

V. Different handling mechanisms

Each software package is designed for its own handling mechanism, states and caches thus they should be synchronized in the application layer.

VI. No shared resource

Each software package has its own resource handling mechanisms. They store and process the same data in different ways. For example, physics engine stores physics mesh differently than rendering engines rendering mesh. Therefore, it is necessary to convert these resources.

VII. Different Code Space

Each software package uses its own libraries. Thus in some cases the same tasks are handled twice or more. Such that physics engines use transformation routines, rendering engine uses it as well, and machine vision engine also uses it. Thus using different software means duplication of the same code twice or more. This leads to un-optimized memory usage and in some cases it can slow down the computing.

Type conversations can seem to be a small work but remember that we use billion vertices meshes. So type conversions are extremely dangerous in the manner of speed.

1.3 Contributions

Regarding the incompatibilities above, we need an optimized solution. Hence we built a stable, portable, platform independent, highly scalable, expandable

platform, named DMPlatform. Every component shares the same data structures, same resources, same handling mechanisms, same coordinate axes, same code space and providing a very flexible and complete architecture. Platform is built from scratch**.**

We built DMCore as a Low level layer for platform and API abstraction. Moreover, DMPhysics is developed as a detailed physics engine in order to model mathematically the tissue information. Last but not least, DMRenderer is built for for photo-realistic rendering and visualization. The whole applications and algorithms are developed as integrated and all-in-one approach.

1.4 Outline

This thesis consists of five chapters. Chapter 2 summarizes and reviews the related work in the field. Chapter provides information regarding DMPlatform and its Architecture. Chapter 4 continues with the related works done with DMPlatform. Finally, Chapter 5, finalizes this thesis and outlines future works

2. LITERATURE REVIEW

Robotic Surgery started by placing a needle for a brain biopsy by CT Guidance with a PUMA 560 Robot[1]. In 1988 PROBOT is developed in the Imperial Collage London. In 1992 PROBOT was developed by Integrated Surgical Systems. First Modern Surgical Robotic system was developed by Computer Motion named HERMES. Then the most successful surgery robotics system Da Vinci was built by Inutiative Surgical Inc. Today Da Vinci has thousands of surgery records officially.

2.1 Robotic Devices In Surgical Applications

The main target of the robotic surgery is to increase the surgeon's capabilities with integrating the precision of robots rather than replacing the surgeon with the robots [2].

The main advantage of using robots in surgery is the high precision of the robots. Robots provide minimal invasions thus this helps patients to recover quicker, reduces the infection risks, reduces the pain and decreases blood loss. They help surgeons to operate in smaller regions with higher sensitivity and reduce the effort [3]. By adding more degrees of freedom Robots provide great dexterity enhancement [4]. They decrease the surgeon's tremor [5,6].

The main advantage of using robotic surgery to computer supported surgery (which surgeons uses just computer analysis) is the precision, easily repetition of the movements , providing more movement capability and flexibility to the surgeon in small regions and provides extended surgery times spending less effort [7]. Especially in micro-surgery the hand vibration of the surgeon is decreased from ≈40μm to ≈4μm [2,8].

2.2 Haptic Devices in Surgery Robotics

Current generation surgery robotics systems do not contain one of the most important things for surgeons, the sense of touch. The success of complex surgical operations, rely on the force-feedback. There are some researches going for this subject [9,10]. Surgeons are used to manipulate the tissue in the presence

of forces thus without force-feedback they are subjected to some mental work and lack of force-feedback sometimes causes application errors [11, 12].

2.3 Modelling The Force

With the increasing computing power of today's processors, surgical simulations are seemed to be a viable solution. Simulators can be used for planning the surgery [13, 14] and most importantly training surgeons [15, 16]. In order to generate force feedback the soft-body and tool intersections should be modeled accurately [11, 17, 18, 19]. But there is a tradeoff between speed and accuracy. Accurate methods need more processing power and memory [20, 21]. And the challenging problem is the collision detection. Collision detection calculations are expensive procedures that should be handled carefully [22, 23, 24, 25].

One of the main challenges is the registration of the 3D Model with robotic coordinate space and the patient's coordinate space. The Robotic Surgery is an image guided system thus 3D image space, patient space and the robot space should be synchronized. Shared (Haptic Device-Robot) applications could overcome these problems by giving force-feedback and 3D navigation support and giving surgeons more precise control, movement capability and dexterity [26, 27].

3. DMPLATFORM

DMPlatform is built in order to overcome the problems discussed in the previous section. Platform can be examined in 5 layers as shown in Figure 23. The first layer is the bottom layer of the platform, DMCore, DMCore is responsible for communicating with the OS-Kernel for thread-process management, memory management and device management. Second level consists of libraries, third level incorporates engines for specific tasks, forth level includes the finite state machine which is responsible for injecting user code (Application Logic) into the Platform and finally fifth layer provides GUI editors for handling resources. Platform architecture is shown in Figure 24.

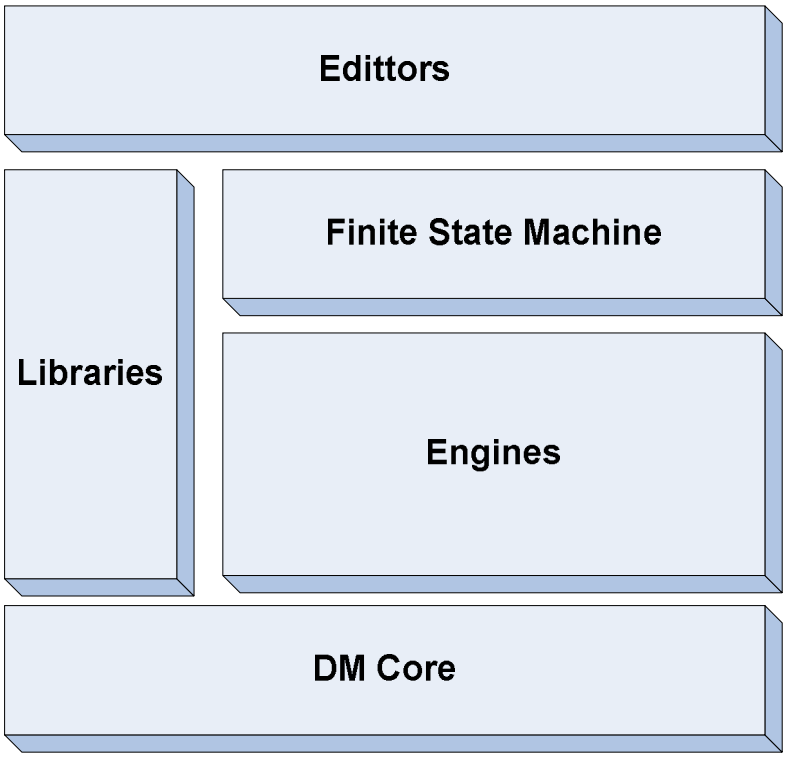


Figure 23 - 5 layers of the platform

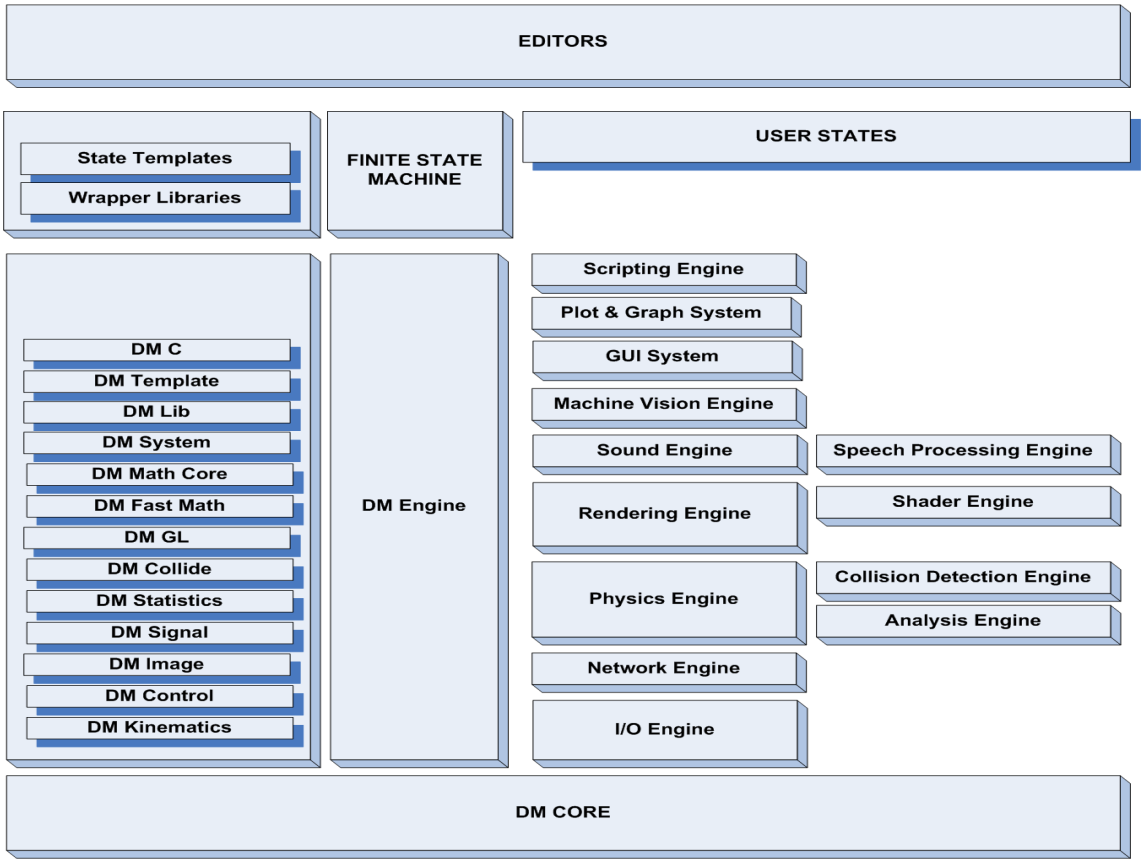


Figure 24 - Detailed architecture of DMPlatform

3.1 Design Criteria

The platform is carefully designed on the basis of below criteria:

- **One for all solution**

Platform should be easily extended for other solutions. For instance, it should not be used for only brain surgeries but also heart, kidney or like.

- **Easy to use**

Platform should be as easier as possible for supporting wide professions.

- **Portability**

Platform should run on different CPUs or GPUs.

- **Platform independency**

Platform should be run on different OSs.

- **Open architecture**

Platform should be transparent to the user thus he/she can interfere anytime, anywhere.

- **Overridablity**

Platform should be overridable. Users should inject their own implementation into every region in the pipelines.

- **Scalability**

Platform should be run on different scale platforms such as system on board modules to super computers.

- **Extendable**

Platform should be extendable such that any new engine or module should be integrated without interfering other engines.

- **Interfacability**

Platform is natively written in C/C++ and x86 ASM but components should be accessed from different computer languages. Such as easier languages like C#, visual basic etc.

- **Parallelized architecture**

CPUs reached their physical limits in the manner of clock speeds. The new trend is increasing the core amount per die. Thus in order to utilize parallel cores, platform should handle symmetric data processing

- **Symmetric architecture**

Using just CPU is not sufficient to process such amount of data thus we use the GPU for handling batch data. The CPU and the GPU should run symmetrically in every step of the calculations and data should be synchronized.

- **Distributed architecture**

Even if all the optimizations works well, still the system has limited processing throughput, thus adding second machine into the system theoretically doubles the processing power. Adding more than one machine into the system multiplies the throughput.

- **Speed**

This is the most important criteria of the platform. There is a huge amount of data to process as a result platform should perform calculations as fastest as possible.

3.2 Platform Components

The platform can be examined in 4 stages; the libraries, engines, core and editors.

3.2.1 Libraries:

In DMPlatform in order to increase reusability and share the same code space, common codes are kept in the libraries.

3.2.1.1 DMC

DMC is the re-implementation of the standard C library which provides cache and memory optimized re-implementation of some algorithms regarding to memory and strings which are dealing with batch data. DMC implementations provide significant speedups over standard C library.

3.2.1.2 DMTemplate

DMTemplate is the template library free of exception handling and creates significant performance decrement in batch processing.

3.2.1.3 DMLib

DMLib is the library for general purposes and contains algorithms for sorting and searching.

3.2.1.4 DMSystem

DMSystem is the system library which contains data structures and implementations regarding to memory management, process and thread, mutex, semaphore handling, file and disk management.

3.2.1.5 DMMathCore

DM math core contains basic math data structures and implementations shared with the lowest level of the platform.

3.2.1.6 DMMath

DMMath is a big math library which contains algorithms for numerical integration, differentiation, differential or linear equation solvers, matrix solvers, interpolators and complex algebra.

3.2.1.7 DMFastMath

DMFastMath contains fast approximations of some complex functions using taylor series expansions. Such as square root, trigonometric functions, cosine sin etc. It is more appropriate to use this library where precision is not an important feature like rendering calculations.

3.2.1.8 DMCollision

DMCollision contains all the data structures and routines for intersection tests. It is developed not only for collision detection engine but also it is open to all other engines.

3.2.1.9 DMGL

DMGL contains algorithms for graphics calculations like transformation, projection and more.

3.2.2 Engines:

3.2.2.1 DMEngine

DMEngine is the master engine in the platform. It handles all the engines, and the data shared by the engines and the resources. Furthermore it runs the user code.

3.2.2.2 DMPhysics:

DMPhysics is the most important engine in the platform. It creates a virtual world that is so close to real world and performs the simulations in this virtual world.

3.2.2.3 DMCollide:

DMCollide is the sub engine of DMPhysics and responsible for testing and reporting the collisions occurred in the virtual world.

3.2.2.4 DMAnalysis:

DMAnalysis is the second sub engine of the DMPhysics, provides some analysis techniques for more accurate physics like finite-element analysis, finite difference analysis, boundary element analysis. It is still in development. It has currently no function in the platform.

3.2.2.5 DMRenderer:

DMRenderer is the second most important engine in the platform. It simulates lighting equations and renders resulting physics calculations for the user.

3.2.2.6 DMVision:

DMVision is the machine vision engine which supports real time image processing on video streams and 3D image processing on MRI data.

3.2.2.7 DMAI:

DMAI is the machine learning and AI Engine.

3.3 DMPhysics – Physics Engine

DMPhysics generates a virtual world that is so close to real world. It allows physically modeling and simulation of Rigid-Bodies, soft bodies, fluids and other physics systems. DMPhysics is a force and impulse based physics engine that provides very close approximations to the exact solutions. The process of the system can be simplified into 3 steps as seen in the Figure 25. The first step is performing simulation of the environment and feeding the interaction forces to the objects and performing time integrations of the objects. The second one is detection of collisions between the objects. And the third step is calculating the collision responses for the colliding objects if there exist any. The detailed architecture of the physics engine is shown in Figure 26.

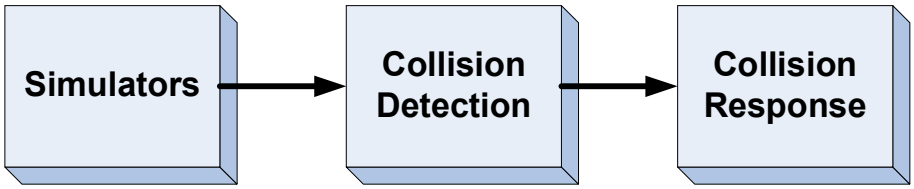


Figure 25 - Physics Loop

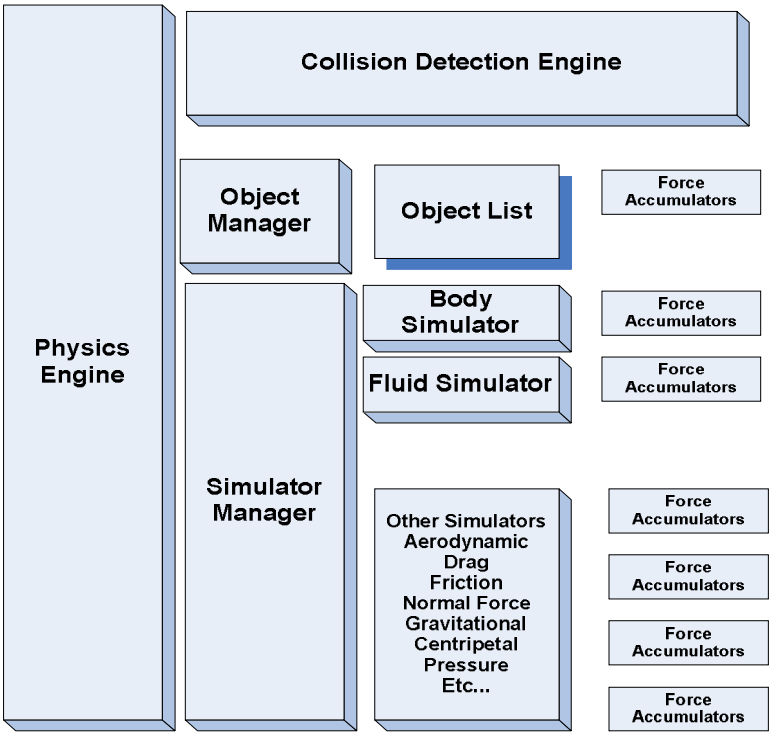


Figure 26 - DMPhysics architecture

As discussed above, DMPhysics is a force and impulse based physics engine. Therefore, it is necessary to define force and impulse in order to explain the inner works of the DMPhysics engine.

Force:

Force is an influence that changes the applied object's state, which is, change in acceleration, change of direction or change of geometric shape.

The Force is abbreviated as F.

Impulse:

Impulse is the integration of force, in a small Time-Delta.

Impulse is abbreviated as I or J.

In this document, the impulse will be abbreviated as J; for not to confuse with the Inertia symbol I.

Equation 1 - Impulse

$$\boldsymbol{J} = \int \boldsymbol{F}.\, dt \qquad (3.1)$$

Types of Forces:

There are 4 fundamental forces in physics.

Thrust: Changes the acceleration of the object.

Torque: Changes the rotational acceleration of an object.

Drag: Decreases the velocity of an object.

Stress: Changes the geometric shape of an object.

Forces in DMPhysics

In DMPhysics, four types of forces given above are implemented. Additionally sub types of these forces are implemented such as contact force which occurs from two colliding objects, normal force which is an anti-force generated from an object by contacting with another object, friction force which is a resisting force caused by the surfaces of the object, elastic force which occurs when deforming an object from its natural geometry, and other like tension, torque, fluid, pressure, centripetal and gravity forces.

After describing type of forces used in the engine, this section will continue with the explanation of physic loop elements.

3.3.1 Simulators:

Platform has various simulators for each of the natural forces. The first one is the body simulator which simulates particles, rigid bodies and soft-bodies. The second one is the fluid simulator which simulates fluids. And the third one is the electromagnetic simulator which is still in development.

3.3.2 Force accumulator base object

According to D'Alembert's principle net force acting on a body is equal to the summation of force vectors acting on that body. This means forces can be accumulated. Thus we use a special object named force accumulator for each object to simplify simulation and parallelism. Hence every simulator creates an accumulator for each object and stores the simulation results in its accumulator. At the end when the body simulator needs all the forces acting on that body, all the accumulators linked to that body are summed to find the net force and torque.

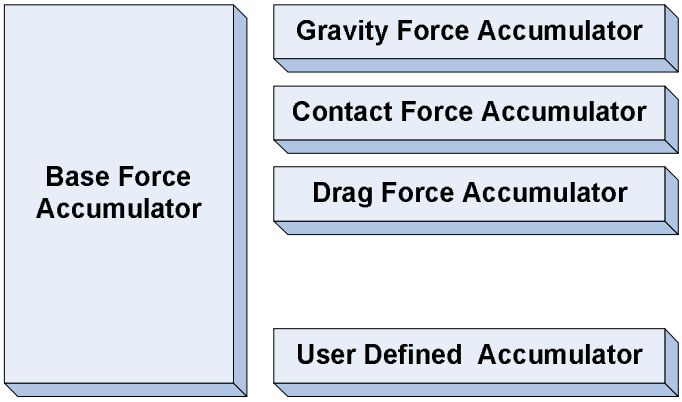


Figure 27 - Sample accumulators

Force accumulator is another object from the abstraction layer which constructs a base for other force accumulator.

Some types of force accumulators implemented in DMPhysics are: force accumulators, drag force accumulators, impulse accumulators and torque accumulators as seen on Figure 27.

Figure 28 and 29 show linking the force accumulators with the simulation objects.

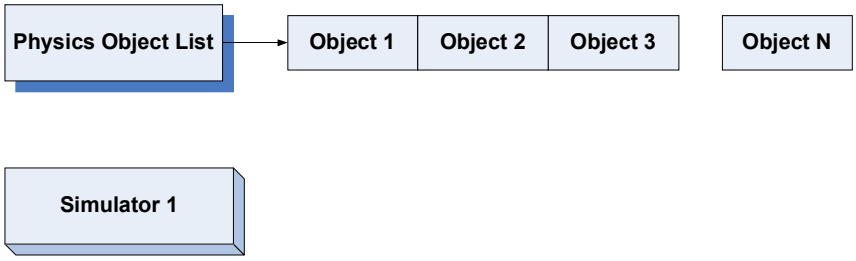


Figure 28 - Linking objects and simulators with accumulators

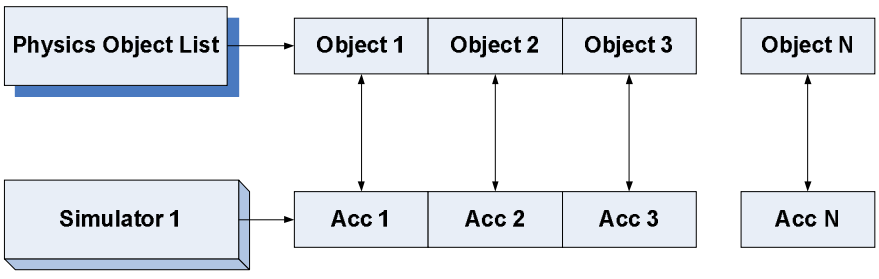


Figure 29 - Objects are linked with simulators and accumulators

Some examples regarding the usage of force accumulator and the overridability of platform components are given below.

Example 1: Simple gravity accumulator

Assume that gravity is constant and equal to 9.81 m/s^2

Force accumulator for constant gravity can be written as shown in Table 1:

Table 1 - Constant gravity accumulator code

```
//-----------------------------------------------------
//   Called by the Simulator on each frame update
//-----------------------------------------------------
void UserDefinedGravityAccumulator1::Accumulate(DMReal DeltaTime)
{
     TotalForce += DMVector3D(0.0, -9.81* DeltaTime, 0.0);
}
```

And the simulator can be written as shown in Table 2:

Table 2 - Example simulator running force accumulator object

```
//-----------------------------------------------------
//  Physics Engine Automatically Calls this Module
//-----------------------------------------------------
void   UserDefinedGravitySimulator1::Simulate(DMReal DeltaTime)
{
            //   For each accumulator in Accumulator list
            //   Calculate Gravity
            for(int i=0; i<NumberOfForceAccumulators; i++)
            {
                 AccumulatorList[i]->Accumulate(DeltaTime);
            }
}
```

Example 2: Moderate gravity accumulator.

Gravitational force between earth and the object.

Equation 2 - Gravitation force

$$F = G\sum \frac{M.\mathrm{m}}{r^2} \qquad (3.2)$$

Where; M is the mass of earth, m is the mass of the object r is the distance of object from center of earth and G is the universal gravitational constant.

Force accumulator can be as shown in Table 3:

Table 3 - Moderate Gravity force accumulator

```
//-----------------------------------------------------
//          Called by the Simulator on each frame update
//-----------------------------------------------------
void UserDefinedGravityAccumulatorl::Accumulate(DMReal DeltaTime)
{
   DMReal G  = GRAVITATION_CONSTANT;
   DMReal m1 = MASS_OF_EARTH;
   DMReal m2 = Object->GetMass();
   DMVector3D  ObjectPosition  = Object->GetPosition();
   DMVector3D  EarthPosition   = POSITION_OF_EARTH;
   DMVector3D  ObjectToEarth   = ObjectPosition - EarthPosition;

   DMReal r = DMVectorLength(ObjectToEarth);
   DMVectorNormalize(&ObjectToEarth);
   TotalForce += ObjectToEarth * ( (G*m1*m2)/(r*r))* DeltaTime;
}
```

Simulator is the same as example 1.

```
//-----------------------------------------------------
//  Physics Engine Automatically Calls this Module
//-----------------------------------------------------
void   UserDefinedGravitySimulator::Simulate(DMReal DeltaTime)
{
   //   For each accumulator in Accumulator list
   //   Calculate Gravity
   for(int i=0; i<NumberOfForceAccumulators; i++)
   {
       AccumulatorList[i]->Accumulate(DeltaTime);
    }
}
```

Example 3 : Advanced gravity accumulator.

Gravitation between object, earth and other objects.

Equation 3 - Advanced gravity

$$F = G\sum \frac{m.m_i}{r^2} \tag{3.3}$$

Where; m_i is the Mass of the earth and other objects, m is the mass of the object r is the distance of objects m and m_i G is the universal gravitational constant.

Force accumulator can be written as shown in Table 4 and Table 5:

Table 4 - Gravity calculation module

```
//-------------------------------------------------------------------------------------------
// Calculates Gravitation Force Between two Objects
//-------------------------------------------------------------------------------------------
DMVector3D CalculateGravity(DMVector3D Obj1Pos, DMVector3D Obj2Pos, DMReal m1, DMReal m2, DMReal DeltaTime)
{
            DMVector3D  Object1ToObject2 = ObjectPosition - EarthPosition;
            DMReal r =  DMVectorLength(Object1ToObject2);
            DMVectorNormalize(&ObjectToEarth);
            TotalForce += Object1ToObject2 * ( (GRAVITATION_CONSTANT*m1*m2)/(r*r))* DeltaTime;
}
```

Table 5 - Advanced gravity accumulator

```
//-----------------------------------------------------
//          Called by the Simulator on each frame update
//-----------------------------------------------------
void UserDefinedGravityAccumulator1::Accumulate(DMReal DeltaTime)
{
   DMReal Mass   Object->GetMass();
   DMVector3D ObjectPosition = Object->GetPosition();
   DMPhysicsObject * ObjectList = PhysicsEngine->GetObjectList();

   //   First Calculate Gravitation with Earth
   TotalForce += CalculateGravity(ObjectPosition,POSITION OF EARTH,Mass,MASS OF EARTH, DeltaTime);
   //   Then with other objects
   for(int i=0; i<ObjectIndex; i++)
       TotalForce += CalculateGravity(ObjectPosition,ObjectList[i] >GetPosition(), Mass,ObjectList[i] >GetMass(),
                                                                                          DeltaTime);

   for(int i=ObjectIndex+1; i<NumberOfObjectsInList; i++)
        TotalForce += Calculate_Gravity(ObjectPosition,ObjectList[i] >GetPosition(), Mass,ObjectList[i] >GetMass(),
                                                                                          DeltaTime);
}
```

Simulator is the same as example 1, as shown in Table 2.

3.3.3 Integrator

The integrator is one of the most important objects in the physics engine. The integrator object gives flexibility over the physics calculations to users. User has control over using different integrators for each object or creating and adding one. DMPhysics supports Euler, Verlet, Runge-Kutta 4 and Runge-Kutta 8 integration techniques.

3.3.4 Physics base object

Physics object is the object abstraction layer which constructs a base for other physics objects. Particle object, rigid-body object, and soft-body objects are derived from this object (Figure 30, Figure 31).

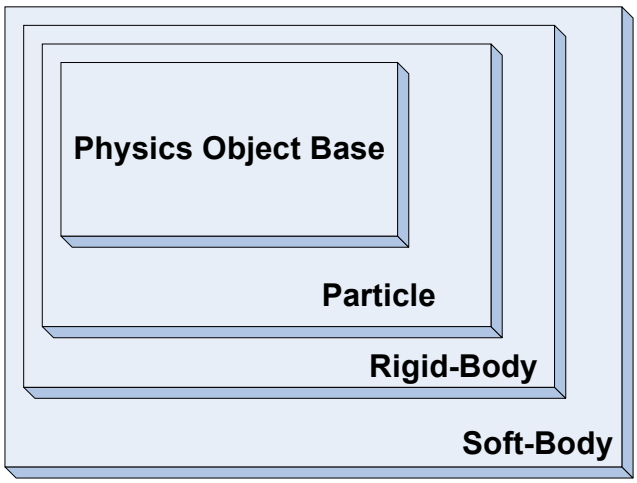


Figure 30 - Relationship between physics objects

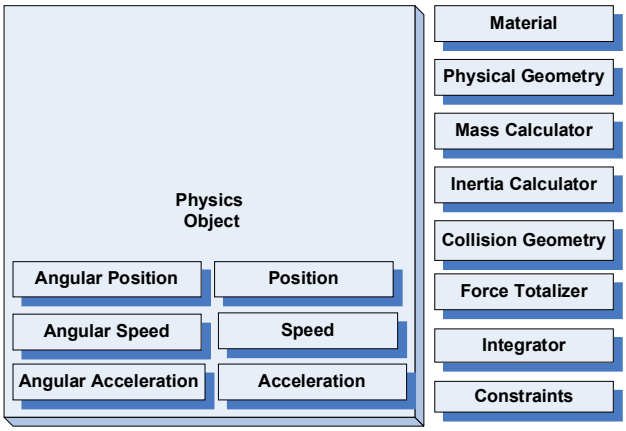


Figure 31 - Base Physics Object Container

3.3.5 Particle

Particle is the basic object in physics engine. It has neither shape nor volume. Particle only has mass which is defined in particle's material. Components of particle object are shown in Figure 32.

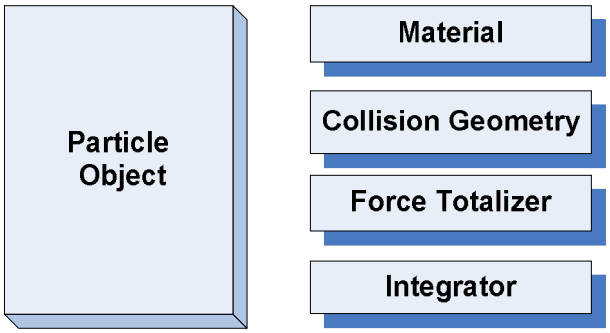


Figure 32 - Particle Object

On particle update, force totalizer sums all the forces affecting that particle and passes these forces to the integrator. The integrator first calculates the particle's acceleration using Equation 3.4.

Equation 4 - Newton

$$a = \frac{F}{m} \tag{3.4}$$

Then it integrates this acceleration in order to find final velocity. After finding the final velocity, integrator integrates final velocity and acceleration with the old velocity to find the final position. On particle update only force totalizer and the integrator are used (Figure 33).

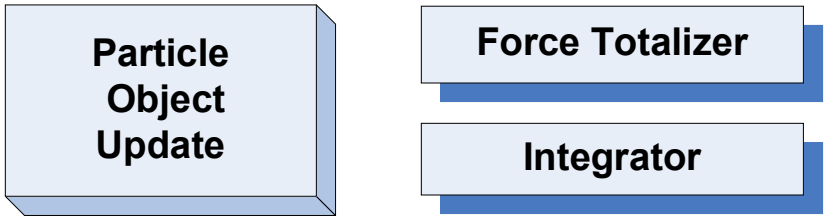


Figure 33 - On particle update

3.3.6 Rigid body object

Rigid-body is a solid body with a finite size and mass in which deformations are neglected. Material has object's density function, kinetic, static friction coefficients. In load time, firstly, mass calculator calculates objects mass using the density function with Equation 3.5. Then it calculates the center of mass with Equation 3.6.

Equation 5 - Mass calculation of rigid body

$$m = \iiint \rho(x, y, z) d_x d_y d_z \tag{3.5}$$

Equation 6 - Center of mass calculation of rigid body

$$m_{cm} = \frac{1}{m} \sum_{i=1}^{n} m_i p_i \tag{3.6}$$

Secondly the inertia calculator calculates the inertias for x, y and z axis respectively by using Equation 3.7.

Equation 7 - Inertia calculation

$$I_a = \sum_{i=1}^{3} m_i a_{pi} b_{pi} \tag{3.7}$$

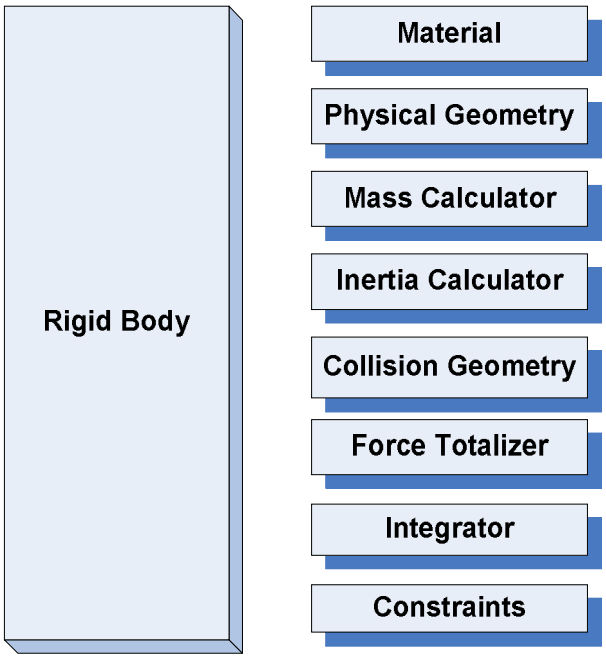
Then it calculates inertias for XY, XZ, YZ planes by using the same equation.

Then it forms the inertia matrix (Tensor) as shown in Equation 3.8:

Equation 8 - Inertia Tensor

$$I = \begin{bmatrix} I_x & -I_{xy} & -I_{xz} \\ -I_{xy} & I_y & -I_{yz} \\ -I_{xz} & -I_{yz} & I_z \end{bmatrix} \tag{3.8}$$

Then it calculates the I^{-1} (inverse inertia matrix).



-Figure 34 - Rigid body object

Linear velocity and position are calculated by finding the acceleration as done in particle equations. Angular velocity and rotation are calculated with the Equation 3.9

Equation 9 - Torque

$$\tau = P_f x F \tag{3.9}$$

After finding the final position, world transformer transforms object, constraint attach points and the bounding volume into world coordinates. Figure 35 shows components running on update state.

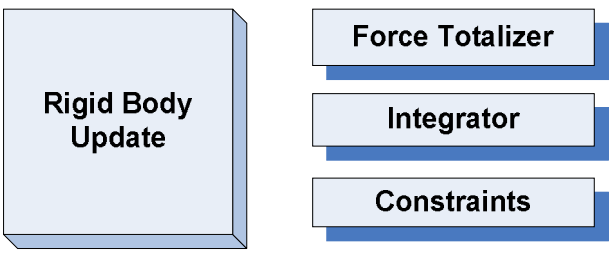


Figure 35 - Rigid body update

3.3.7 Maxwell-mesh

Maxwell mesh is used to represent the nonlinear behavior of the tissue model. Maxwell element can be represented with a series connected damper and spring. Thus Maxwell model represents a second order differential equation that depends on the position with spring equation, position's derivative with respect to time – velocity- with damper and position's second derivative with respect to time – acceleration with the mass.

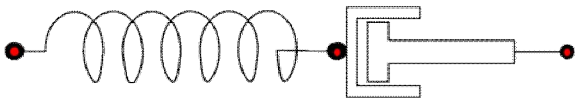


Figure-36 - Maxwell element

3.3.8 Soft body

The soft-body encapsulates rigid-body and particle objects and incorporates geometric deformations. Geometric deformations are modeled with Maxwell elements attached to particles that represent a small volume of the element. Joining these elements in a mesh structure is called Maxwell mesh. Tension forces are simulated with structural meshes as seen on Figure 37 (left), shear forces are simulated with shear mesh as seen of Figure 37 (right) and bend forces are simulated with bend mesh as seen on Figure 38 (left). Figure 38 (right) shows Maxwell mesh in 3D.

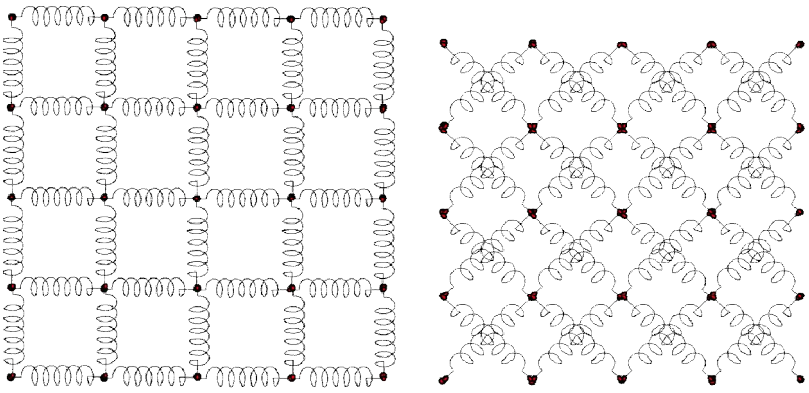


Figure 37 - Structural (left) and shear (right) Maxwell mesh

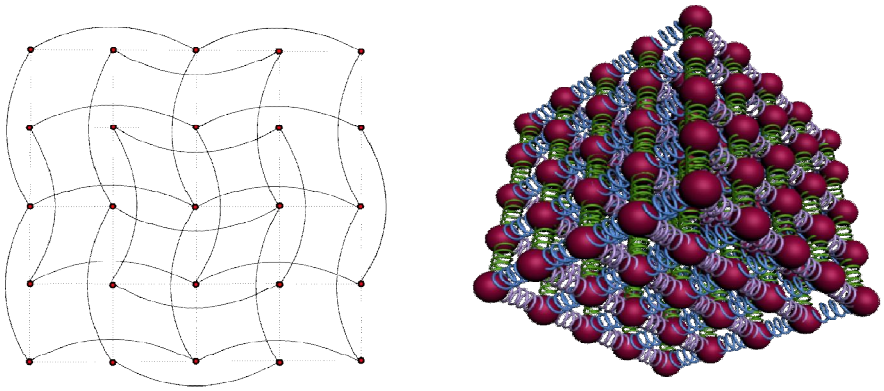


Figure 38 - Bend Maxwell mesh (left) and Maxwell mesh in 3D (right)

3.3.9 Grids

Grids are used to represent a small volume or area of physical objects. For example in fluid simulation each grid cell holds parameters like fluid density, fluid pressure, fluid temperature for that volume delta, in electromagnetic simulation each grid cell holds parameters like electric field vector and for soft volumes each grid cell represents the volume of the soft-body objects. Figure 39 shows an example of a volume, modeled with Maxwell meshes.

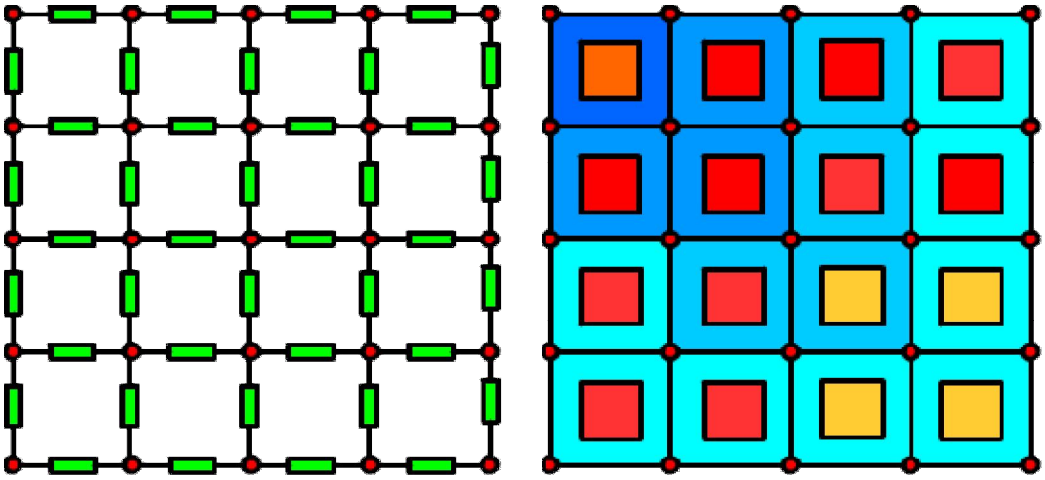


Figure 39 - Example grids Maxwell grid (left) fluid grid (right)

3.3.10 Hierarchical grids

Grids hold a huge amount of data thus processing all these data is sometimes impossible. Hierarchical grids allow us to simulate the interested region with the same amount of data and the distant regions with fewer amounts of data**.** As seen in the Figure 40, interested region is modeled with normal mesh size. As going further the Maxwell mesh amount decreases exponentially for a fixed size volume.

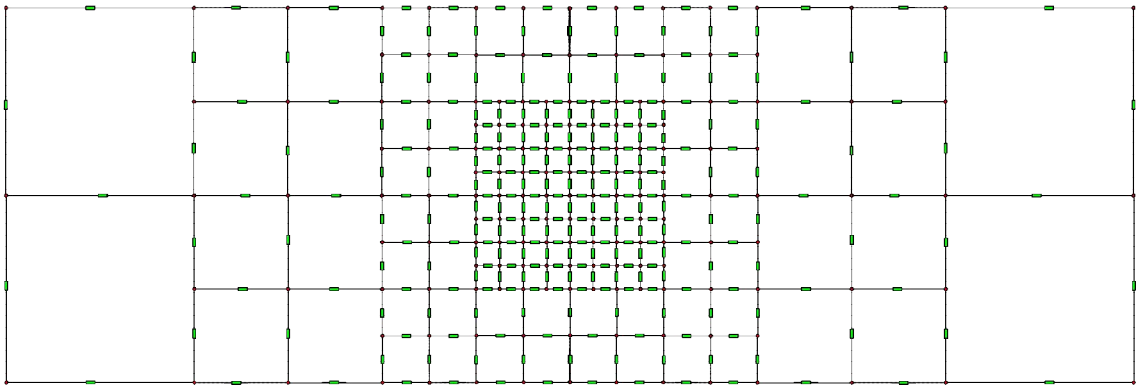


Figure 40 - Hierarchical Maxwell grid

3.4 DMCollide – Collision Detection Engine

Collision detection engine is responsible for detecting collisions between objects. The collision detection engine has three steps. First one is the coarse collision detector, which uses simple and fast collision detection algorithms and passes possible collisions to the narrow-phase collision detector. Second stage is the narrow-phase collision detector which uses detailed collision tests that is more computationally expensive. And third one, in the last stage collision resolver solves when and where the collision is happened and reports intersection data to the physics engine where the collision response solver is.

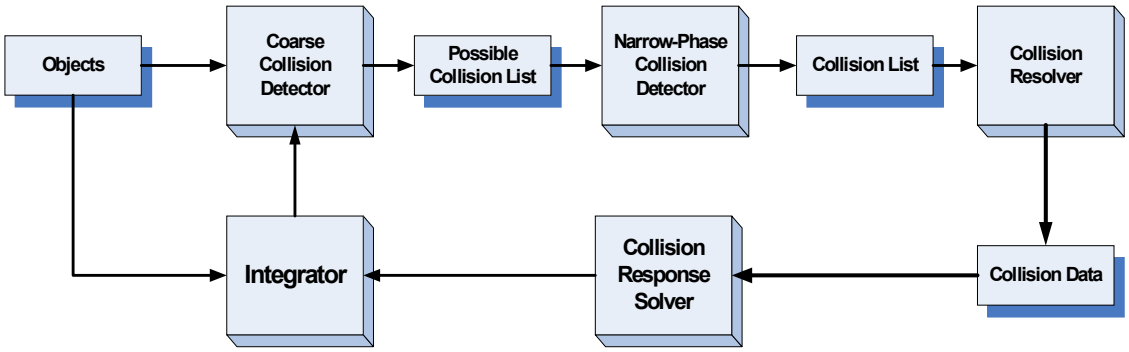


Figure 41 - Collision detection loop

3.4.1 Fundamental collision tests

Collision detector deals with meshes, combined by vertices, are samples of the model. Three vertices make triangle which has 3 edges and 3 corners. Figure 42 shows an example of two triangles close to each other.

Thus three fundamental collision objects are triangle itself, corners and edges. This three objects leads to six collision tests per triangle which are:

- Corner-Corner Intersection
- Corner-Edge Intersection
- Corner-Triangle Intersection

- Edge-Edge Intersection
- Edge-Triangle Intersection
- Triangle-Triangle Intersection

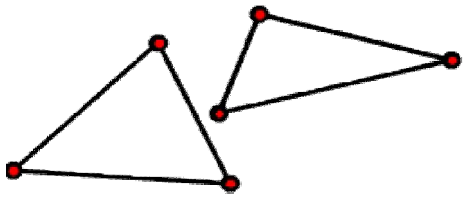


Figure 42 - Collision test on two triangles

In two 3 billion triangle meshes there would be $9.0x10^{18}$ triangle tests and each test should be performed with six tests as shown above. This leads to $54x10^{18}$ collision tests which is impossible to process in real-time for today's computers. Therefore, these triangles will be packed into bounding volumes in order to decrease computation time in the next part.

3.4.2 Bounding volumes

In order to reduce the complexity of mesh intersection tests, bounding volumes which are easier and more computationally cheaper are used. Figure 43 shows some example bounding volumes.

The primitive bounding volumes and primitive collision geometries are listed below:

- Point
- Line
- Ray
- Segment
- Plane
- Triangle
- Sphere
- Axis Aligned Bounding Box
- Object Oriented Bounding Box

- Capsule
- Cylinder
- Compound
- Convex Hull

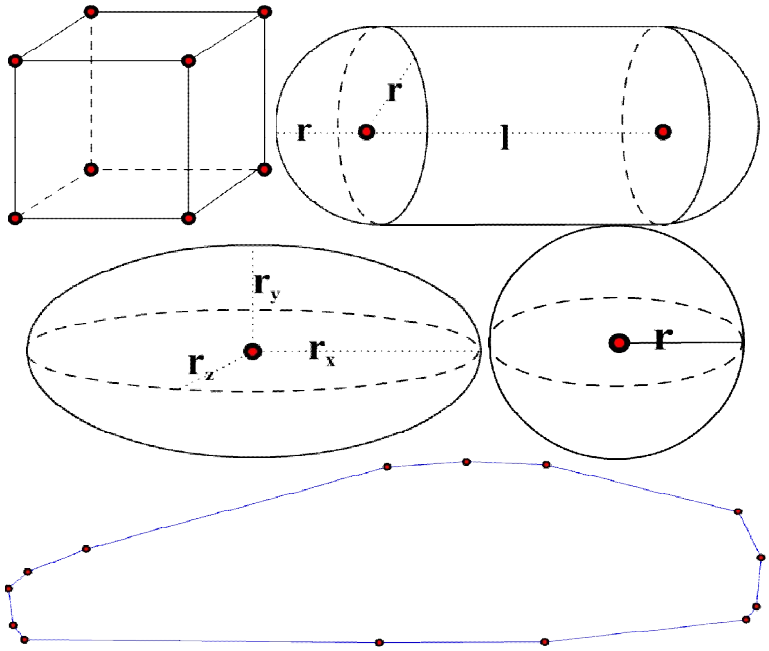


Figure 43 - Some examples of bounding volumes

3.4.3 Advantages & disadvantages of the bounding volumes

Some volumes perfectly fit into the mesh but some are not. Thus we need to use best fitting bounding volume in order to represent our mesh but we have to consider the computational complexity of the volume. The sphere is one of the cheapest bounding volume shown in Figure 44. We just compare the distance of the sphere centers with the summation of the sphere radiuses. But sometimes –at most cases- mesh does not fit into the sphere volume perfectly as seen on the Figure 44. This time selecting a better fit like the AABB (Axis aligned Bounding Box) will be a better solution (Figure 45). But when the object transformation is changed the box should be updated to fit the new orientation. But as seen on Figure 45 the fitting is distorted again. Thus selecting OBB – Object Oriented

Bounding Box- is more suitable (Figure 46), however this time all the vertices of the box should be transformed with the object. Selecting another volume such as ellipsoids can be more suitable (Figure 47). Or the object can be divided into compounds with combination of the bounding volumes. If better collision detection resolution is needed the convex hull could be selected which is the most computationally expensive volume before the mesh. These volumes could be arranged into a hierarchy. Next part explains how to calculate the convex hull and after that how to create bounding volume hierarchies.

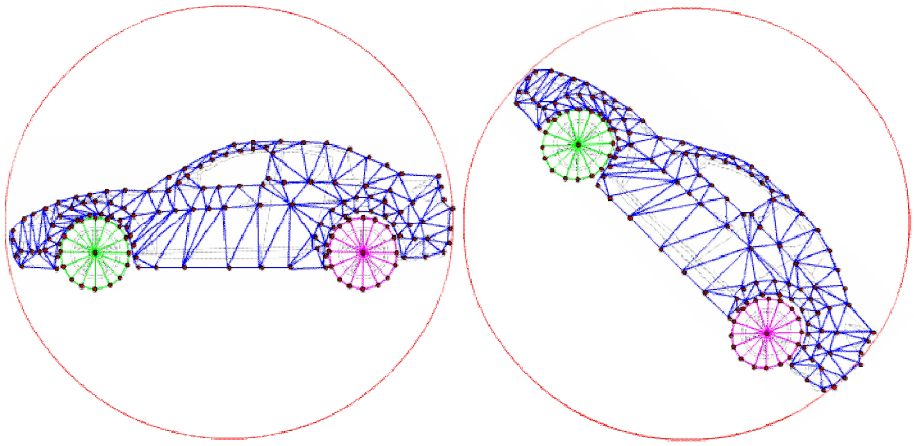


Figure 44 - Sphere: Initial state (left) and rotated (right)

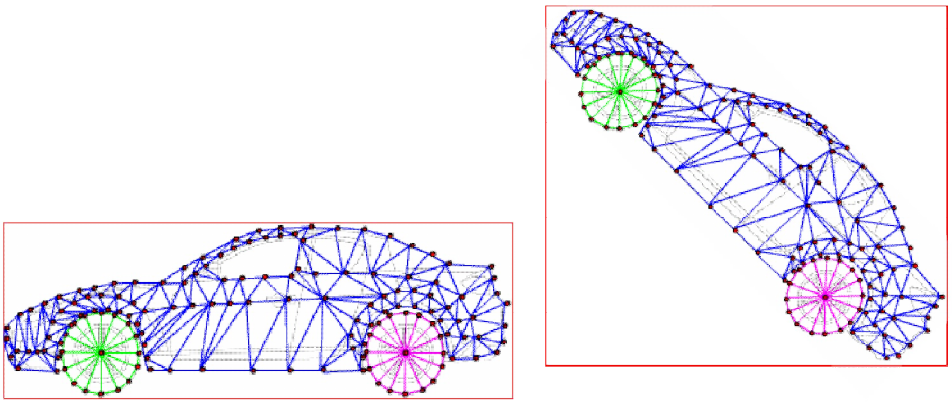


Figure 45 - AABB : Initial state (left) and rotated (right)

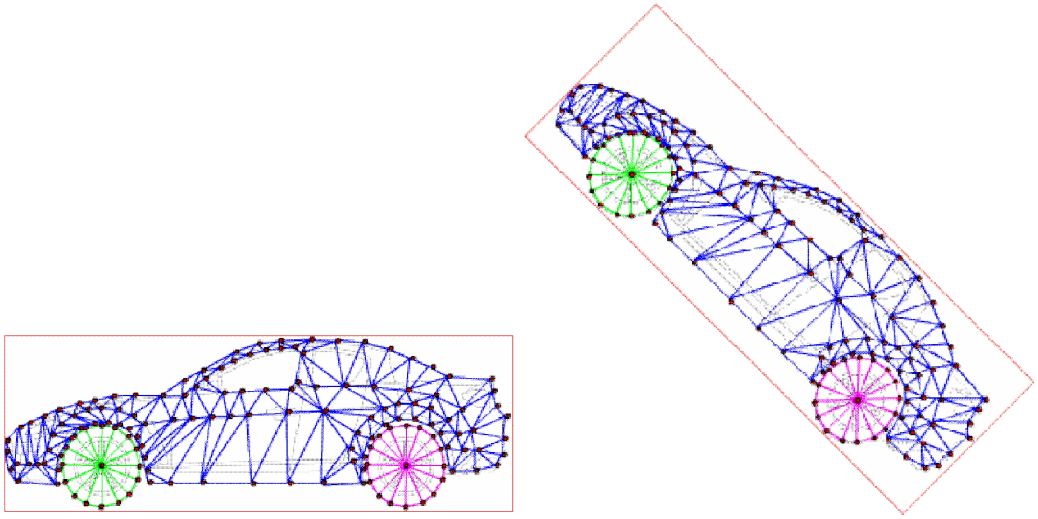


Figure 46 - OBB : Initial state (left) and rotated (right)

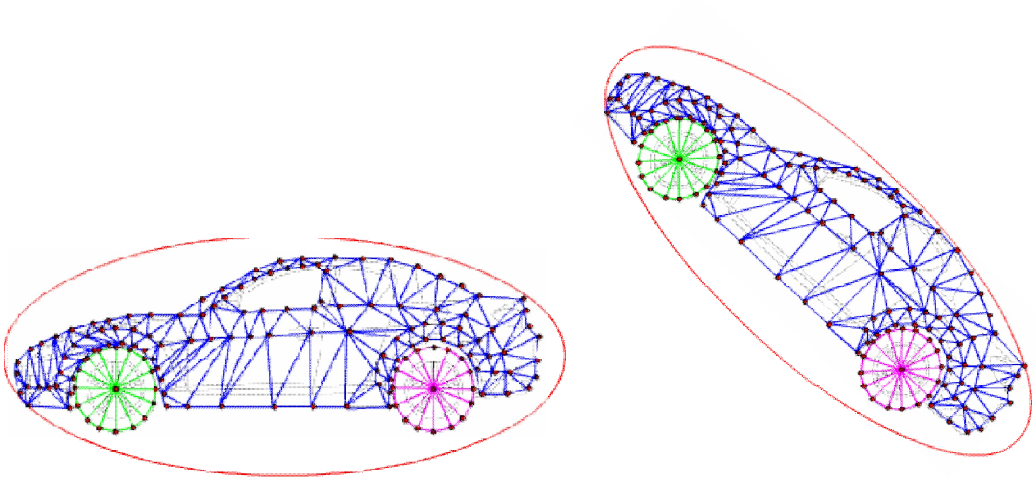


Figure 47 - Ellipsoid, Initial state (left) and rotated (right)

3.4.4 Compound

Compounds are used to represent the mesh with a combination of bounding volume sets. Instead of encapsulating the whole mesh with a single volume, the mesh can be represented by dividing mesh into sub sets and representing each sub sets with the best fitting volume. Figure 48 shows an example of this technique and Figure 49 presents the produced bounding volume tree.

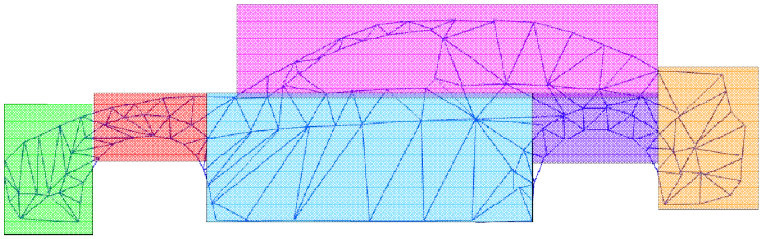


Figure 48 - Compound generated from Axis Aligned Bounding Boxes

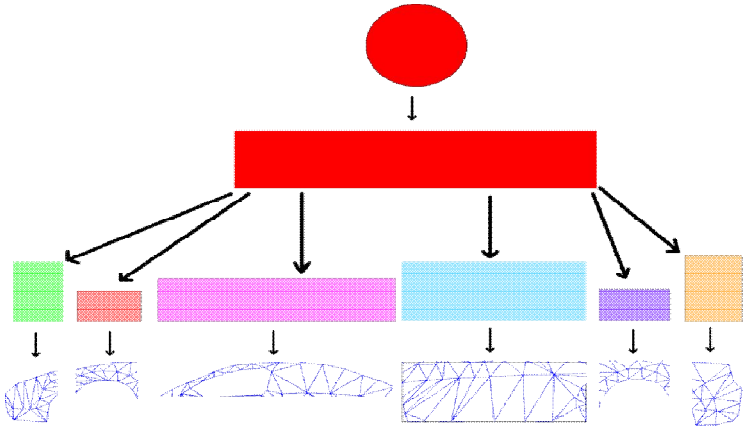


Figure 49 - Collision tree of the compound below

3.4.5 Convex hull

Convex Hull is a mesh encapsulation method. It allows representing the mesh with a simplified version of the same mesh that contains small amount of vertices. Accordingly collision detection computations become more precise. But as its name points out it just encapsulate the convex meshes, concave parts are ignored and error-prone.

Building the convex hull is done by the quick hull algorithm shown in Figures 50 to 62. Firstly the algorithm starts finding the minimum and maximum point vertices for each coordinate axes. Then connecting these vertices produces the first hull and vertices inside the hull are deleted. Secondly the maximum distance vertices to the found planes are found and connected to the hull and vertices inside this hull are deleted. This second stage continues in a loop until there are no vertices outside the hull. And finally our convex hull is generated as seen from the Figure 62.

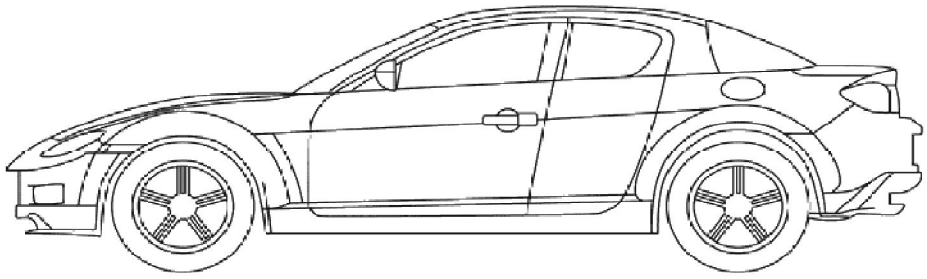


Figure 50 - Original Model

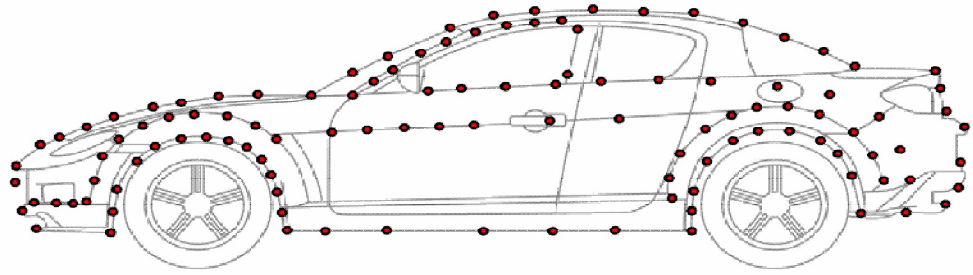


Figure 51 - Sampled vertices

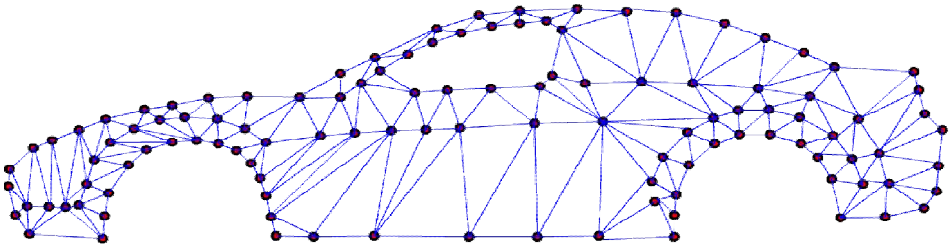


Figure 52 - Generated mesh from vertices

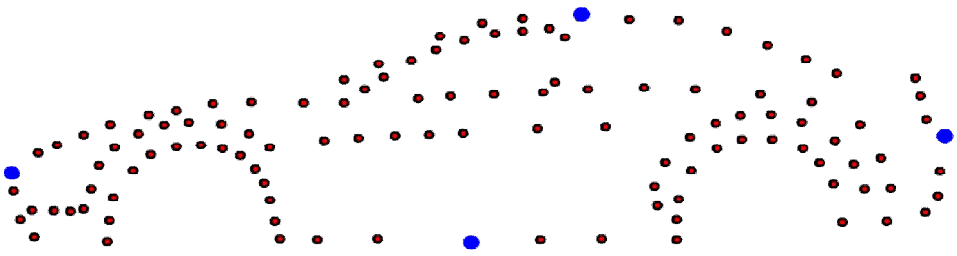


Figure 53 - Maximum points found in blue

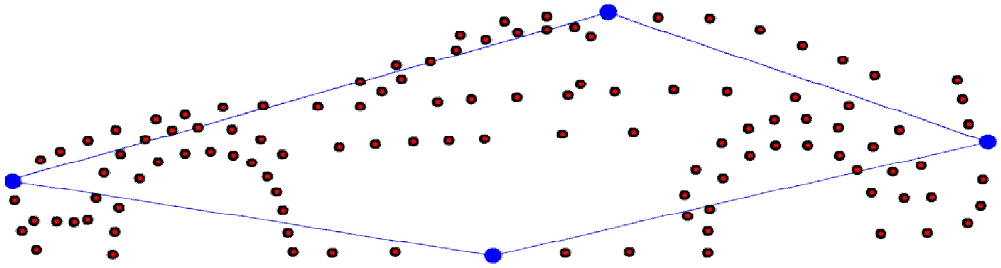


Figure 54 - Connected to each other

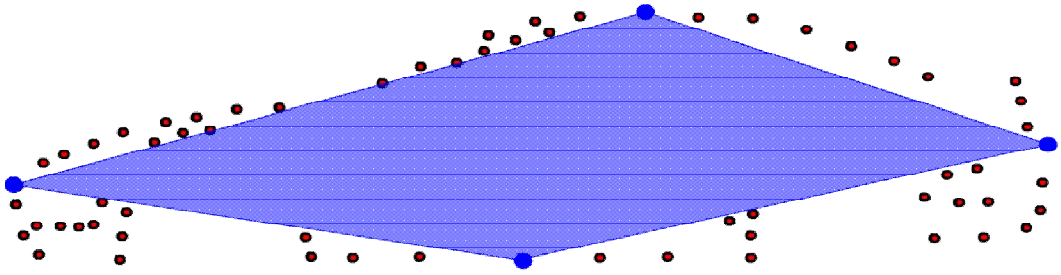


Figure 55 - Deleted vertices inside this volume

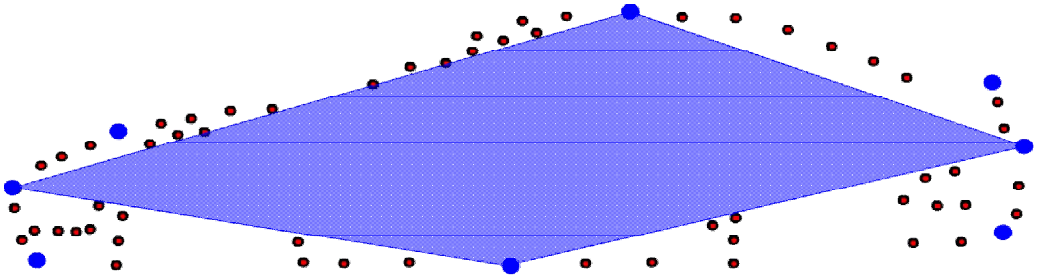


Figure 56 - Maximum distanced vertices from planes found in blue

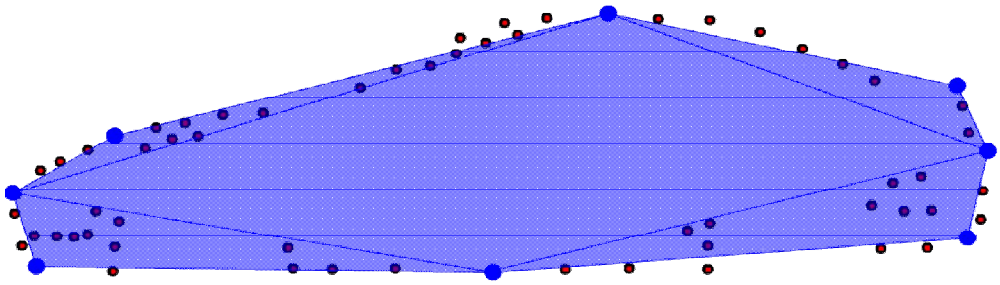


Figure 57 - These vertices connected with hull and vertices inside are cleared

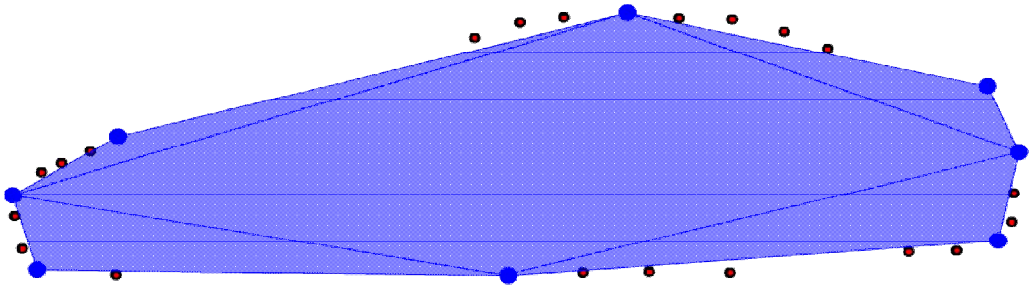


Figure 58 - Algorithm continues with other planes

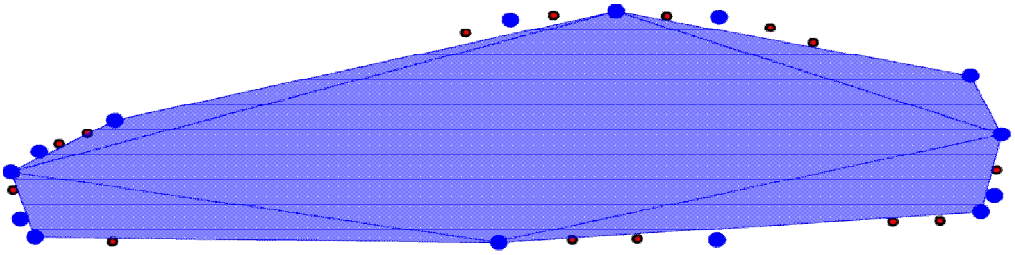


Figure 59 - Maximum points found again

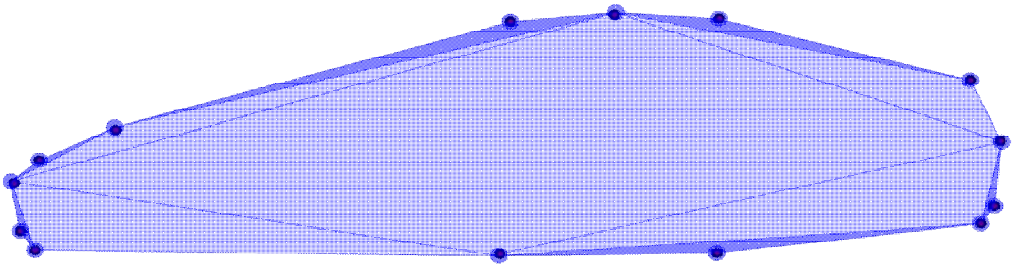


Figure 60 - Maximum points added to the hull and inside the hull is cleared

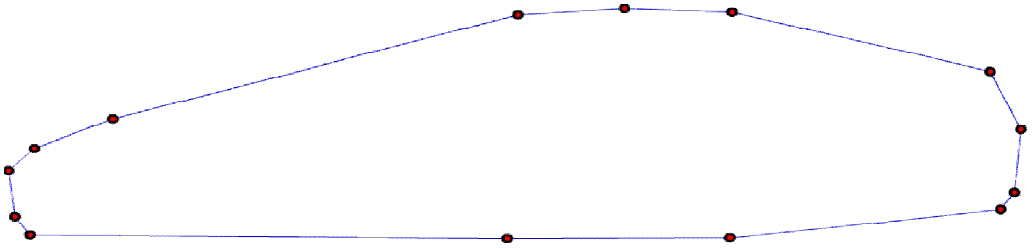


Figure 61 - There is no vertices left the algorithm finished

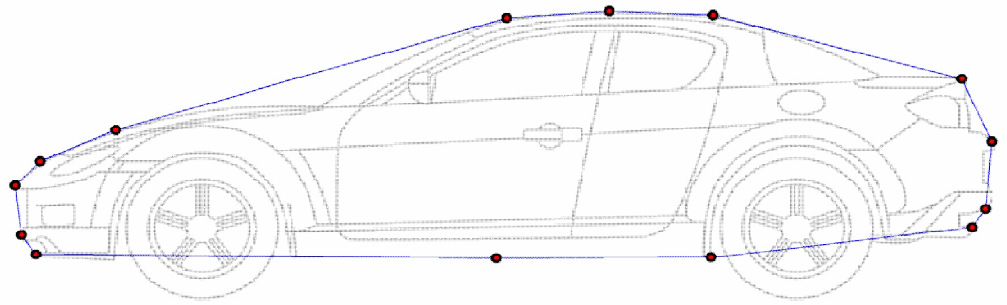


Figure 62 - Convex parts are covered successfully but concave parts are not

3.4.5 Bounding volume hierarchies

Hierarchies are used for placing and sorting the bounding volumes from cheaper to expensive computation requirements. The idea is that with using a cheap collision detection volume we can detect if a collision is possible. If not, we just continue but if possible then we transform the complex bounding volumes and make complex tests. In short if there is no possibility of collision we don't need to make expensive calculations. Figure 63 shows a hierarchy of a table and Figure 64 shows the hierarchy of a complex car mesh.

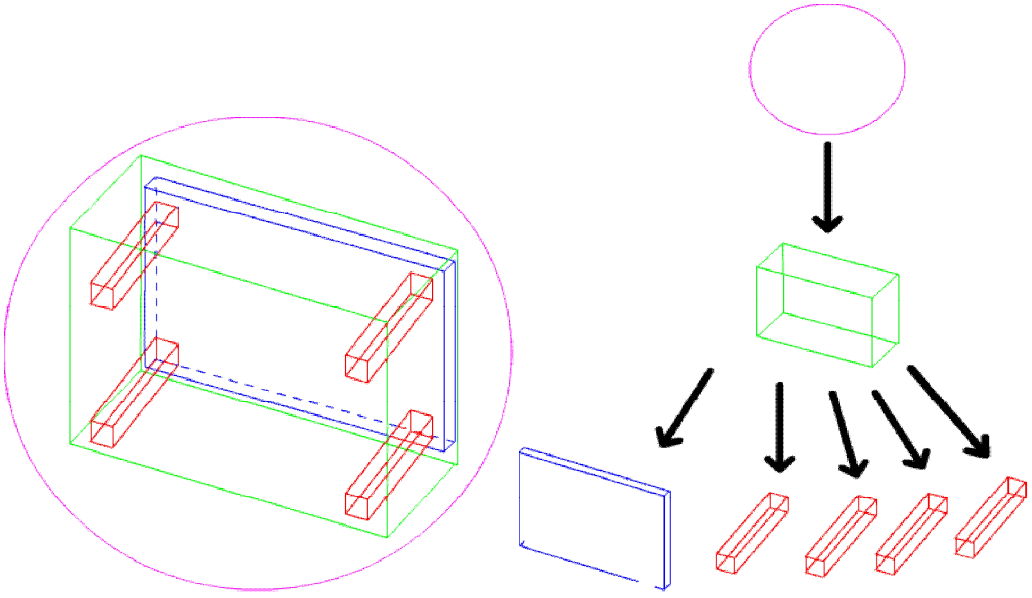


Figure 63 - Example Hierarchy of a desk mesh

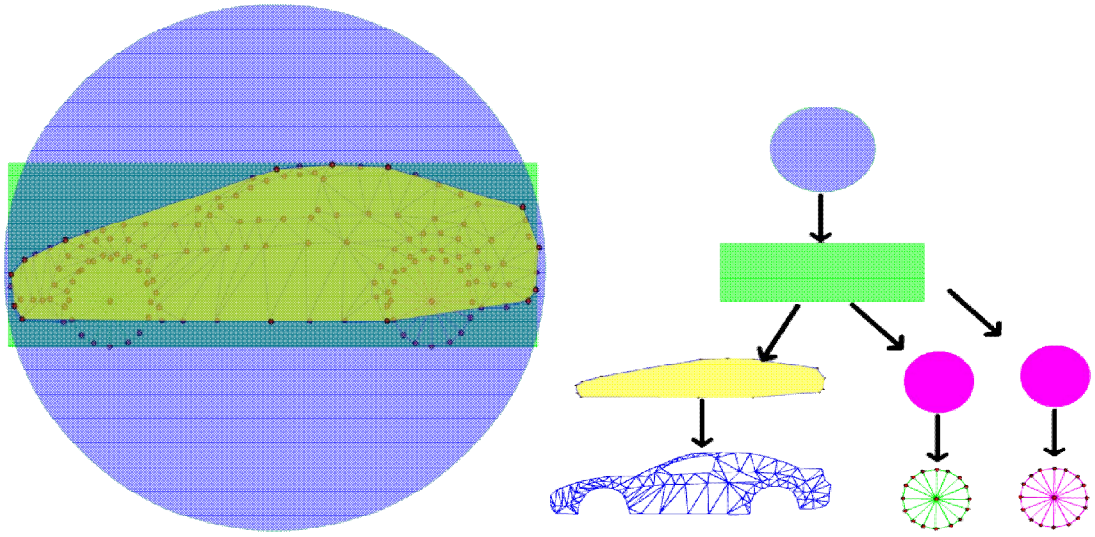


Figure 64 - Example hierarchy of a car mesh

3.4.6 Oc-tree mesh hierarchy

Storing the mesh in an oc-tree bounding volume hierarchy is a divide and conquer technique. Dividing the mesh with an oc-tree decreases the number of intersection tests. Thus this speeds up the collision detection process. Figure 65 to 68 show constructing a oc-tree mesh hierarchy.

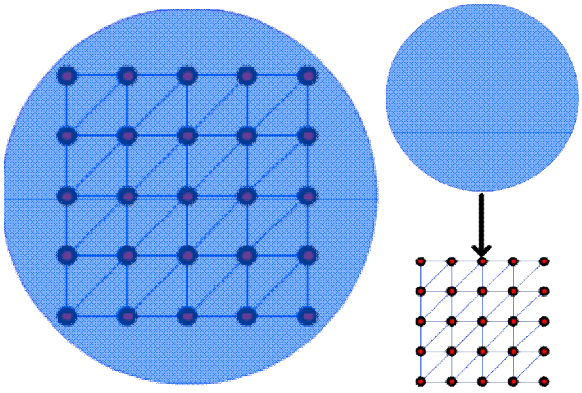


Figure 65 - Mesh is placed into bounding sphere

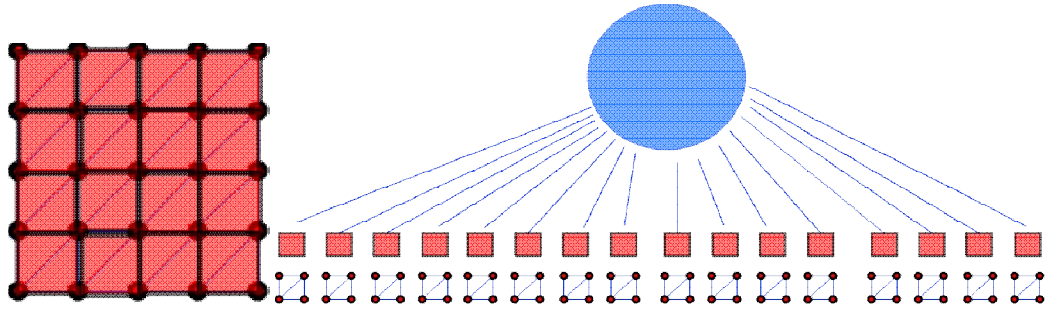


Figure 66 - Each 4 vertices placed into a AABB

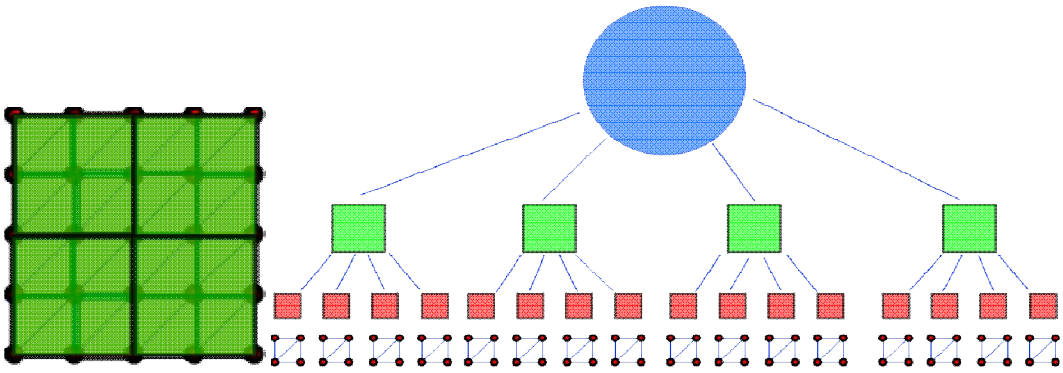


Figure 67 - Each 4 AABBs placed into an AABB

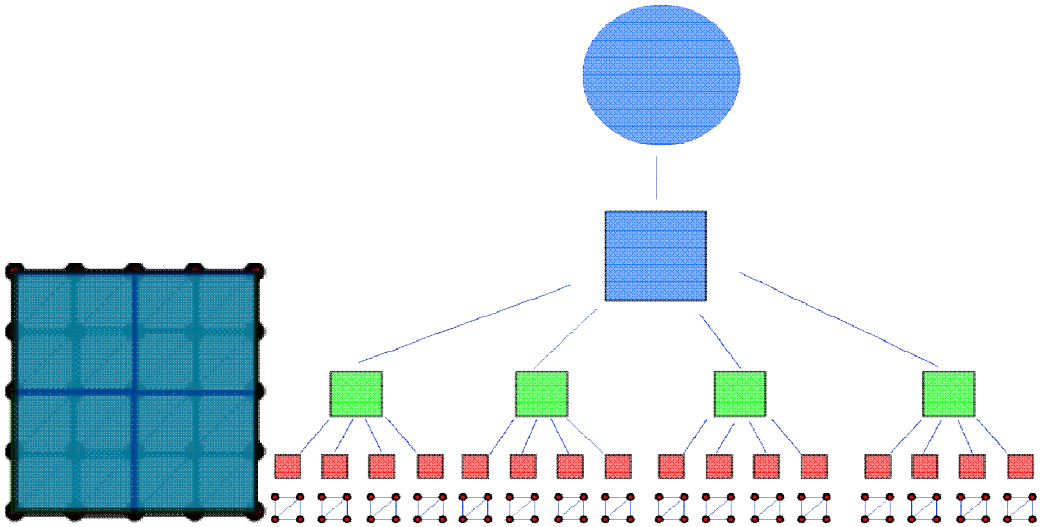


Figure 68 - Finally last 4 AABBs placed into one AABB

In order to demonstrate the performance of the oc-tree mesh hierarchy method an example is given below.

Example Test:

Suppose that an object is tested with oc-tree hierarchy (Figure 69). Normally there would be 64 vertices tests. But if the oc-tree hierarchy is used shown in the Figure 70, the test amount would decrease into 13. As seen from the Table 6 and Table 7, as the vertex count increases the speed-up increases with a power of two in 2D meshes with a power of 3 in 3D meshes. Testing of a 512*512*512 mesh with the oc-tree hierarchy delivers a speedup of 1.8 million times and in 1024*1024*1024 mesh 13 million times. Using oc-tree with the other mesh will double this speed up to 169 000 000 000 000 times.

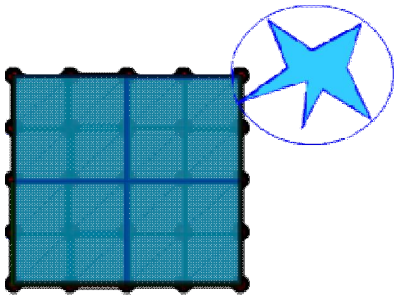


Figure 69 - Example collision scenario

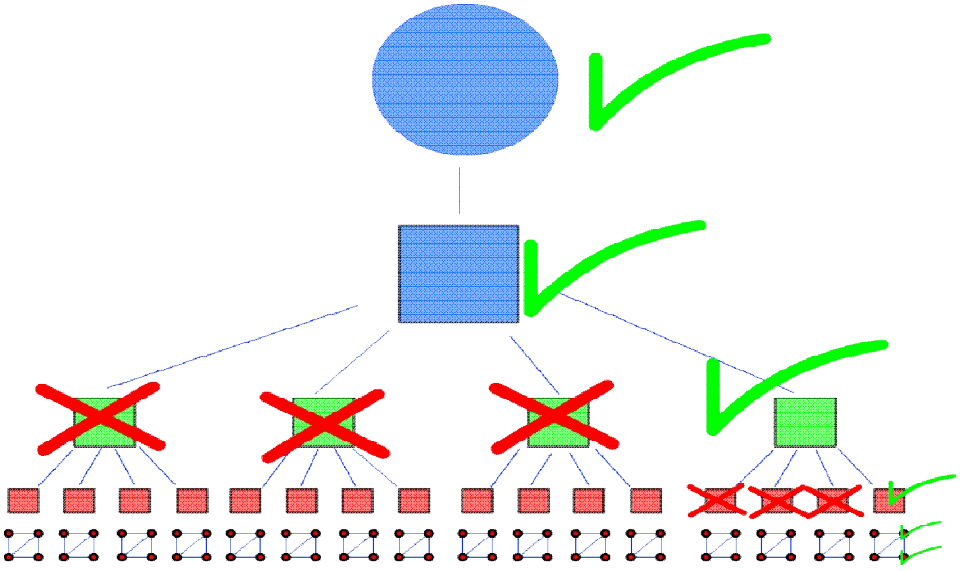


Figure 70 - Example collision test with oc-tree bounding volume hierarchy

Table 6 - Oc-tree mesh hierarchy test table in 2D

Grid Size	Number of Elements	Iterations Required	Iterations With Tree		Speedup
1x1	1	1	1		1.00000
2x2	4	4	5		0.80000
4x4	16	16	9		1.77778
8x8	64	64	13		4.92308
16x16	256	256	17		15.05882
32x32	1024	1024	21		48.76190
64x64	4096	4096	25		163.84000
128x128	16384	16384	29		564.96552
256x256	65536	65536	33		1985.93939
512x512	262144	262144	37		7084.972973
1024x1024	1048576	1048576	41		25575.02439

Table 7 - Oc-tree mesh hierarchy test table in 3D

Grid Size	Number of Elements	Iterations Required	Iterations With Tree		Speedup
1x1x1	1	1	1		1.00000
2x2x2	8	8	9		0.88889
4x4x4	64	64	17		3.76471
8x8x8	512	512	25		20.48000
16x16x16	4096	4096	33		124.12121
32x32x32	32768	32768	41		799.21951
64x64x64	262144	262144	49		5349.87755
128x128x128	2097152	2097152	57		36792.14035
256x256x256	16777216	16777216	65		258111.01538
512x512x512	134217728	134217728	73		1838599.01370
1024x1024x1024	1073741824	1073741824	81		13256071.90123

3.4.7 Coarse collision detector

The aim of the coarse collision detector is to find possible collision pairs as fastest as possible. Thus at this point the impossible collisions are eliminated. Coarse collision detector uses spheres to test collisions. The sphere-sphere collision test is a cheap collision test that only requires subtracting centers of the two spheres from each other and testing if the distance is smaller than total radius of the two spheres. Spheres are beneficial in transformation. We do not need to apply rotations to the spheres because of their geometry, spheres are independent on rotations. For collision detection we have to check each object in the scene with other objects but this is an expensive work. In order to overcome this problem, a fixed sized collision grid is created and objects are placed into this grid. Depending on the location and the size, objects can be placed in multiple grids. As seen on Figure 71 there are 28 objects in the scene and testing each other with using the standard algorithm (Figure 72) makes 378 collision tests (figure 73). Placing the collision grid on the scene, separates the test regions as seen on Figure 74-75. Testing just the objects inside the grid cell increases the test speed. In this example, we decrease the test count to 23 which is 16 times faster than the standard test algorithm.

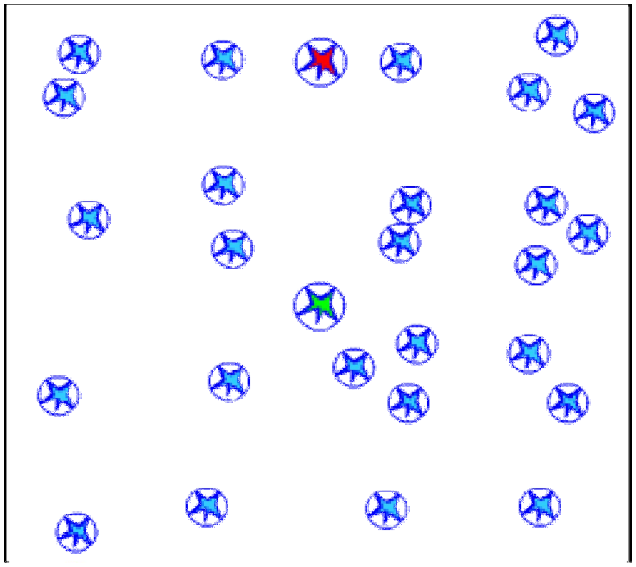


Figure 71 – Test objects in the scene

```
for(i=0; i<N; i++)       /* N   Objects          */
   for(j=i+1; j<N; j++) /* N-1 Objects to test  */
      {
         /* Test if they collide     */
         if( TestIntersection(Object[i],Object[j]) )
            {
               /*   If colliding Add to the Possible Collision List */
               AddToPossibleCollisionList(Object[i],Object[j]);
            }
       }
```

Figure 72 - Each object should be tested with others in the scene

$$Total\ Intersection\ Tests = \sum_{i=1}^{N-1} i = \frac{N(N-1)}{2} = 378$$

Figure 73 - Total intersection count for 28 objects

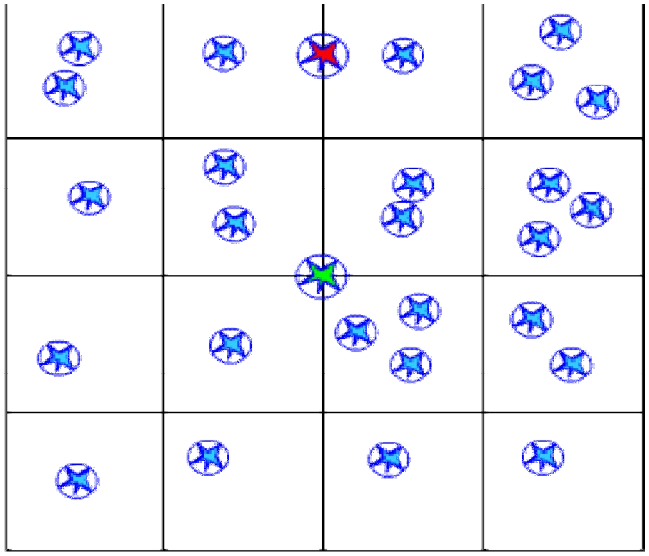


Figure 74 - Applying the grid to the scene

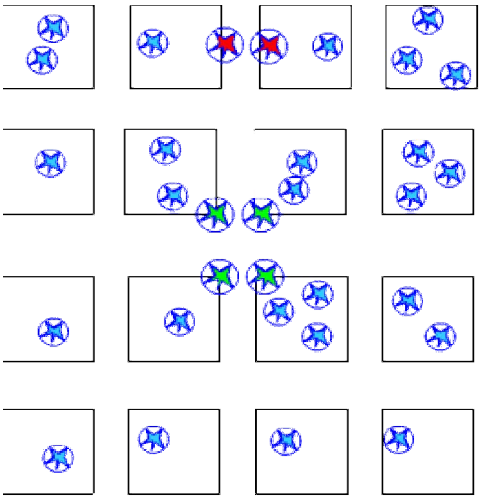


Figure 75 - Objects are arranged into corresponding grid cell

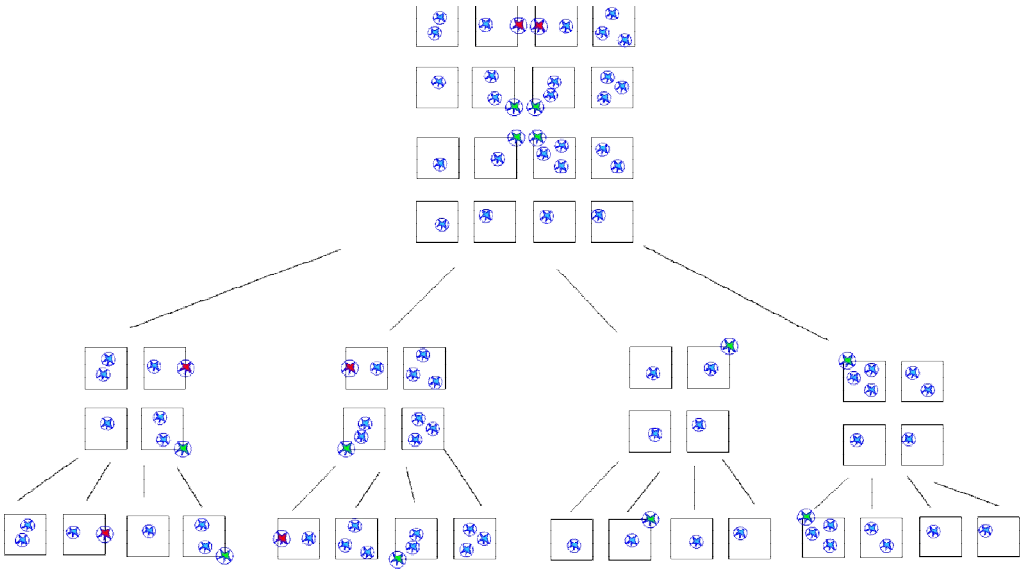


Figure 76 - Oc-tree hierarchy of the grid

3.4.8 Narrow-phase collision detector

Coarse collision detector is the pre-elimination step of the collision detection process. The Narrow-Phase Collision detector is the real collision detection system which uses complex collision detection volumes, hierarchies and mesh trees from cheapest to expensive. The coarse collision detector can pass false

collisions which is actually not a collision but the error of using the sphere as an improper volume. As seen from the Figure 77 the Coarse collision detector reports this collision, the spheres are overlapped but the objects inside the spheres are not. This is called false collisions. After narrow phase detector finds the true collisions it passes this collision lists to the Collision Resolver which calculates where and when the collision has happened.

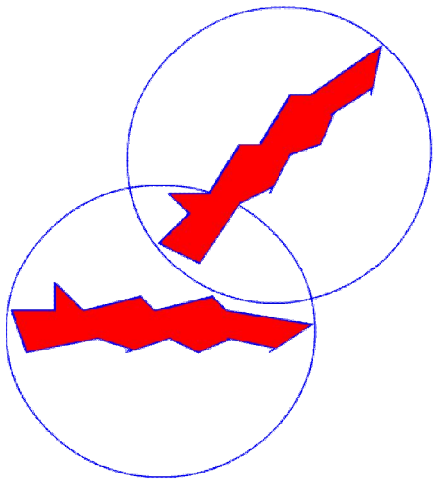


Figure 77 - Example of a false collision

3.4.9 Collision resolver

Collision resolver gets the true collisions and estimates where and when the collision happened using the object integrator in different time deltas. The algorithm is a binary search algorithm. As seen from the Figure 78 to 83, we know that there is a collision in delta-time but we don't know when it happened. Thus resolver first calculates the integrator between 0 to Delta/2 then tests if the collision is happened or not. If happened it changes the integrator limits from 0 to Delta / 4, if not from 0 to 3*Delta/4 then tests again. It performs this loop for a while. The exact solution can be calculated as increasing the loop count but the loop is limited into a small number like 3 in this study in order to decrease the complexity. Then the approximate contact time and contact point is detected.

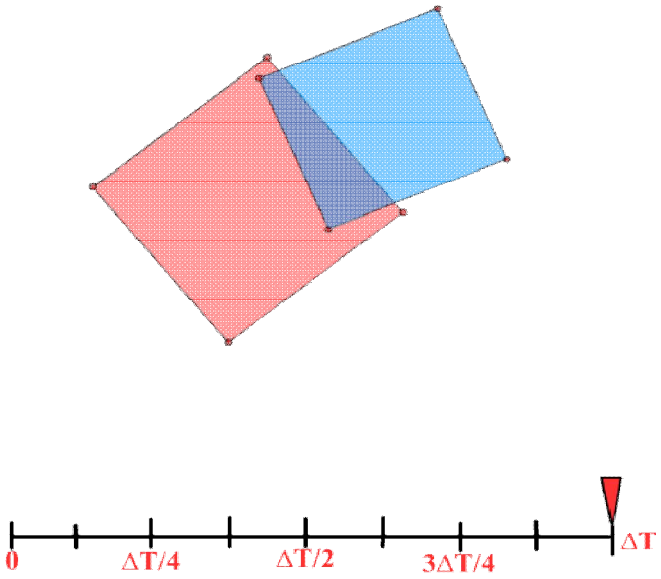


Figure 78 - Detected collision at delta t

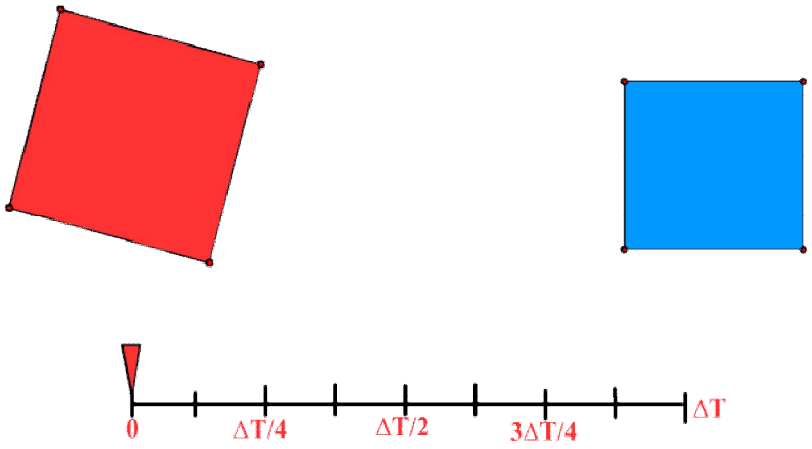


Figure 79 - Object positions at t=0

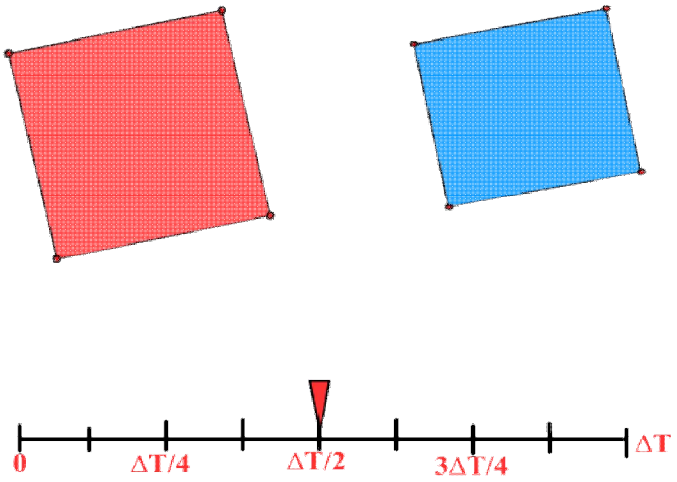


Figure 80 - Object positions at t = delta t / 2 ; There is no collision go forward in time.

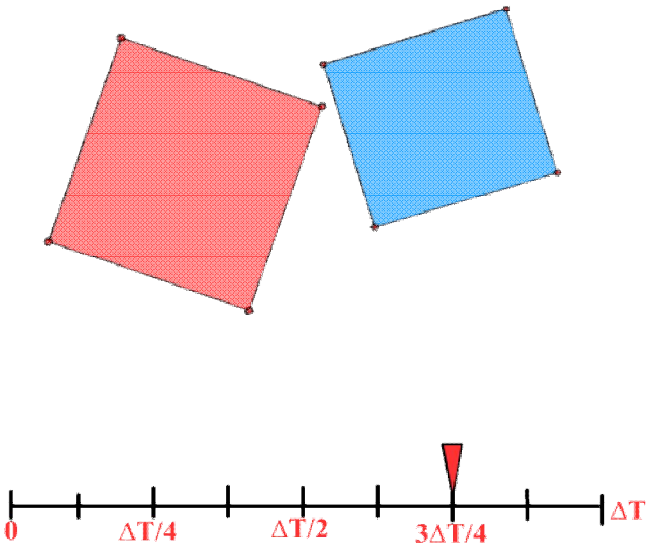


Figure 81 - Object positions at t = 3 delta T / 4; There is no collision go forward in time

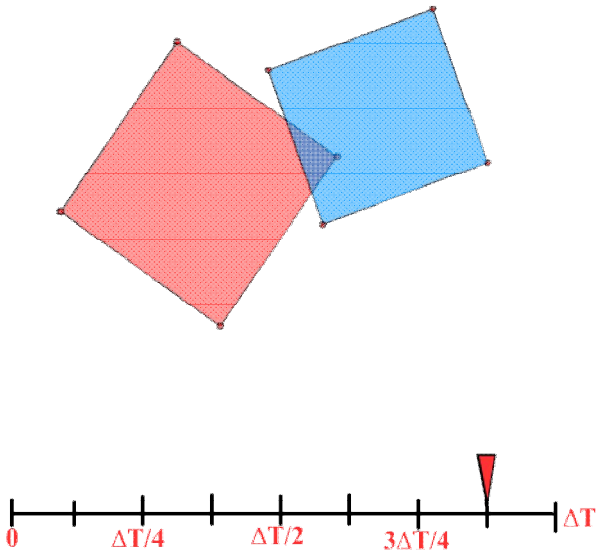


Figure 82 - Object Positions in 9 delta T/8; there is collision go back in time

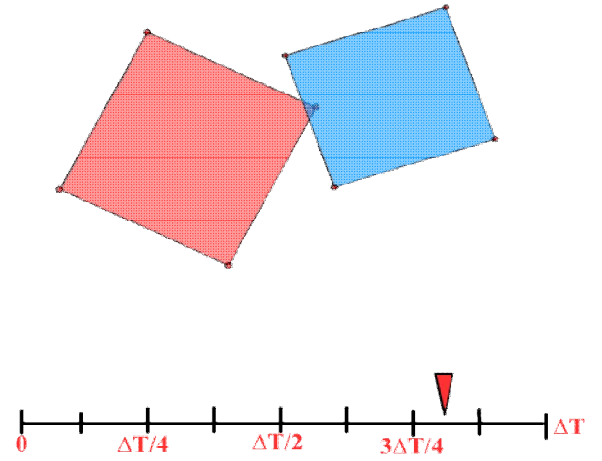


Figure 83 - Final collision positions found; Collision report will be created.

3.4.10 Collision Response Solver

Actually, collision response solver is in the physics engine not in the collision detection engine. Collision response solver solves the momentum equations and calculates the final velocities and positions of the objects by using elastic collision equations.

3.5 DMRenderer – Rendering Engine

DMRenderer is the Rendering Engine of the DMPlatform. It is responsible for calculating the lighting equations. The rendering a mesh operation can be simplified as shown in Figure 84. Firstly, transformation pipeline fetches each vertex of the mesh and translates them into the world space and applies the lighting equations. Secondly, each transformed vertices projects into the screen space coordinate system. Thirdly, each primitive projected into the screen space are rasterized by shading and texturing and evaluated in the Z-buffer. Finally, the output image is created and sent to the display system (Figure 85).

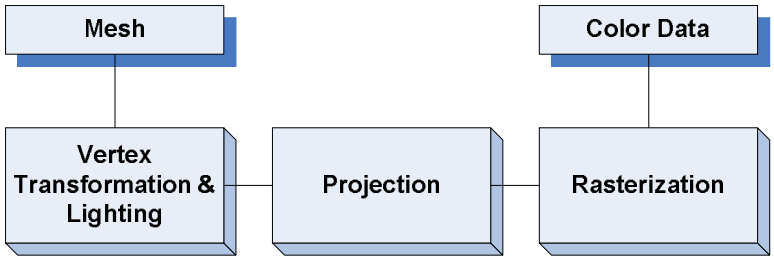


Figure 84 - Basic rendering pipeline

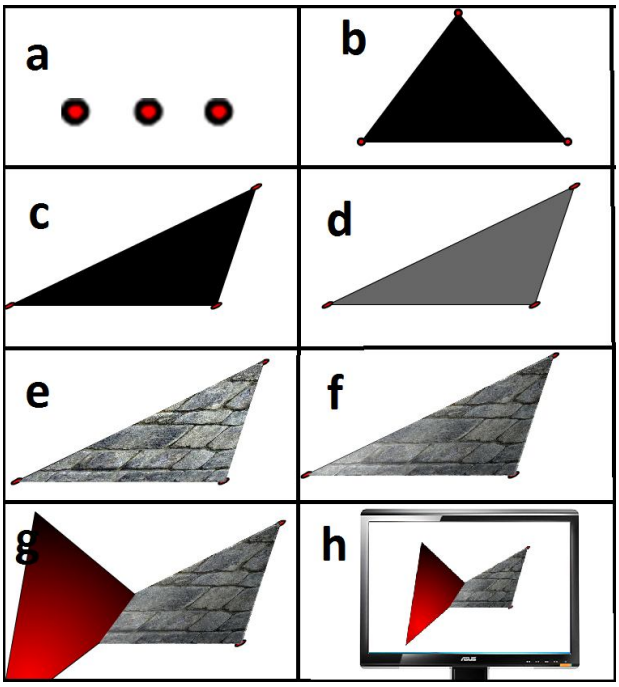


Figure 85 - Rendering Pipeline: a) vertices b) primitive generation c) transformation. d) vertex lighting. e) texturing. f) shading. g) Depth Test h) Display

3.5.1 Transformation

Objects are modeled in their local coordinates. They should be transformed to the actual world coordinates in order to find their real orientation by multiplying the vertices with the offset-translation, scaling, rotation in z, rotation in y, rotation in x and translation matrices. And then they need to be transformed to the camera space by multiplying the view matrix. Finally, they need to be projected to the screen space coordinate system by multiplying with the projection matrix. Transformation is illustrated by the Figures 86-90.

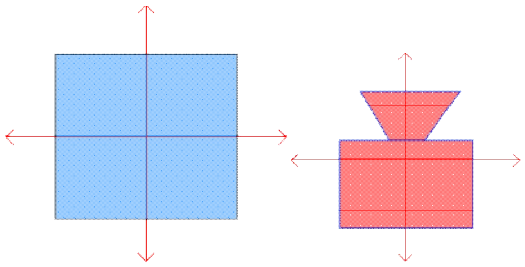


Figure 86 - Object and camera in local coordinate system.

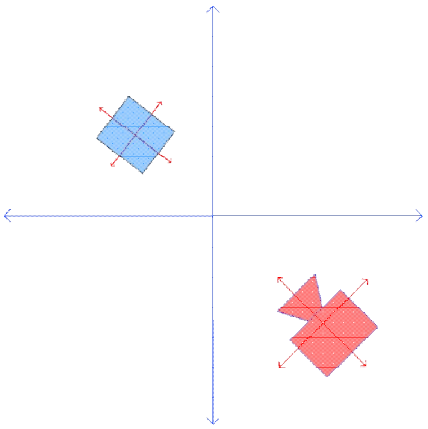


Figure 87 - Objects are transformed to the world coordinate with transformation matrix

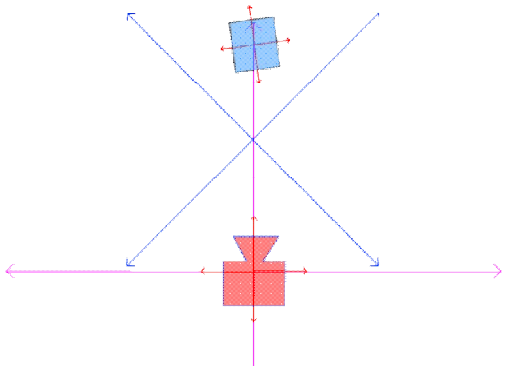


Figure 88 - Objects are transformed to camera space with view matrix

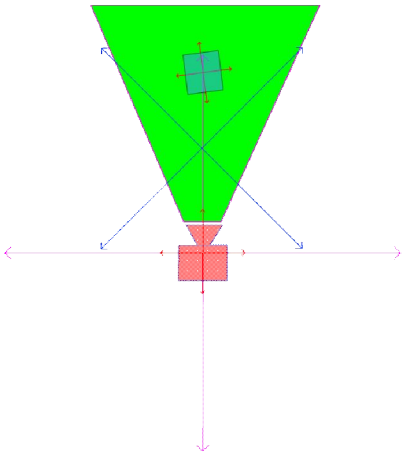


Figure 89 - The visible volume is highlighted by green (viewing frustrum)

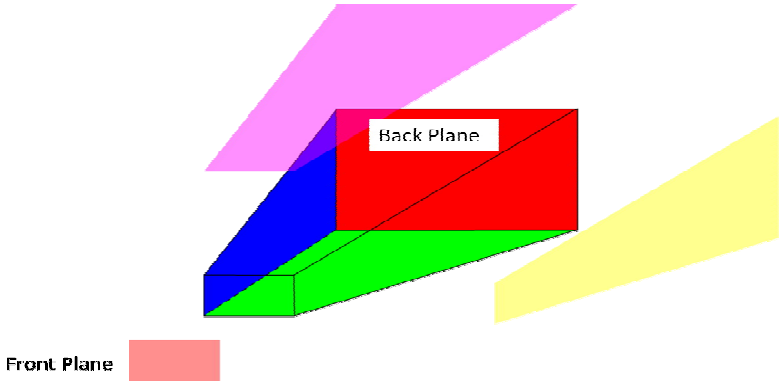


Figure 90 - 6 planes of view frustrum (Front Plane is the Screen). The objects are projected to the front plane with projection matrix.

3.5.2 Lighting

Objects are shaded in line of with the fall of light on them. Light amount arriving to the object and the amount of light transferred to the viewer should be calculated. Figure 91 (left), shows the simple model of lighting arriving to the object and reflected to the viewer. Figure 91(right) shows the global lighting model which accounts the inter object light reflections.

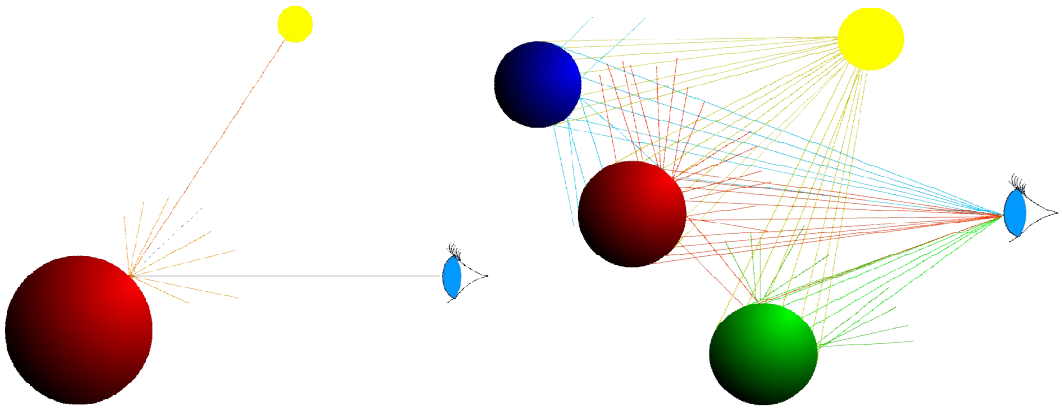


Figure 91 - Normal lighting (left) and global lighting model (right)

3.5.3 GPU architecture

The GPUs (Graphics Processing Units) are designed for performing calculations below. The transformation pipeline is responsible for applying per vertex transformation equations. These are multiplying vectors with offset, scaling, translation, rotation in x axis, rotation in y axis, rotation in z axis, view and projection matrices. This requires a powerful processing power and hardware matrix multiplication capability. Shading pipeline is responsible for texturing which is a look-up table operation and applying per pixel lighting equations which uses simple dot product equations. Thus the vertex processing needs more processing power and more hardware requirements which occupies more space on the die. On the other hand pixel processors need just a few dot products and arithmetic operations which need less space on the die but there are more pixels than vertices. Thus GPU manufacturers integrated multiple vertex processors into the

die and linked each vertex processor with multiple pixel processors. As shown in Figure 92. This massive parallel architecture gives a huge speed up in graphics processing. Later GPUs are used not only for graphics processing but also in general purpose computing. An example, assume that GPU and CPUs have same Arithmetic Logic Unit (ALU), example CPU is a 3GHz with 8 Cores and example GPU is a 1GHz with 1024 Cores Test Code is a 1000.000.000 cycle Task; the CPU can perform 3 Tasks per core per second and 12 times with 4 core. On the other hand, the GPU can perform 1 task per core but 1024 times with 1024 core. This gives a speed-up about 43 times and we can still use the CPU with GPU. It is obvious that this is a just theorical, in practice there are some bottlenecks effects on the GPU performance. The biggest bottleneck is the graphics bus speed. The CPU and the memory are in cooperation and they are connected from the north bridge with GPU. GPU bus can access the data only when there is no data transfer between CPU and memory (Figure 93).

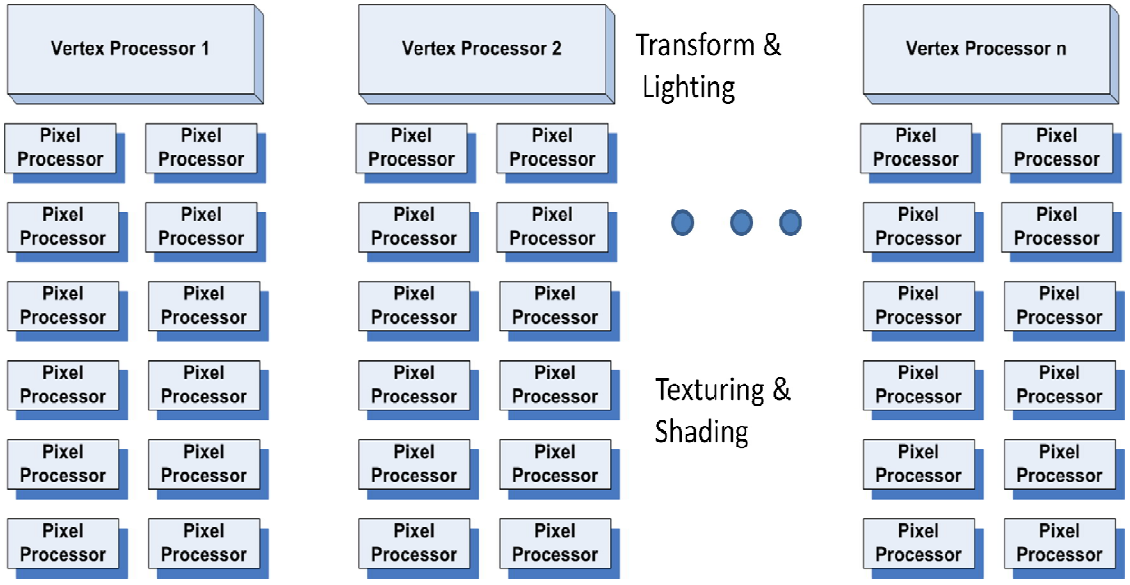


Figure 92 - GPU architecture

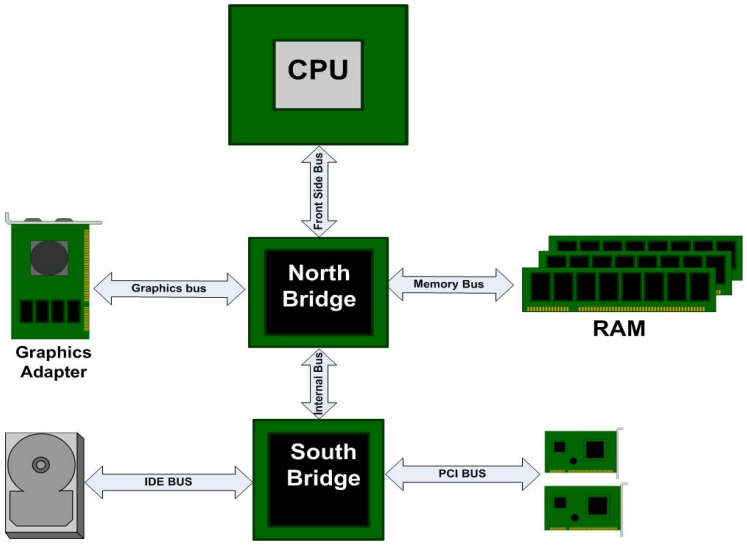


Figure 93 - Computer architecture

3.5.4 Architecture of DMRenderer

The architecture of DMRenderer can be seen on Figure 94 and its components are listed below.

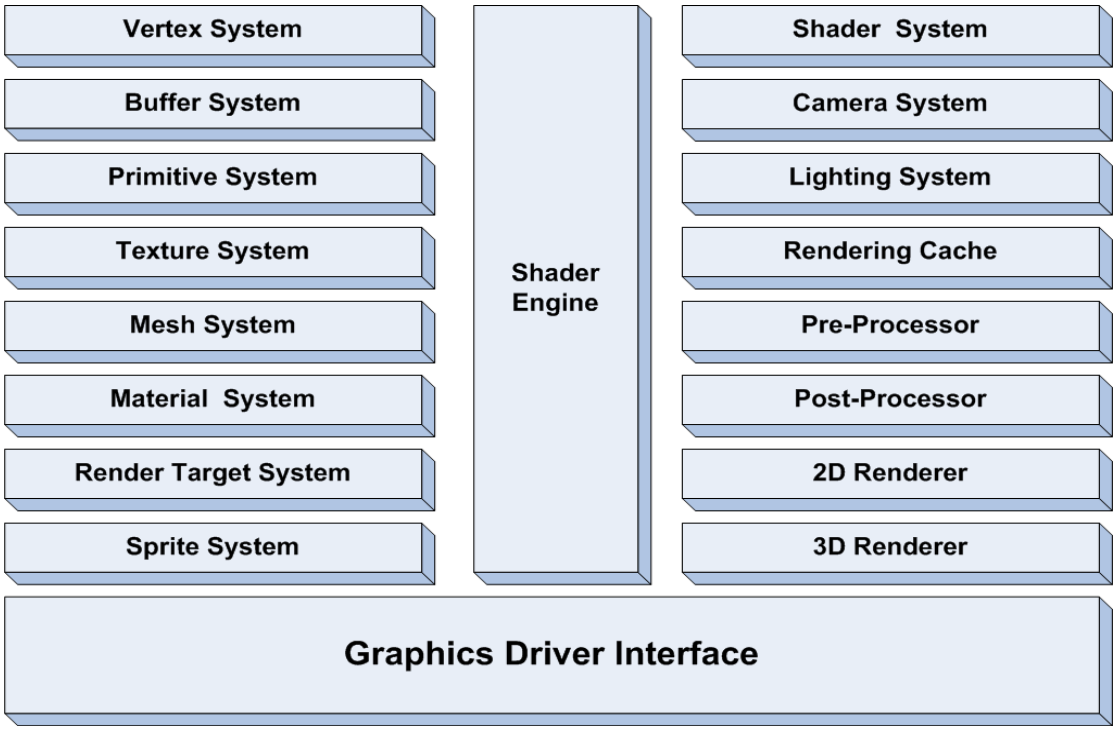


Figure 94 - DMRenderer architecture

Vertex system:

Like all of the processors GPUs have their own registers for storing and performing instructions. The vertex system is responsible for registering the vertex that is used with the GPU registers for shaders (the GPU programs) to access the correct data (Figure 95). The user can access to a pre defined vertex type or can register his/her own vertex declaration via the vertex manager.

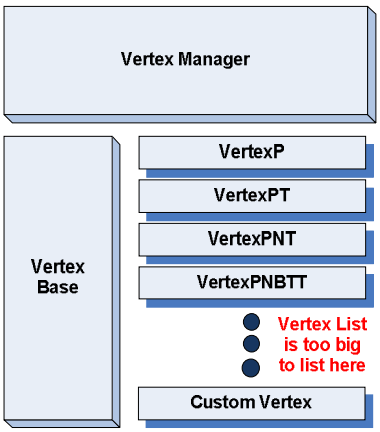


Figure 95 - Vertex system

View port system:

When multiple views of the scene in the same window are needed, view ports are used. The rendering engine automatically handles the rendering process for each created view ports. The view port system is shown in Figure 96.

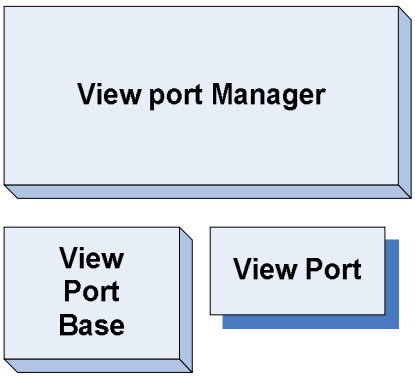


Figure 96 - Viewport system

Buffer system:

When and where the data is stored is one of the biggest problems depending on the computer's transfer limits. Buffer system is crucial for storing the data appropriately on system memory or GPU Memory. Buffer system is shown in Figure 97.

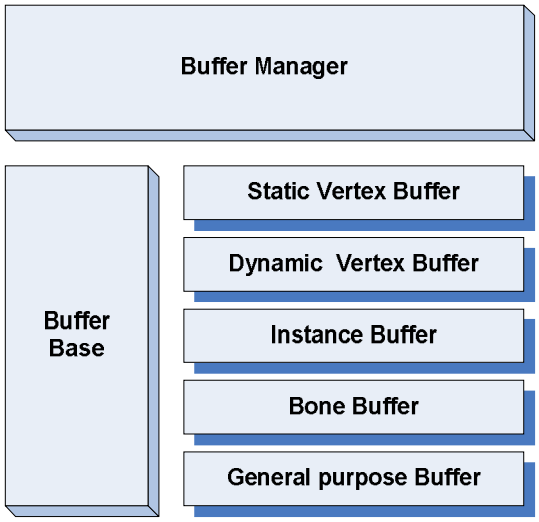


Figure 97 - Buffer system

Primitive system:

Although the default rendering primitive is triangle, sometimes more flexible geometries for rendering different tasks are needed. Like points for rendering a point cloud, lines for rendering a line geometry or plots, quads for rendering imposters. The primitive system is presented in Figure 98.

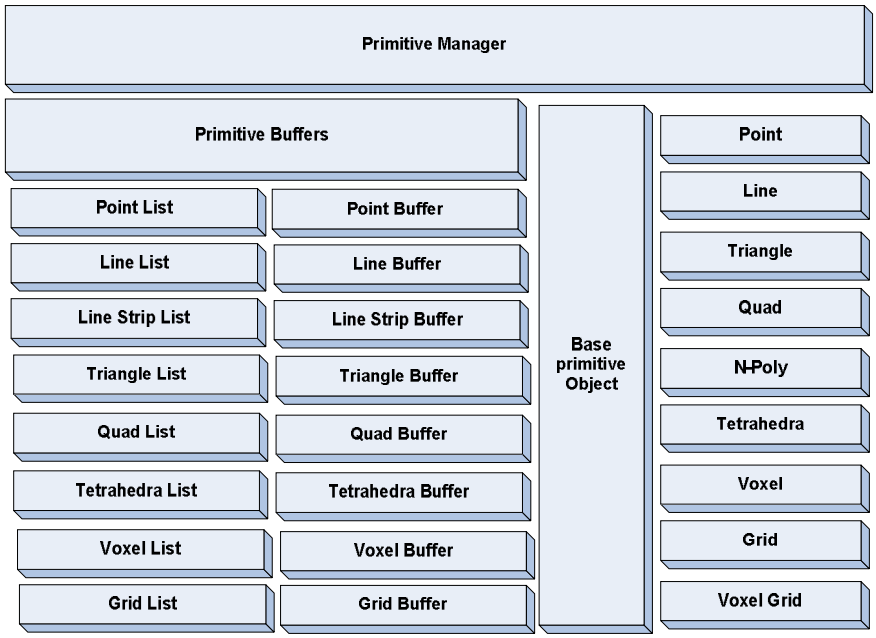


Figure 98 - Primitive system

Texture system:

Texture system is responsible for managing and handling textures (Figure 99). All Textures are loaded and stored by texture manager. Texture operations, filters converters are kept in the texture library. Texture system supports 1D textures for look-up tables, 2D textures for images, normal maps, height maps, displacement maps, 3D textures for storing volumetric data and other texture types.

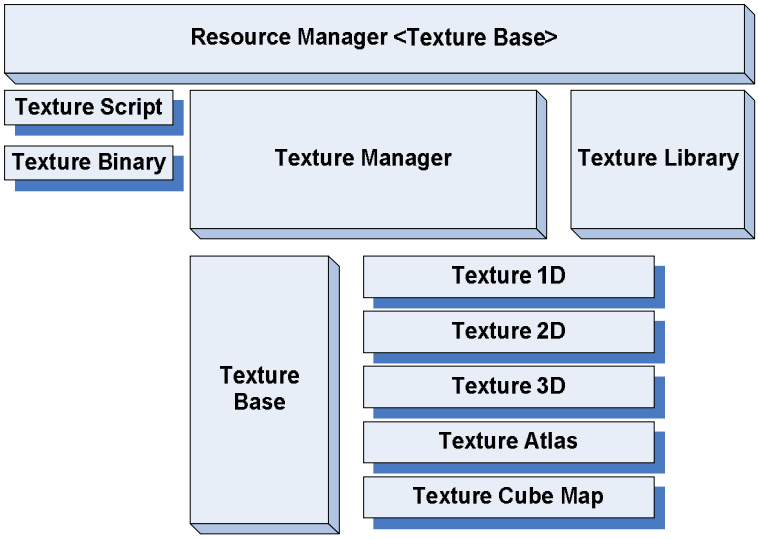


Figure 99 - Texture system

Mesh system:

Mesh system is responsible for handling and managing meshes (Figure 100). The mesh operations are accessible by mesh manager. Mesh library supports mesh operations, conversion and processing methods. Mesh system supports various kinds of meshes.

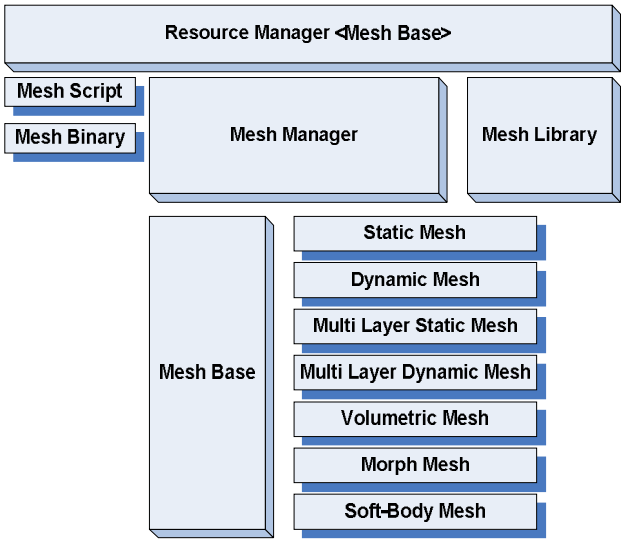


Figure 100 - Mesh system

Material system:

Material system is responsible for managing the mesh material properties (Figure 101), such as diffuse, specular, emissive lighting parameters, diffuse, normal map, height map, specular map, detail map textures and more.

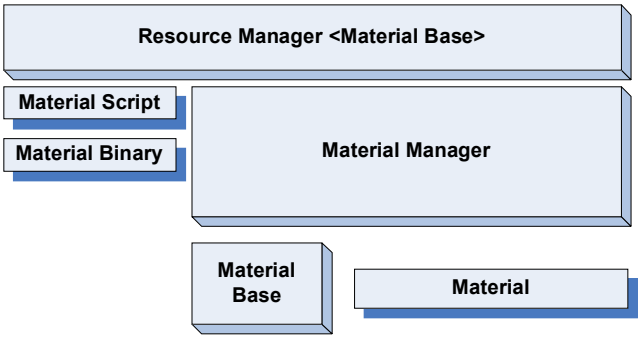


Figure 101 - Material System

Render target system:

Off-screen render targets are used for creating virtual back buffers for GPU computing. Render target system is responsible for creating, releasing and managing these buffers (Figure 102).

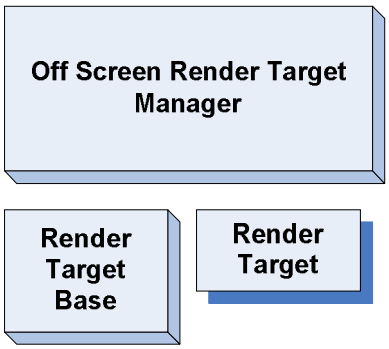


Figure 102 - Off secreen render target system

Sprite system:

Sprites are 2D objects in rendering therefore they are subject to 2D transforms. Sprites are useful in 2D rendering such as rendering GUIs or HUDs. Sprites are handled by sprite system (Figure 103).

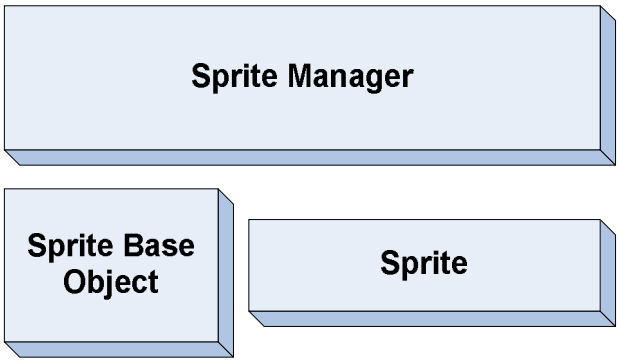


Figure 103 - Sprite system

Camera system:

Camera system is responsible for creating and managing the camera objects (Figure 104). Camera system supports target cam which always tracks the targets, path tracking cam which tracks the given path while tracking the target and stereo cam which renders for two eyes.

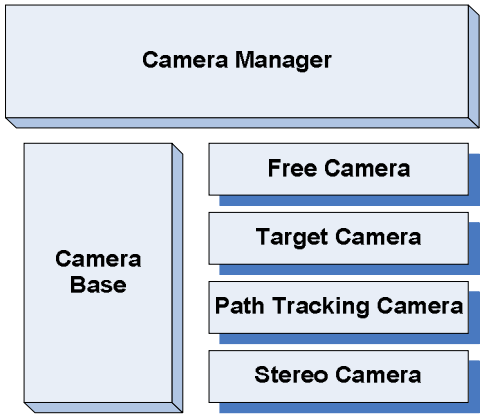


Figure 104 - Camera system

Lighting system:

Lighting system is responsible for modeling the lights in the scene for photo-realistic lighting calculations (Figure 105). Each light in the system has an additive light energy thus every light in the scene has to be handled separately. There are three types of light models in rendering engine; these are directional light model, point light model and the spot light model.

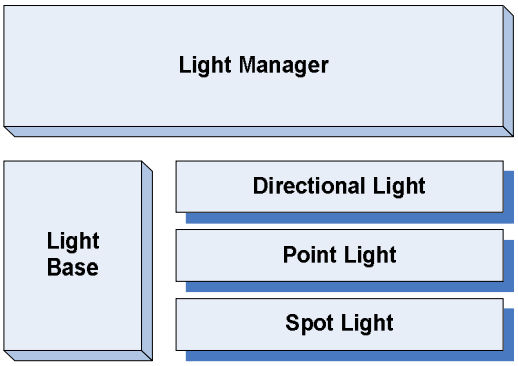


Figure 105 - Light system

Render cache system:

Rendering cache system is one of the most important parts of the rendering engine. Switching between shaders and textures are computationally expensive thus rendering cache system is responsible for sorting the objects for the best order before rendering (Figure 106).

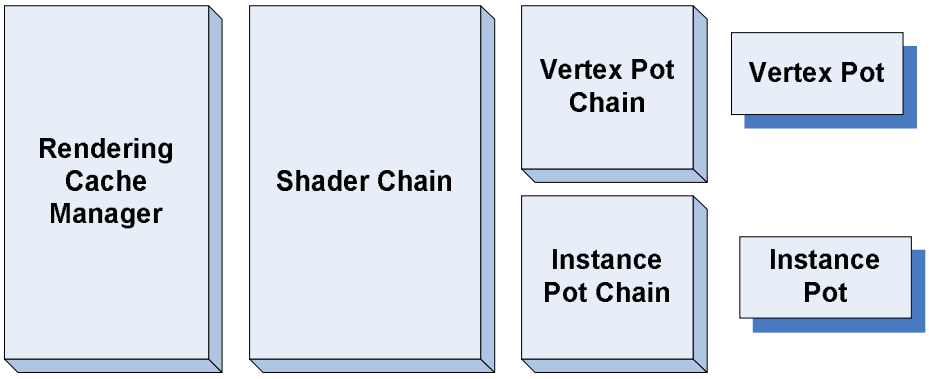


Figure 106 - Render cache system

Pre-processing system

Pre-processing system is responsible for preparing the rendering engine for rendering (Figure 107). Firstly, the culling processor eliminates the invisible objects in order to decrease vertex count. Secondly, the level of detail processor decides which mesh level should be used. Thirdly, pre-fetch processor fetches the required data for rendering. Fourthly, the rendering cache system sorts the rendering order and finally pre-computing processor calculates the required maps for rendering such as shadow, reflection, refraction, caustics, scattering maps.

Post-processing system:

Post-Processing system is responsible for performing post-processing tasks and enhancing the resulting image generated by the 3D renderer, before presenting to the display (Figure 108). Post processing system supports high dynamic range imaging.

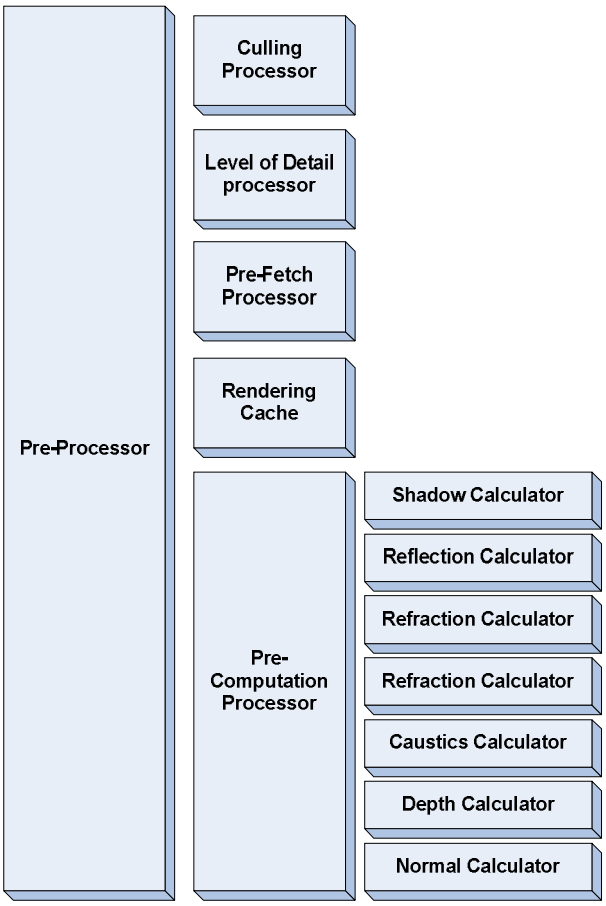


Figure 107 - Pre processing system

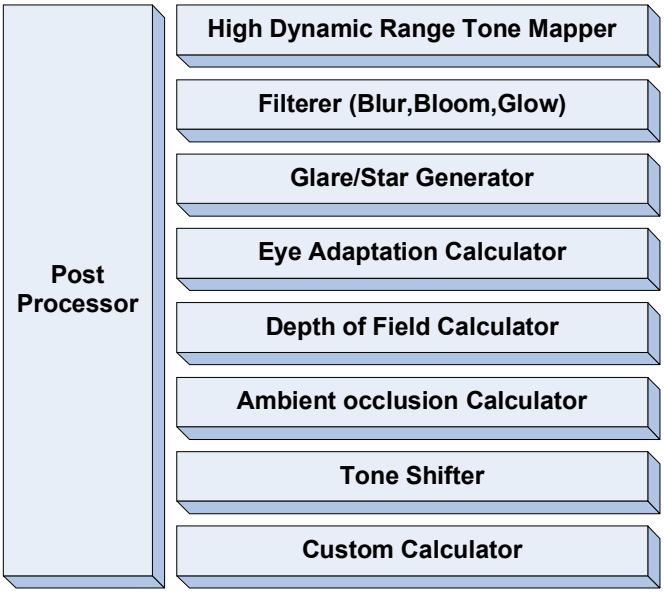


Figure 108 - Post processor

2D renderer:

2D renderer is responsible for rendering 2D objects like sprites for GUIs (Graphical User Interfaces) and HUDs (Heads Up Displays).

3D renderer:

The 3D renderer is responsible for rendering 3D objects and performing lighting equations.

Device driver interface:

Device Driver interface is responsible for accessing the graphics device (GPU).

3.5.5 DMShader – shader engine

Shaders are GPU programs. Shader engine is responsible for managing, loading, state tracking, message and data sharing with the shaders (Figure 109).

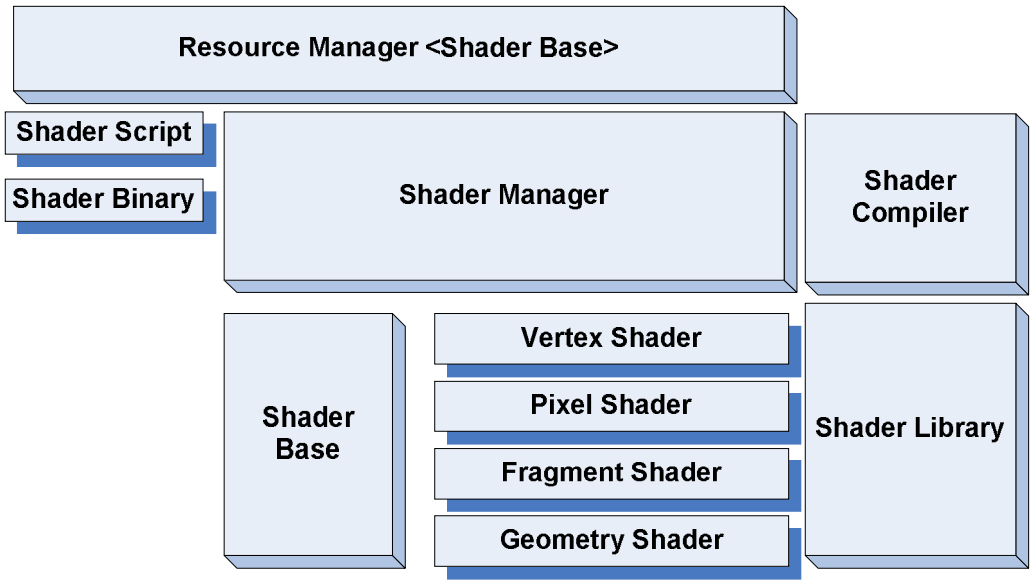


Figure 109 - The Shader Engine

3.5.6 Volume rendering techniques

In DMRenderer there are 4 volume rendering techniques. These techniques can be classified into 2 as mesh based (using a mesh of vertices) and meshless volume rendering methods (using a 3D image of the volume).

3.5.6.1 Evaluating the volume data - transfer functions

The MRI data is a 3D image contains different intensities for different tissue densities. MRI data should be classified into three; the skin, the tissue and the bone thus a transfer function should be created. Figure 110 shows an example of a transfer function for classification of the tissue and bone. Transfer function creation is an easy step. For each tissue densities, control points should be created, and these control points should be connected with a cubic spline interpolation technique. Finally, a lookup table should be constructed by these splines.

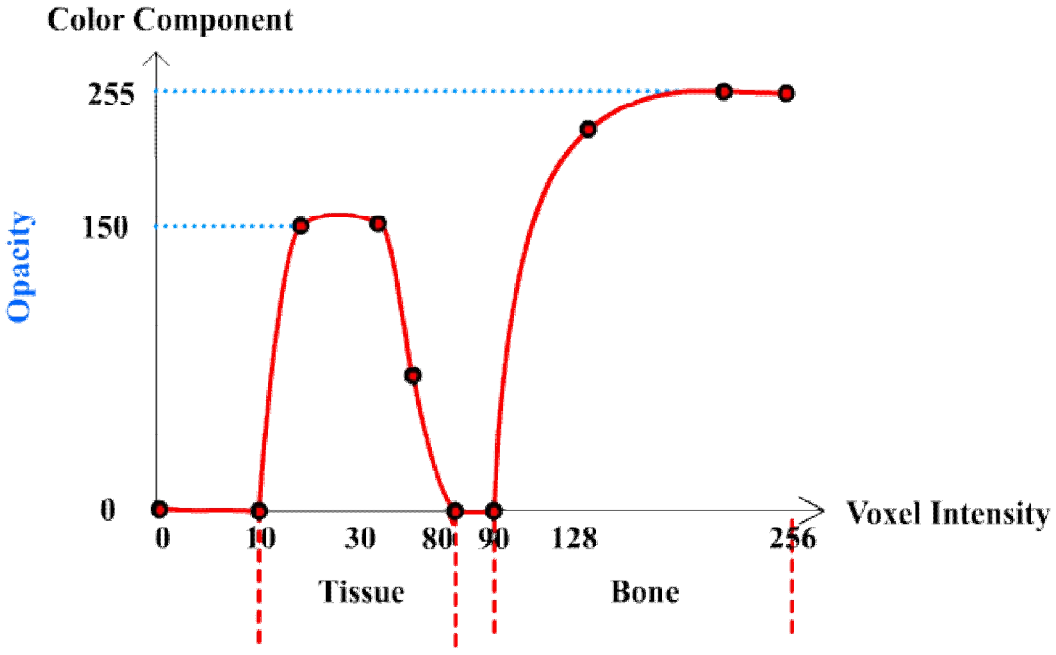


Figure 110 - Transfer function of an MRI image for extracting Tissue and Bone information

3.5.6.2 Mesh based volume rendering techniques

Mesh based volume rendering techniques are based on creating a 3D mesh of the 3D volume data. MRI data is used in this thesis.

Marching cubes

Marching cubes is an algorithm developed by Lorensen and Cline for extracting a polygonal mesh of iso surfaces from 3D volume data. Algorithm takes 8 neighboring pixels from the volume data and creates an imaginary voxel. Using a look-up table, compare function determines which mesh configuration is appropriate for the evaluated voxel. After finding the mesh configuration, the vertex positions are calculated by interpolating the connected voxels. As seen on Figure 111 to 114.

The vertex normals are extracted from the gradient of the volume data at each grid point which are needed for lighting and shading.

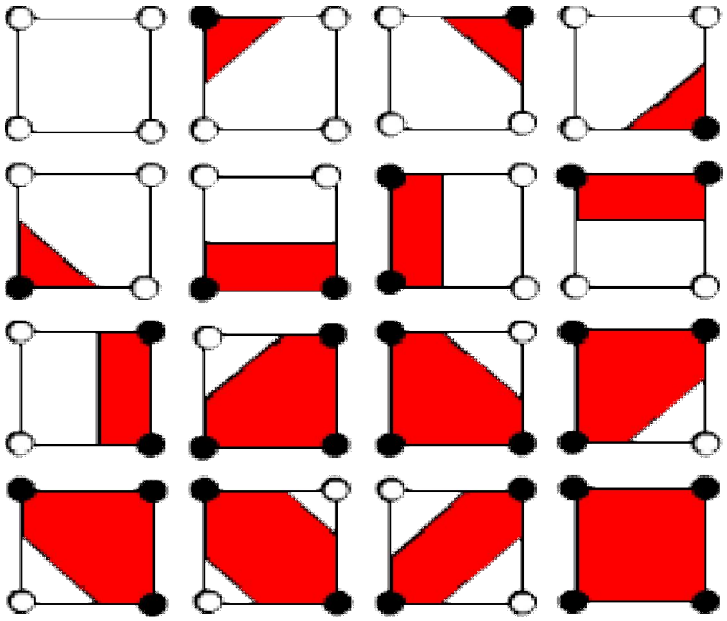


Figure 111 - Marching squares configuration

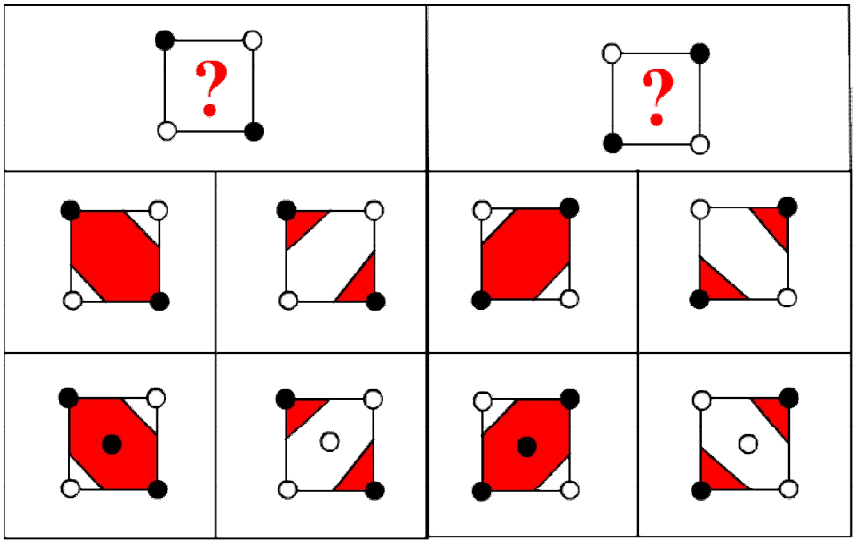


Figure 112 - Selecting the correct mesh

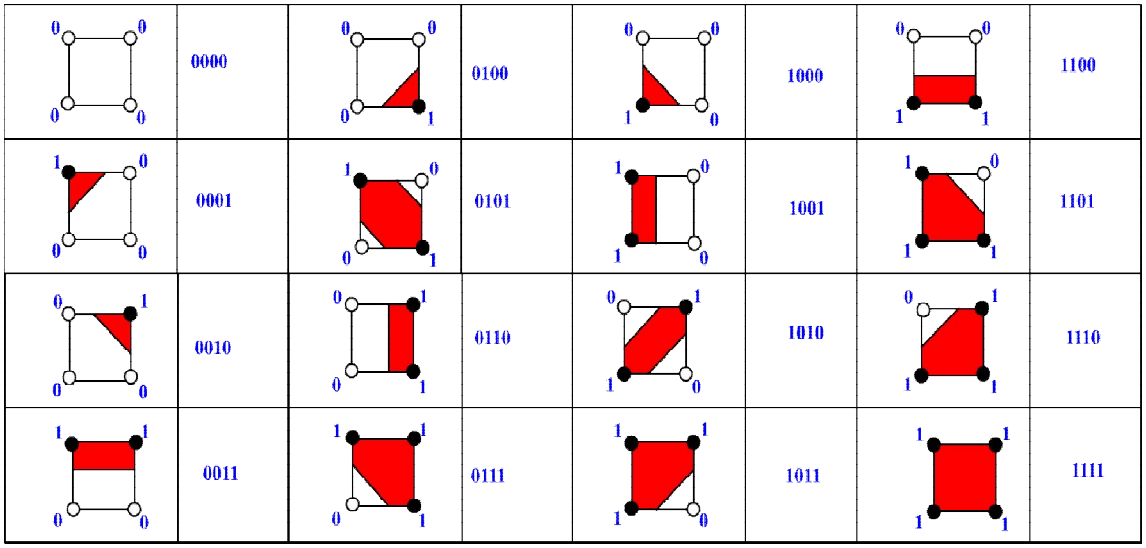


Figure 113 - Look-up table generated for the possible 16 configuration

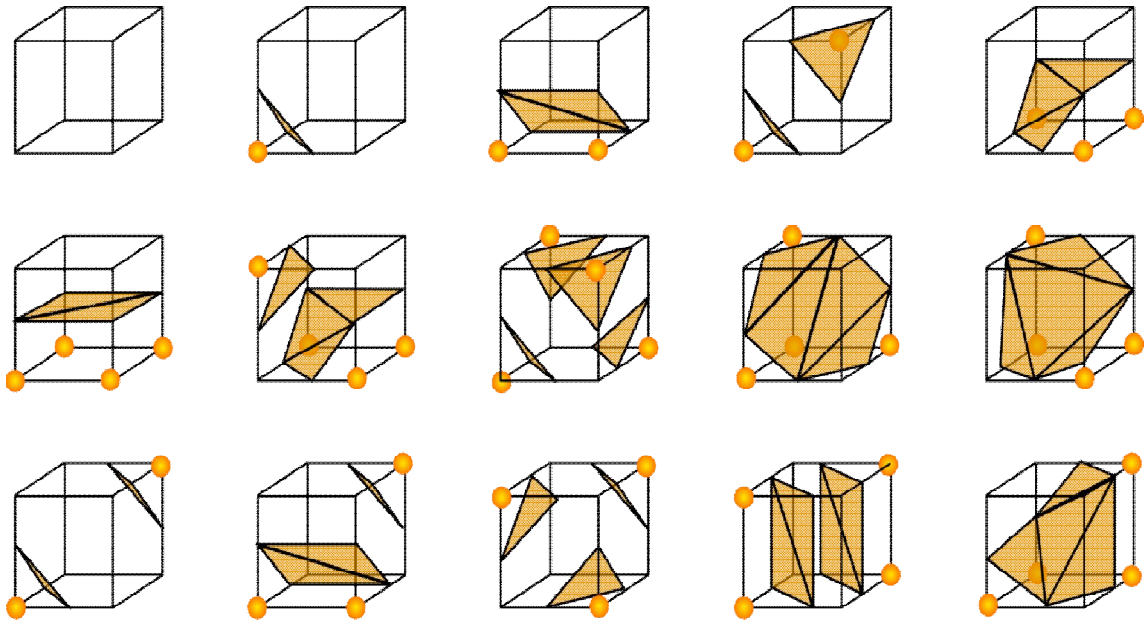


Figure 114 - First 16 of 256 configurations of marching cubes

Marching tetrahedrons

Marching Tetrahedrons is a technique similar to the marching cubes. Voxels are divided into 6 tetrahedrons and for each tetrahedron a look-up table is created as seen on Figure 115.

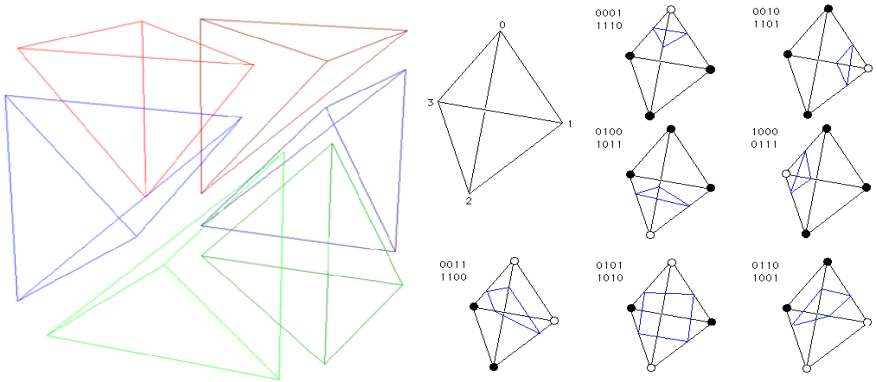


Figure 115 - Six tetrahedrons of a voxel (left) and mesh configuration for one tetrahedron (Right)

3.5.6.3 Meshless volume rendering techniques

Meshless methods are useful when the physics calculations are not needed or when the mathematical model is not stable.

Cube rendering

Cube rendering is a method of rendering each voxel in the MRI data by representing with a cube. Cube rendering doesn't give photorealistic results but it is useful when examining the MRI data or debugging.

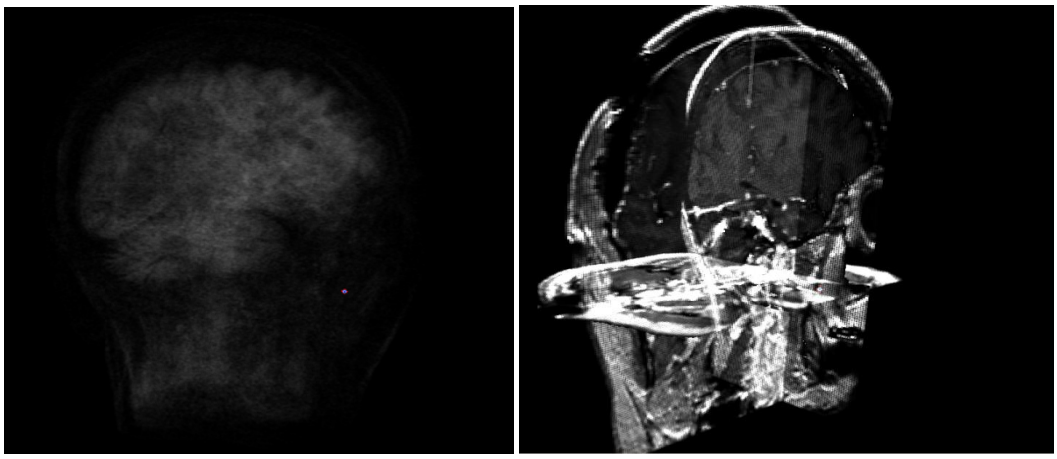


Figure 116 - Cube rendering of brain MRI (Left) and cube rendering of brain as axial slices (right)

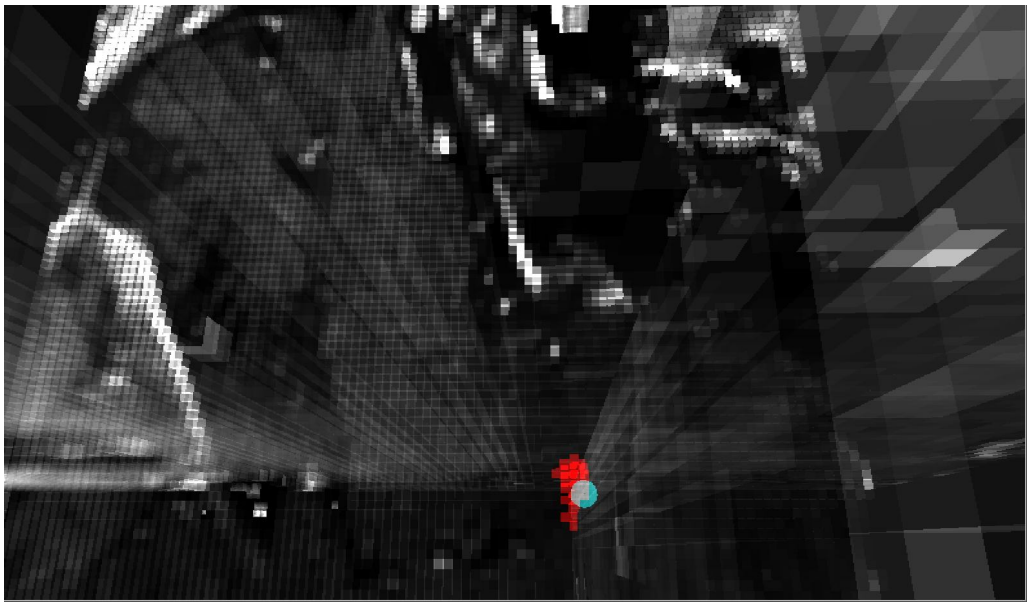


Figure 117 - Cube rendering example for debugging

Ray tracing – volume ray casting

Volume ray casting is an image based volume rendering technique that provides very high quality rendering. The algorithm runs as tracing the rays casted from the viewpoint into the MRI data. The rays are advanced as small deltas as possible by applying a lighting equation selected for the illumination model by multiplying with the transfer function and adding results by back to front order alpha blending technique (Figure 118 - 120). In order to decrease computation, the MRI volume is divided into an oc-tree, blank volumes and non visible volumes are eliminated (Figure 121). The algorithm runs entirely on GPU.

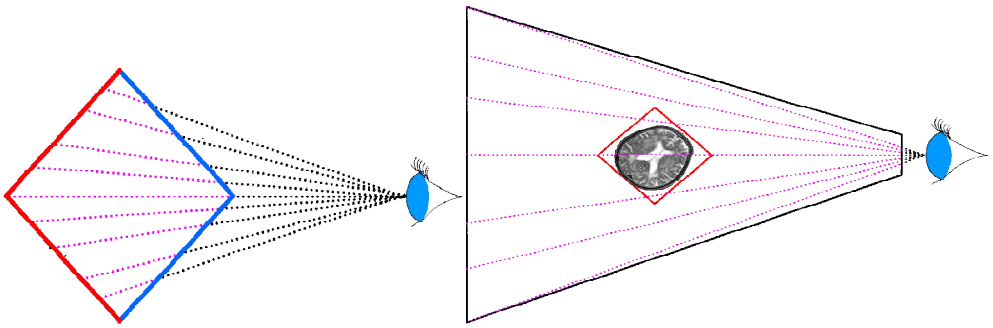


Figure 118 - Rays from eye to the back plane

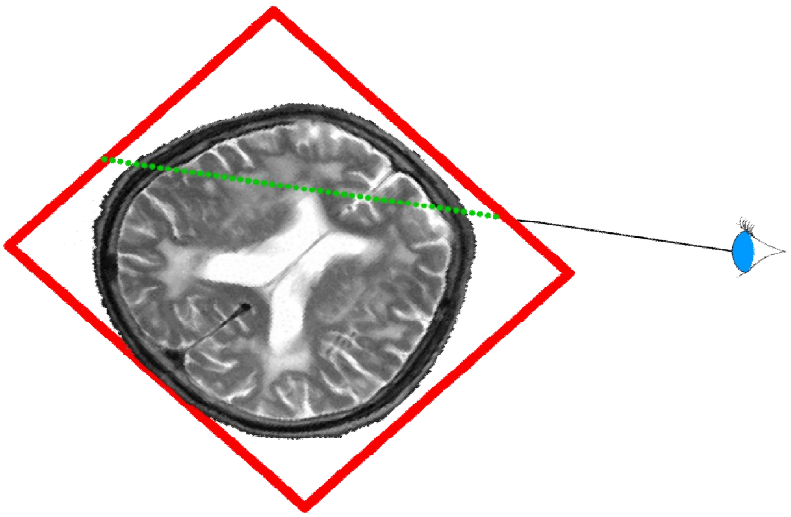


Figure 119 - Rays From eye intersecting the front face to back face

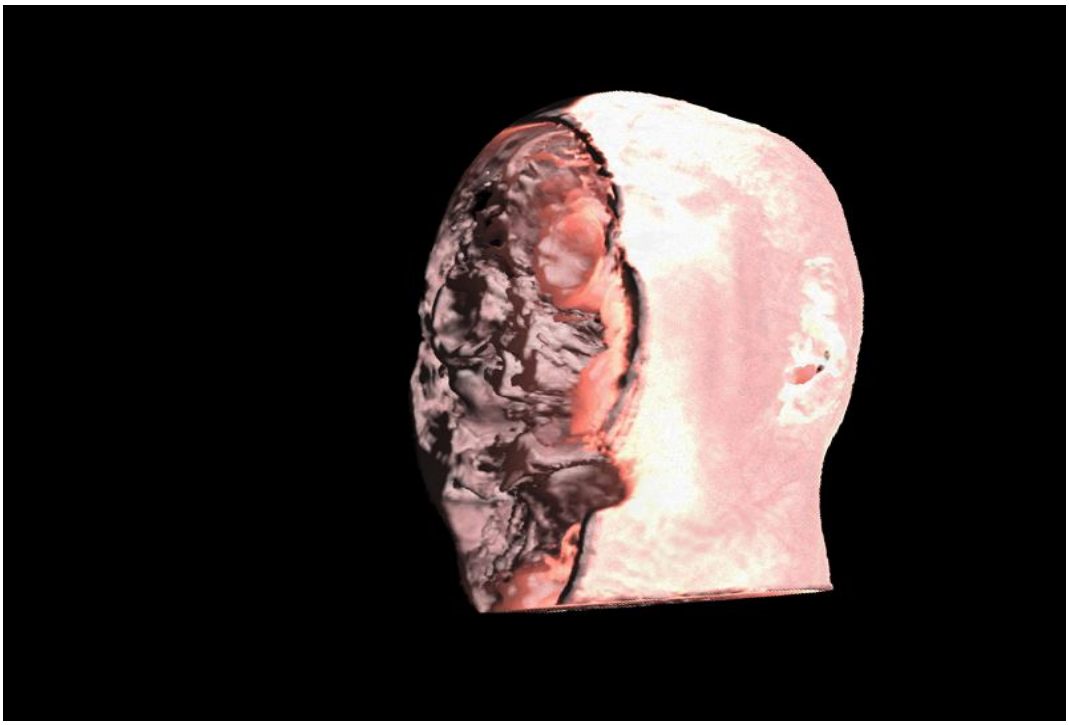


Figure 120 - Volume ray casting to the brain MRI data

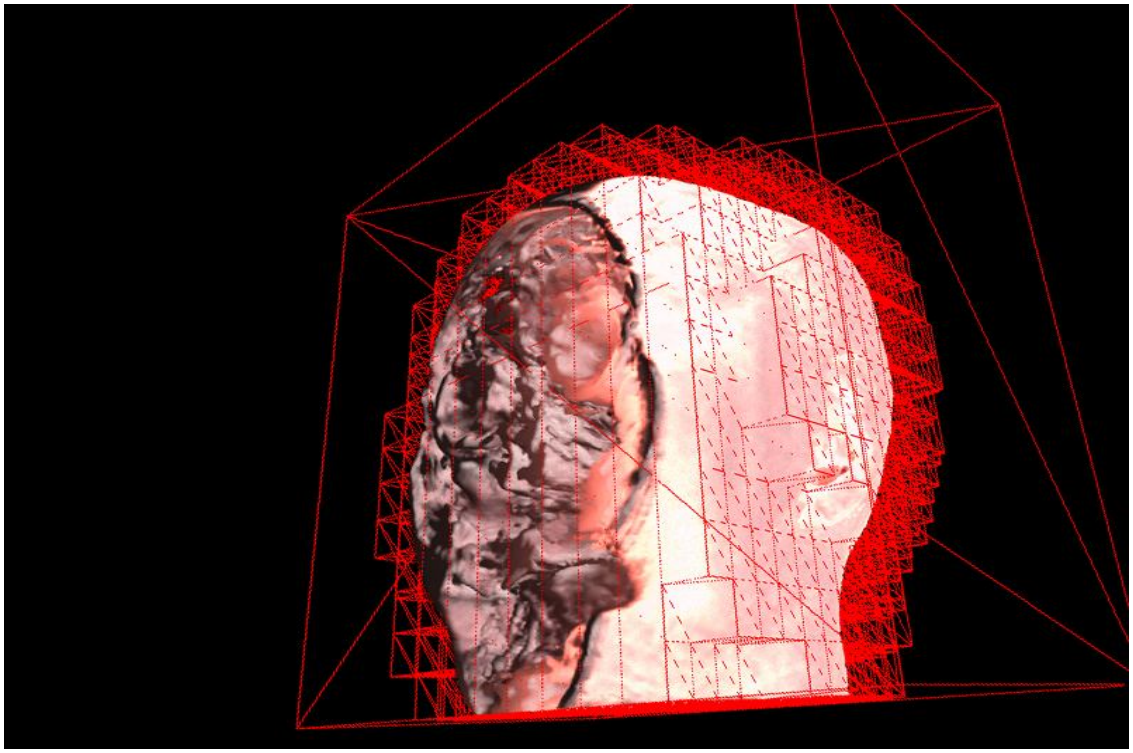


Figure 121 - Volume ray casting using smaller cubes

3.5.7 Tessellation

Tessellation is a process for modifying the geometric primitive's detail or vertex count on run-time. For triangles, in order to decrease vertex count removing a vertex between two vertices creates one triangle instead of four and in order to increase the mesh detail (Figure 112), adding a vertex between two vertices creates four triangles instead of one. In computer graphics tessellation is used for decreasing the vertex count on distant objects. Tessellation is classified into two, continuous tessellation which deals with the mesh on run-time and discrete tessellation which uses pre-computed tessellated meshes.

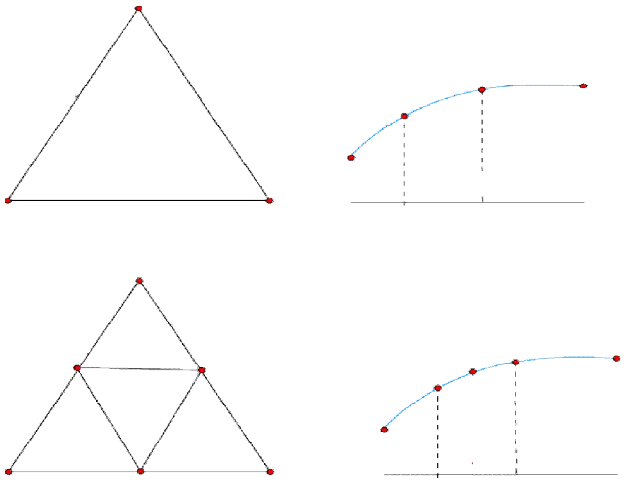


Figure 122 - Tessellation of a triangle

Continuous tessellation

Continuous tessellation is a method for modifying the mesh count in runtime. The vertex count of mesh is increased when the object is close to the viewer, and decreased one by one when getting further (Figure 123). Continuous tessellation is useful as it decreases the vertex count for distant meshes but requires additional continuous processing which is not accepted in the scope of this thesis. Thus, discrete tessellation is used in the platform.

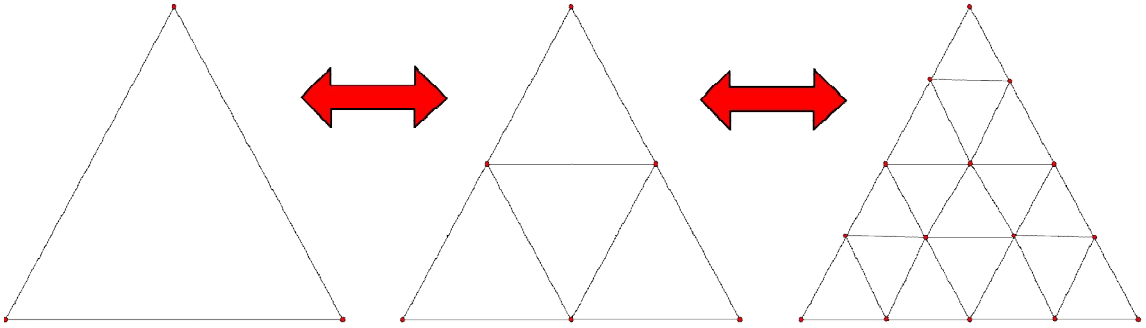


Figure 123 - Continuous tessellation steps

Discrete tessellation

Discrete Tessellation is a method of merging pre-tessellated meshes. The region that the surgeon focused is rendered with the actual vertex count but the distant regions are rendered with gradually decreasing mesh count. If the focused region changes, the mesh of the region is gradually replaced by a higher resolution one and the distant regions are replaced with the lower resolution ones (Figure 124). Using ten discrete levels decreases the vertex count 80 times in 2D (Table 8, Figure 125) and 1000 times in 3D (Table 9, Figure 126).

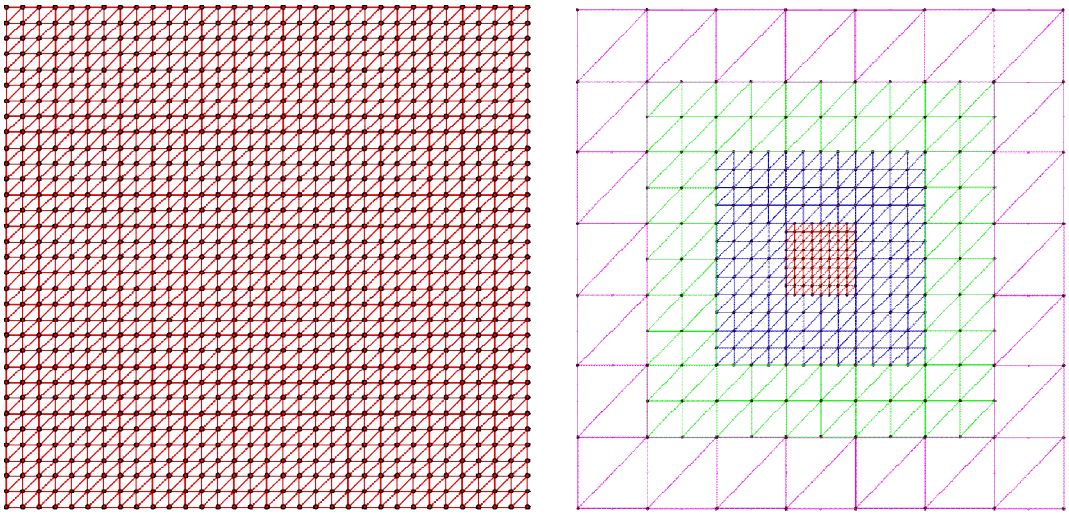


Figure 124 - The original mesh (left) and discrete tessellated mesh (right)

Table 8 - Tesselation speedup in 2D

d	n		Vertex Count	Vertex Count	**Speedup**
0	1	2	2	2	1
1	2	8	8	8	1
2	4	32	288	96	3
3	8	128	3200	512	6.25
4	16	512	25088	2240	11.2
5	32	2048	165888	9216	18
6	64	8192	991232	37184	26.657487
7	128	32768	5537792	149120	37.136481
8	256	131072	29491200	596928	49.404953
9	512	524288	151519232	2388224	63.444313
10	1024	2097152	757071872	9553472	79.245731

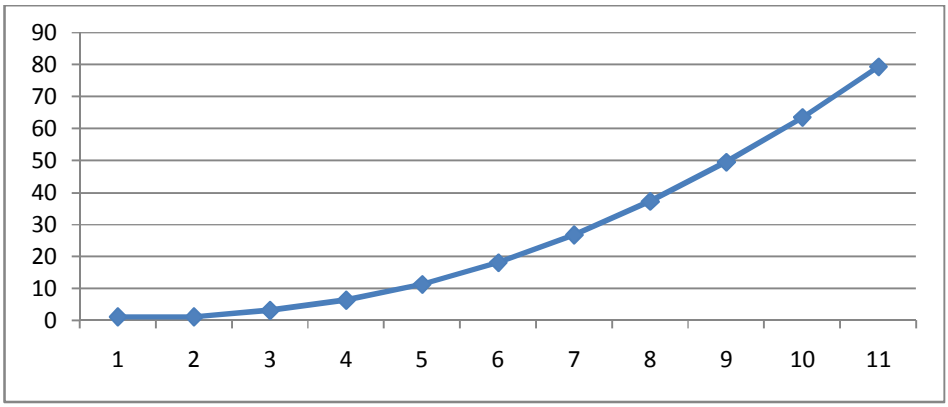


Figure 125 - speedup with tesselation in 2D

Table 9 - Speedup in 3D

0	1	1	1			1	1		1
1	2	8	27			34	216		6.352941
2	4	64	125			370	8000		21.62162
3	8	512	343			3178	175616		55.25991
4	16	4096	729			25810	2985984		115.691

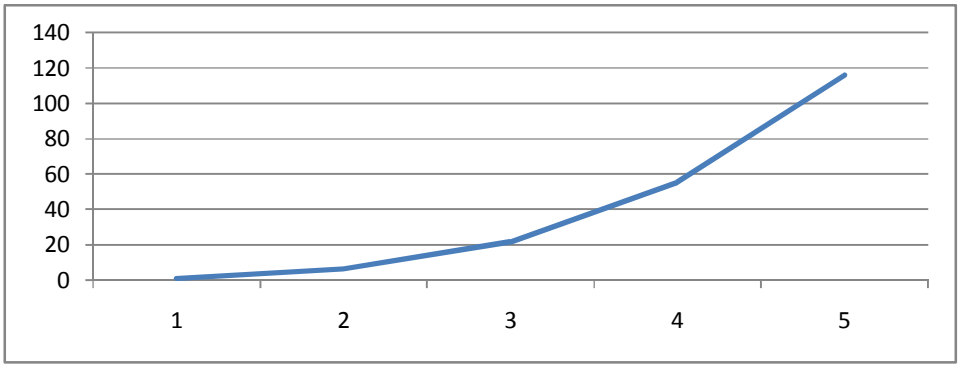


Figure 126 - speedup with tesselation in 3D

3.5.8 Culling

Culling methods are used for eliminating the non visible objects and parts of the objects. In pre-processing system four culling methods are used; frustrum culling, self culling, occlusion culling and back-face culling (Figure 127).

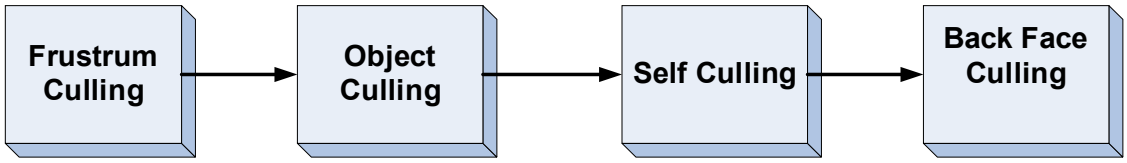


Figure 127 - Culling steps in the preprocessing system

Frustrum culling

Frustrum culling is a method for eliminating non visible objects which are not in the frustrum volume. The logic is simple if the object is in the frustrum volume, the object is visible. Testing each object with the frustrum is time consuming thus a oc-tree hierarchy created by coarse collision detector is used. Therefore, there is no need to create the tree again. Like collision oc-tree detection tests, the viewing frustrum is encapsulated with an AABB. Then it is tested with the tree as seen on Figures 128 - 132. After that only remaining boxes are tested with the viewing frustrum planes as shown on Figure 132.

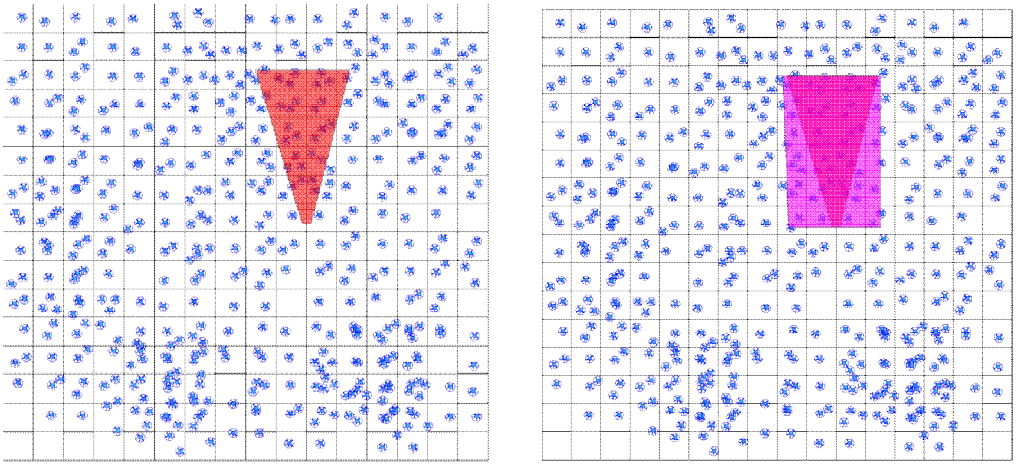


Figure 128 - Frustrum (left) and AABB of the frustrum (right)

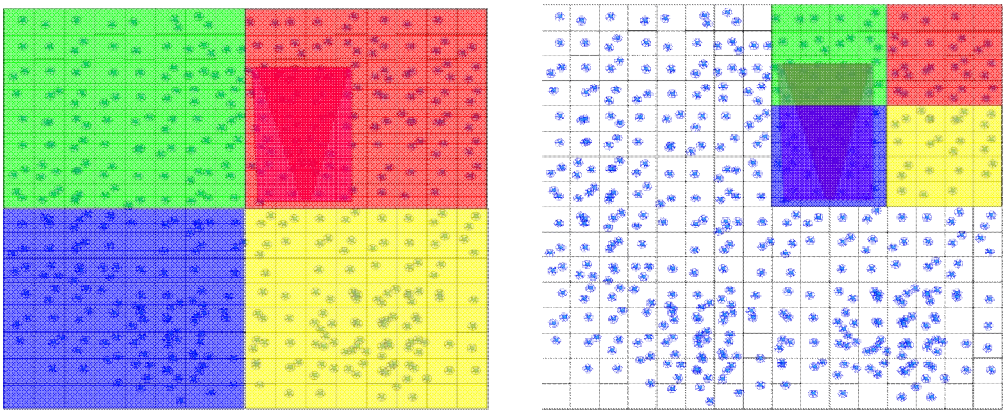


Figure 129 - Frustrum AABB is tested with first boxes in the scene oc-tree (left) and second depth boxes (right)

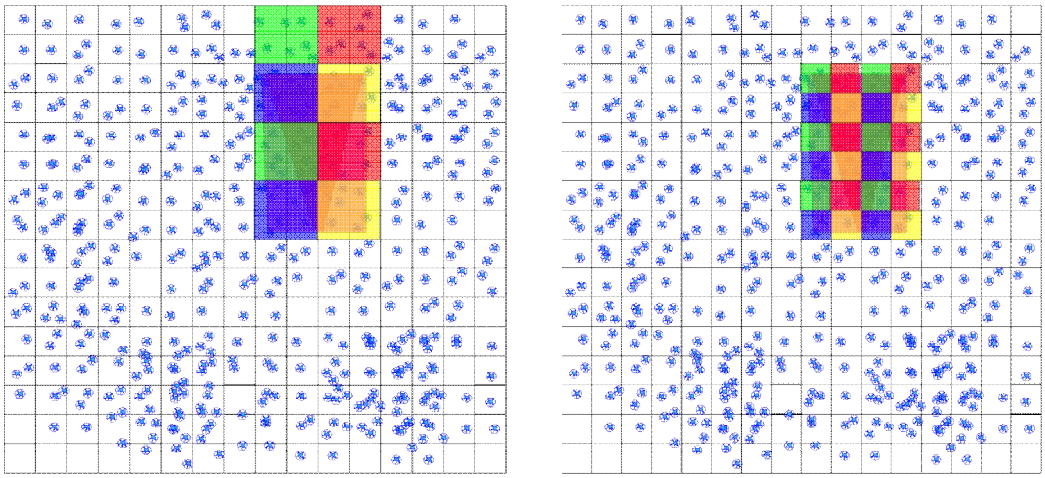


Figure 130 - Oc-tree test continues

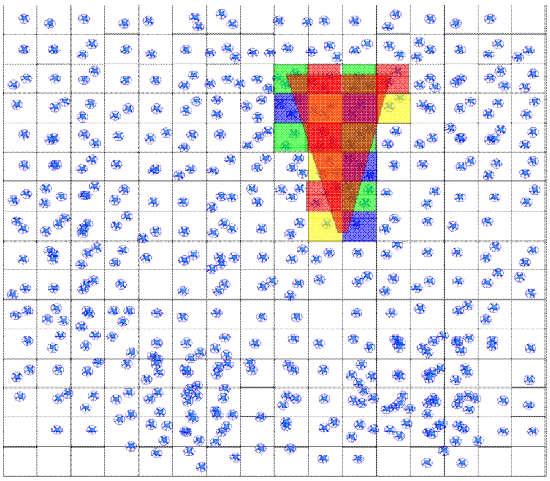


Figure 131 - The intersecting boxes are found

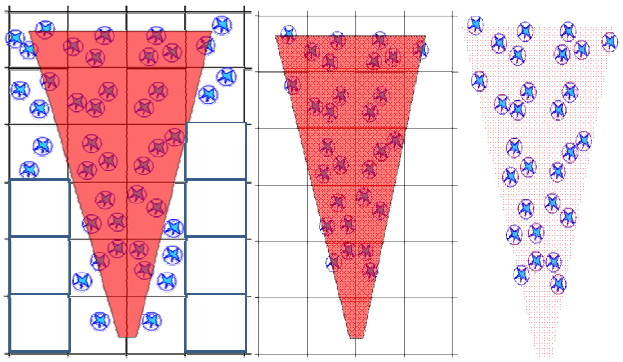


Figure 132 - Testing the remaining objects with the frustrum and finding only the insiders.

Self culling

It is only possible to see the outer shell of the objects. As a result, self-culling eliminates the inner volumes of the elements (Figure 133). An example of self culling in ray casting is shown in Figure 134.

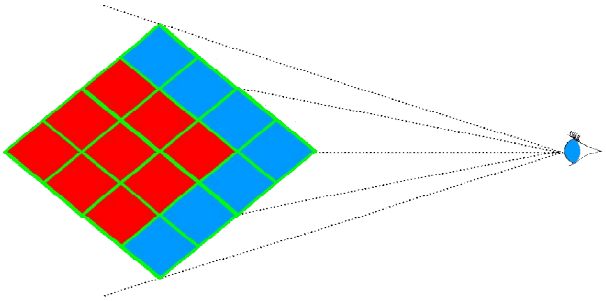


Figure 133 - The outer volume of the objects are visible (blue) but the inner are not (red).

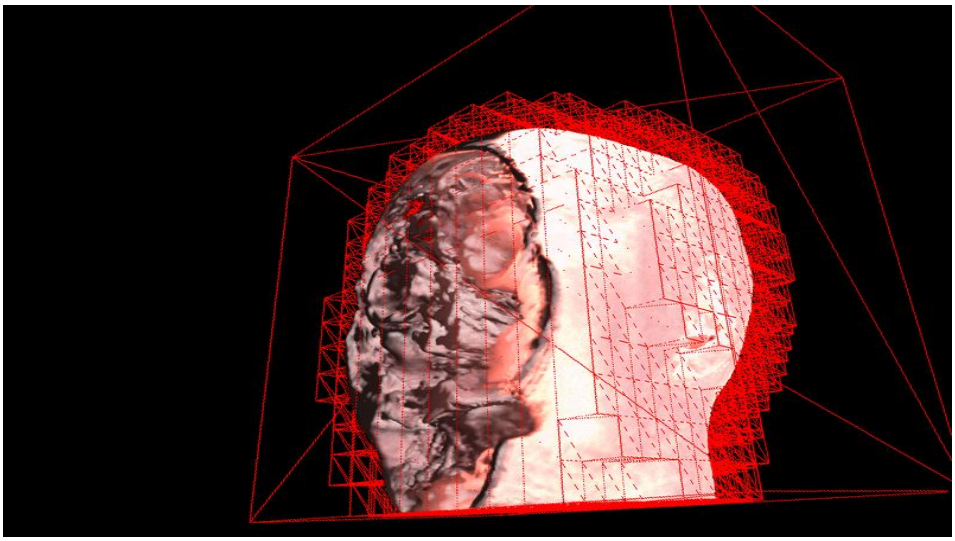


Figure 134 - Self Culling in Volume Ray casting

Occlusion culling

The objects near the front plane occlude the objects which are far away. Occlusion culling is a testing method to find if further objects are visible or not.

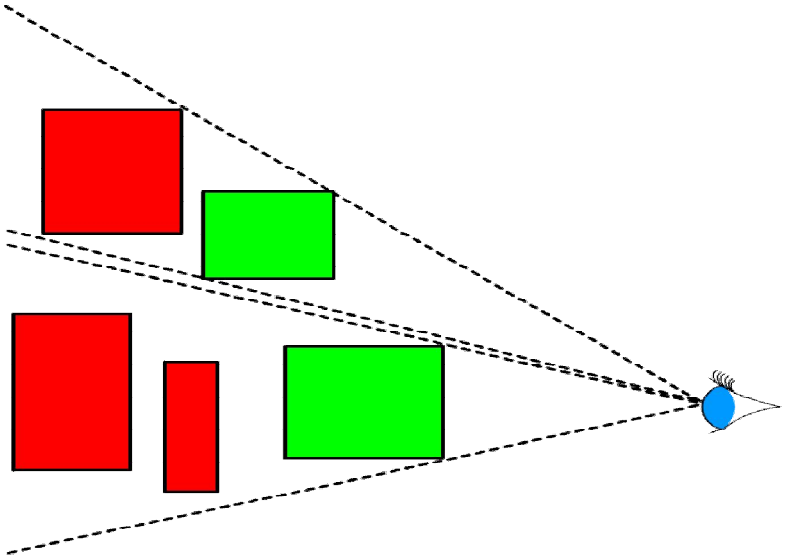


Figure 135 - Red boxes are occluded by the green ones. Red boxes are not visible.

Back-face culling

Like occlusion culling it is impossible to see the back of the objects. Thus in vertex space, vertices that are producing the back faces cannot be seen. This is a very cheap test. Dot producting of to eye vector and surface normal gives a scalar product. If it is smaller than 0, it is a back face and should be culled. This algorithm is adapted into modern GPU pipelines as a result this culling method automatically done by the GPU.

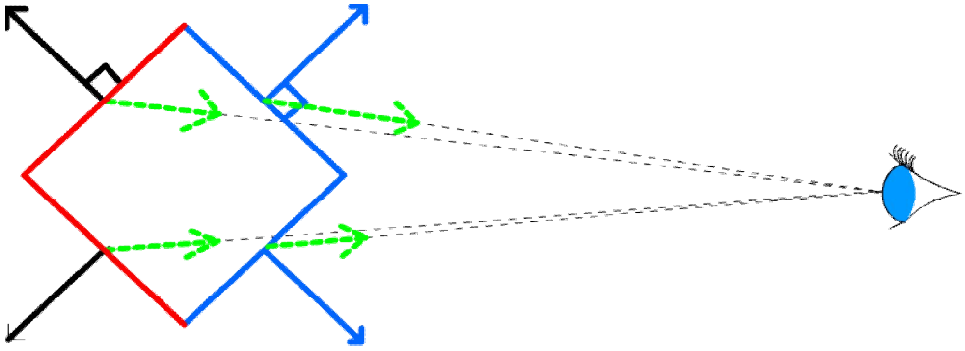


Figure 136 - Red surfaces are back faces and not visible.

3.5.9 Per-pixel lighting techniques

As stated earlier when presenting the GPU architecture, GPUs have more pixel processors. Thus, in order to benefit from this architecture some per pixel processing algorithms are developed.

Texturing

Texturing is a method of creating an illusion for showing down sampled mesh, which looks like the original by just storing the color data of the non sampled vertices. In general for photorealisticity a photo of the object is used. In general, textures use four channels red, green, blue and alpha channel (Figure 137 to 141).

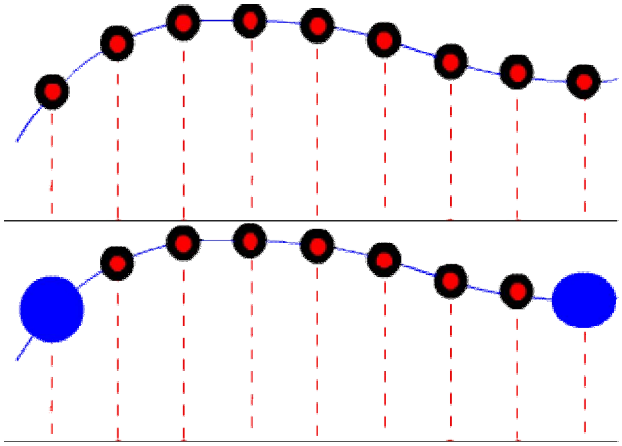


Figure 137 - Original mesh (Up) and down sampled mesh (Down) : Sample points shown in blue

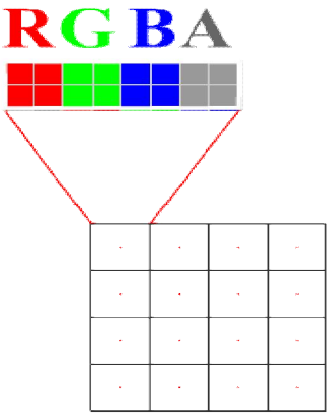


Figure 138 - Texture channels; red, green, blue and alpha

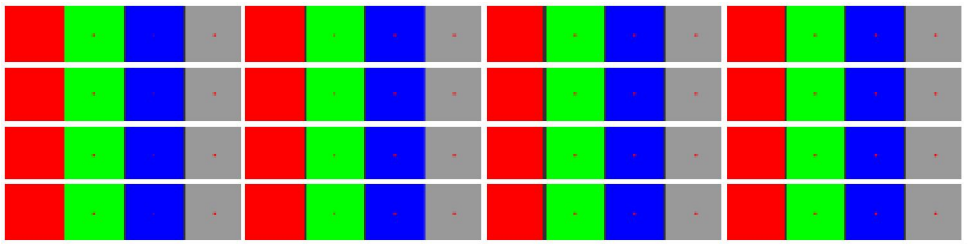


Figure 139 - A 4 by 4 texture in memory

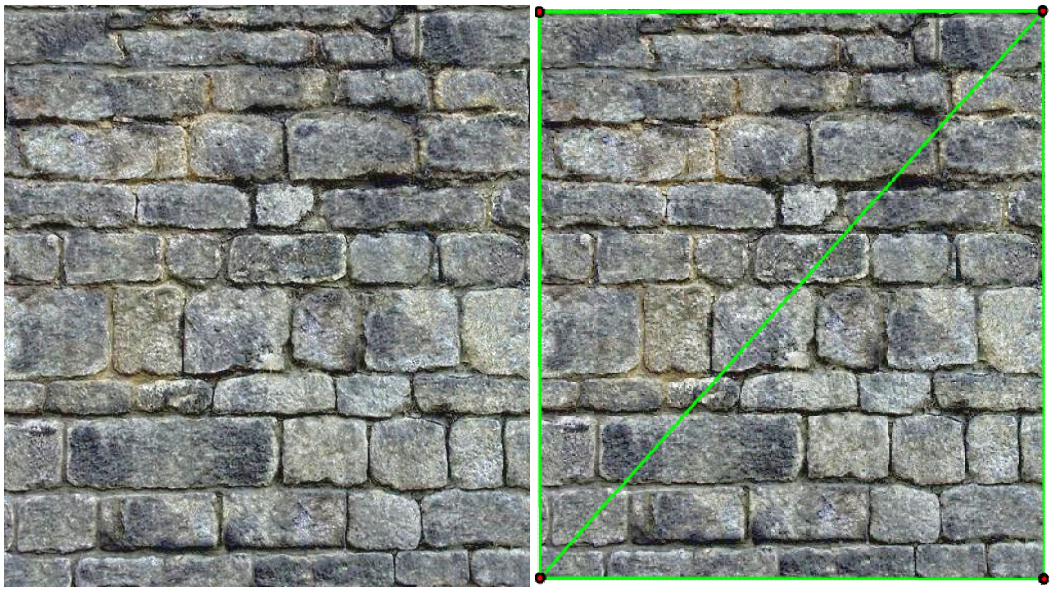


Figure 140 - Wall texture (left) and texture mapped to two by two mesh (right)

Figure 141 - Two by two mesh in Figure 140 is rendered

Mip mapping

Mip mapping is a technique for faster look-up operations. A far object, in the rastering stage covers, lets say 32 by 32 pixels, and suppose that the texture used for this object is 512x512 pixels. For each pixel in the raster pipeline matches to a region of 16x16 pixels in the texture. This means for each pixel in the raster pipeline we have to sample 256 pixels. The problem can be overcome by using mip maps. Mip maps are down sampled images of the original image by two (Figure 142-143). For storing mip maps, 1.333 times space of the original texture is needed but this is acceptable for rescuing from the need of more processing power. Equation 10 shows the space needed for N maps.

$$\lim_{N} \sum_{0}^{N} \frac{1}{2^{2n}} = 1.3333 \qquad (3.10)$$

Equation 10 - Space needed for storing N mip-maps

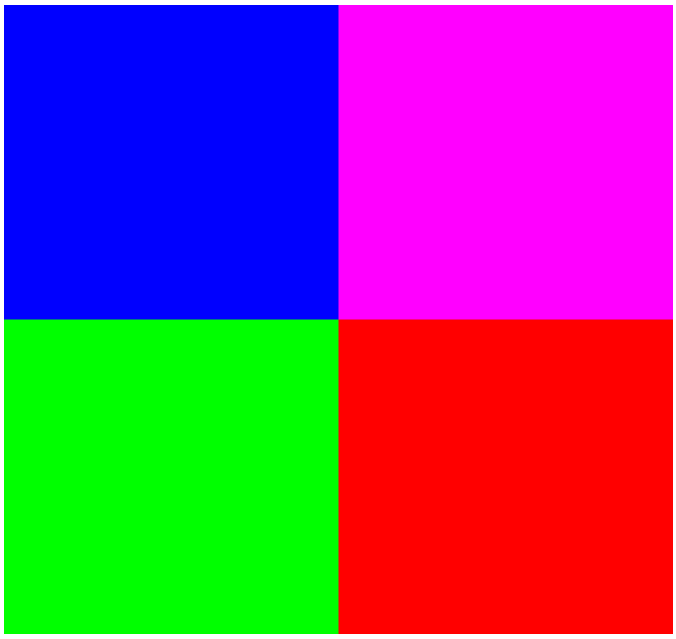


Figure 142 - Example texture

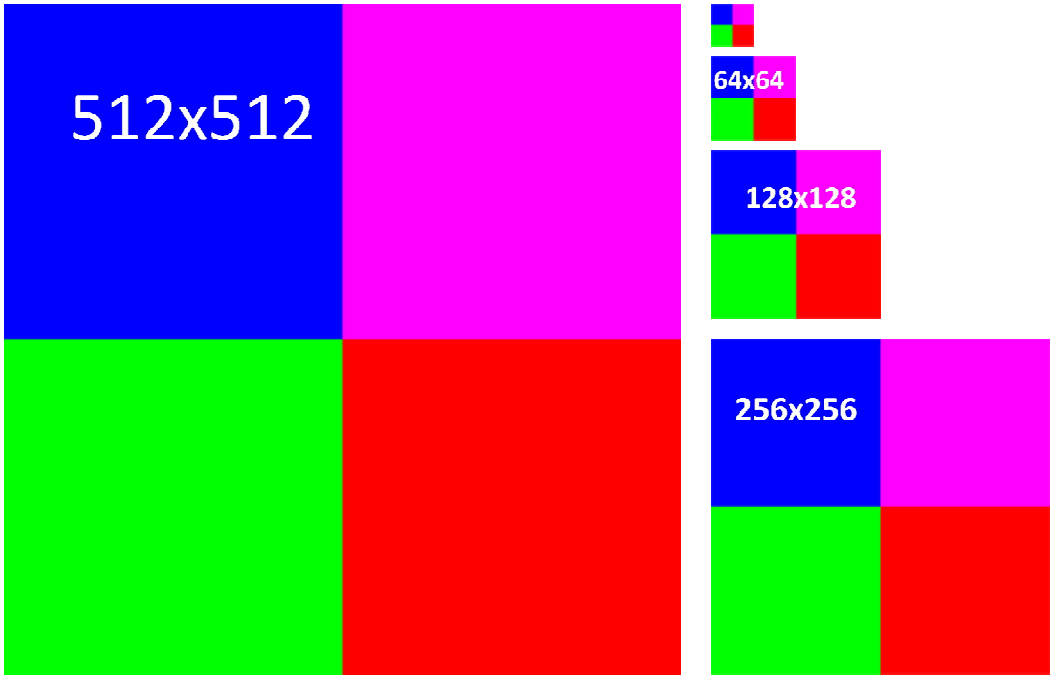


Figure 143 - Mip maps of the example texture

Tangent space normal mapping

Tangent space normal mapping is a technique for storing vertex normals within a special texture called normal map. While constructing the mesh, the tangent space of down sampled mesh is calculated. And missing vector normals are converted to this space and stored in normal map as x, y and z components respectively in red, green, and blue channels. In shading time, these normals are read from the normal map and a close lighting approximation is computed (Figure 144 - 149).

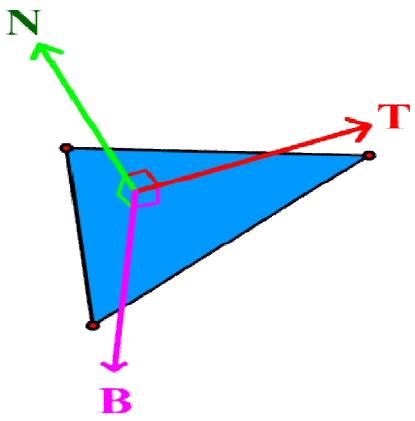


Figure 144 - Tangent, normal, binormal of a surface

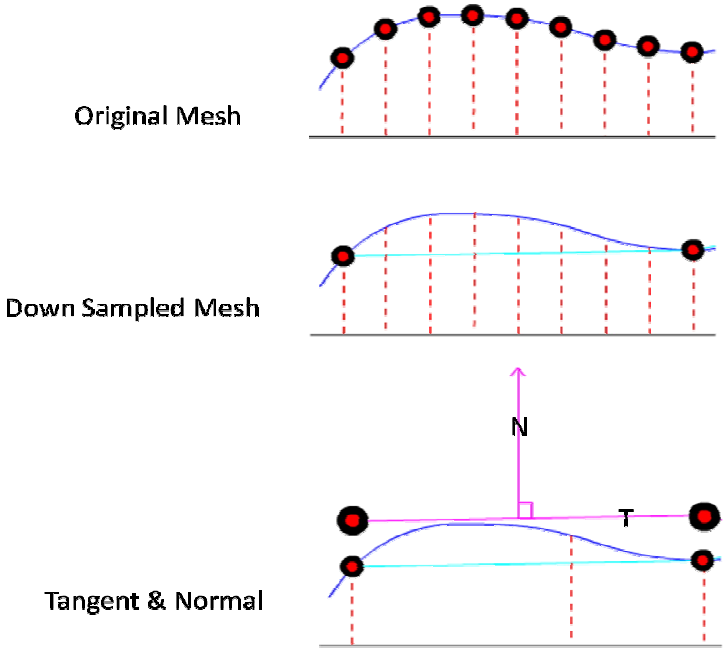


Figure 145 - Creating tangent and normal of the down sampled mesh

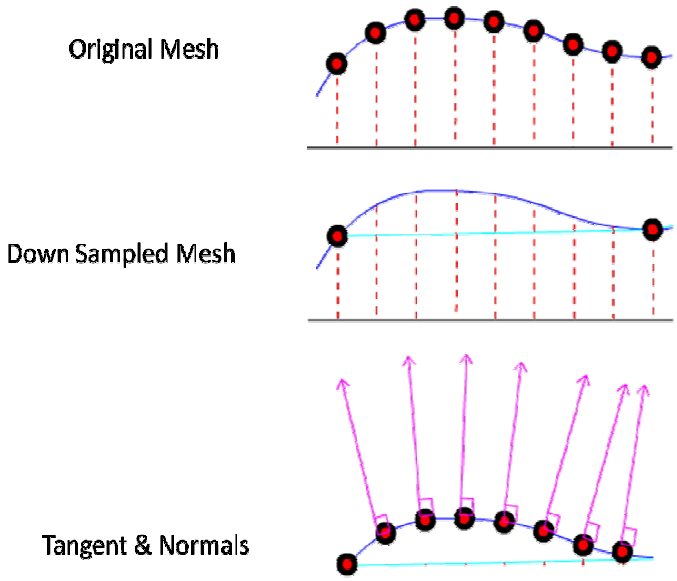


Figure 146 - Normals are converted to tangent space and stored in the normal map

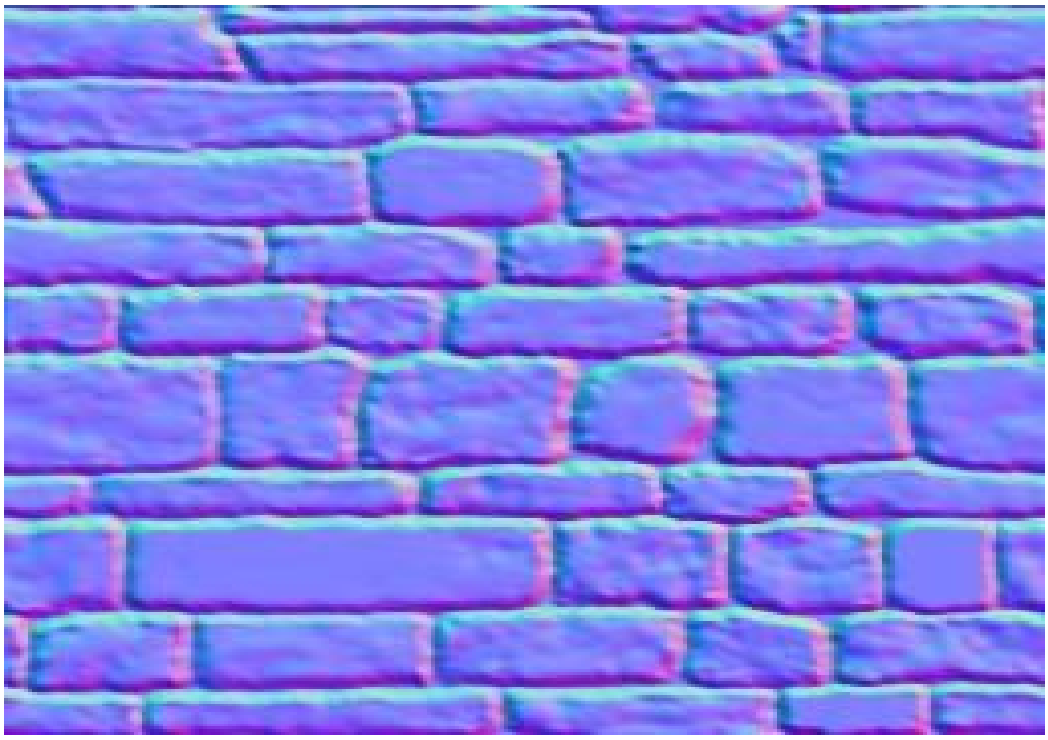


Figure 147 - Normal map of the wall

Figure 148 - Rendering with texture

Figure 149 - Normal map applied (lighting calculations are closer with using Normal Map)

Height mapping

Normal maps increased the realism of rendering but the mesh was not recreated. Thus with height mapping the height offsets of the vertices to the tangent vector are stored and in the shading these offsets are used. In normal map the red, green and blue channels are used for the x, y and z component of the normals; the alpha channel is now used for storing the height offsets (Figure 150-155).

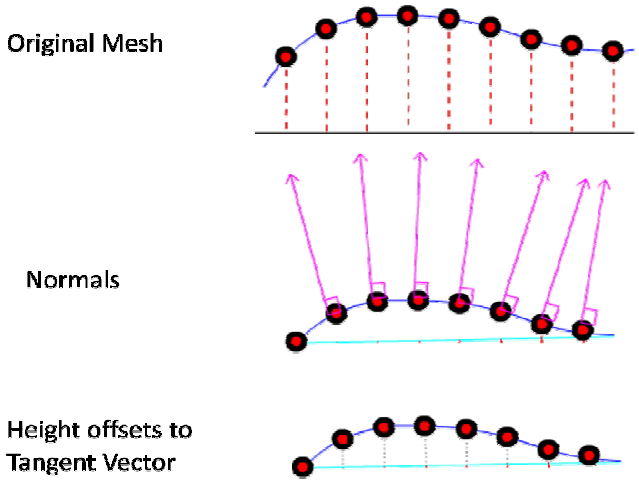


Figure 150 - Calculating the height offsets

Figure 151 - Height map of the wall (Stored in alpha channel)

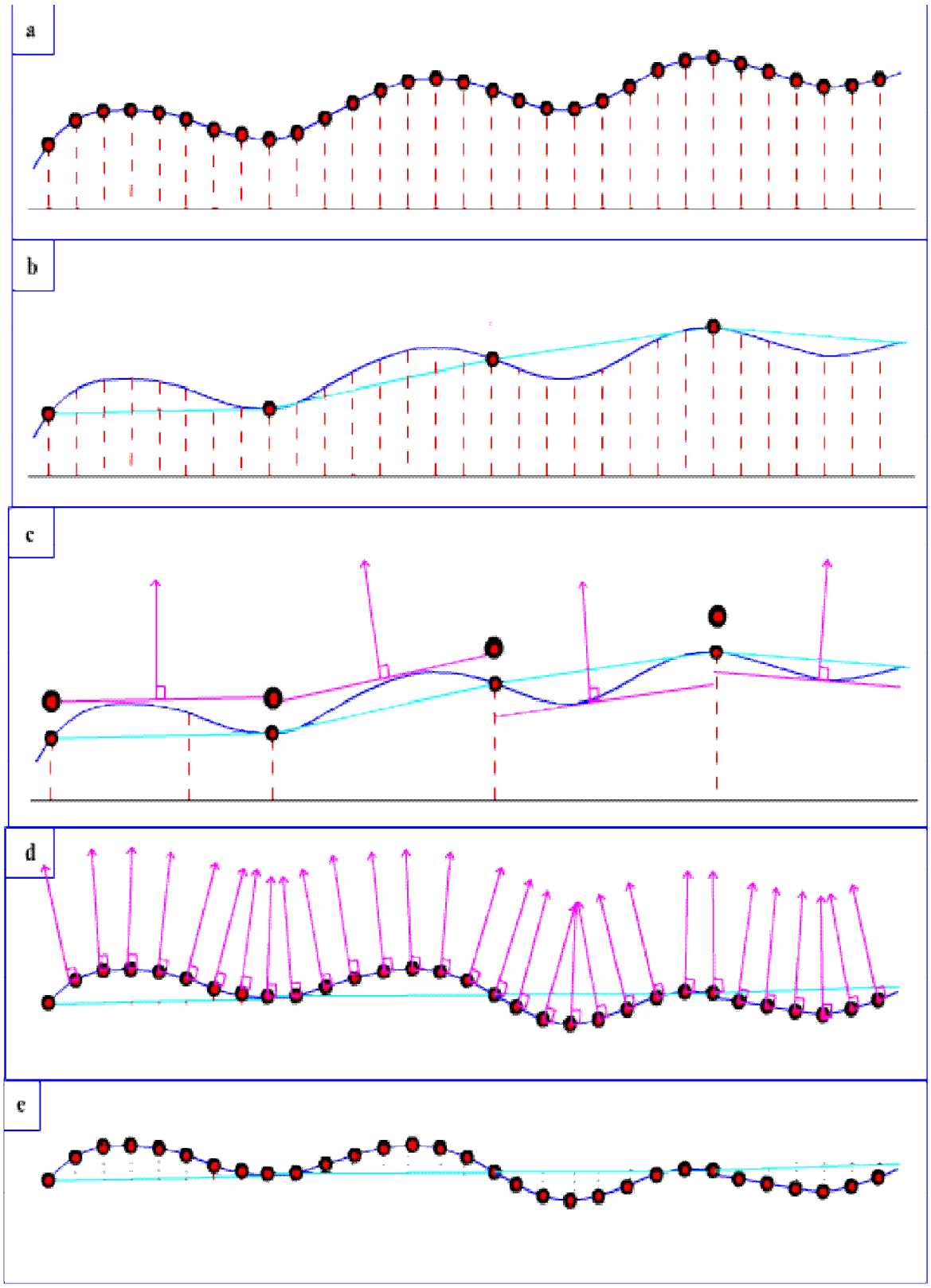


Figure 152 - Calculation of normals and height offsets. note that each of the normals and offsets are transformed to the tangent space

Figure 153 - Wall rendered with just using texture.

Figure 154 - Wall Rendered with using normal maps

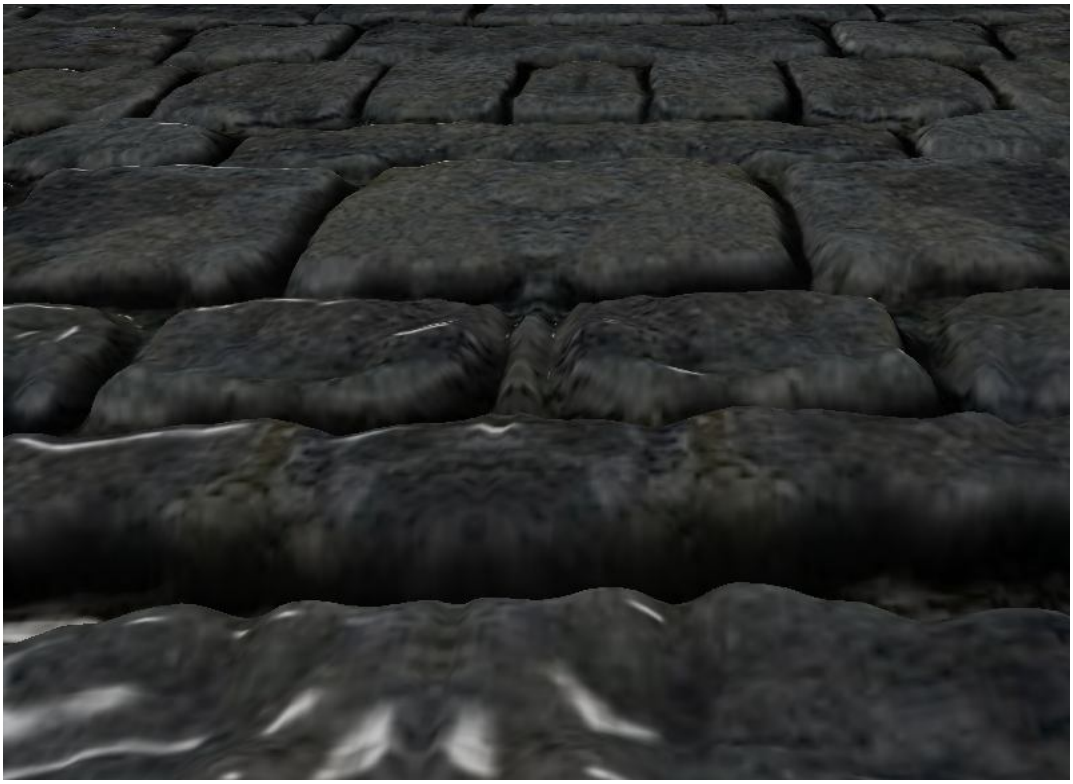


Figure 155 - Wall rendered with normal map and height offsets

Example:

Suppose that a 1024x1024 vertices mesh is constructed and a 1024x1024 pixels normal map is created. The mesh has 1023*1023*2*3= 6279174 vertices for 60Bytes per vertex that approximately equals to 376MB. If the mesh is down sampled with 32 by 32, the new mesh size becomes 31*31*2*3 = 5766 vertices and the size in memory becomes 345.960 bytes and in order to store normal maps 1024*1024*16 bytes approximately 17 MB is needed. Finally the mesh is compressed (376/17 =) 22 times.

Displacement mapping

In order to decrease the GPU bus traffic, meshes and textures are stored in GPU memory. When simulating the soft-body in GPU the output is written to the GPU memory. Thus only outputting the displacements of the vertices rather than the final outputs, these displacements can be used in the rendering stage by offsetting

the vertices' positions at the vertex processing stage. For this purpose, displacement maps are used (Figure 156-157).

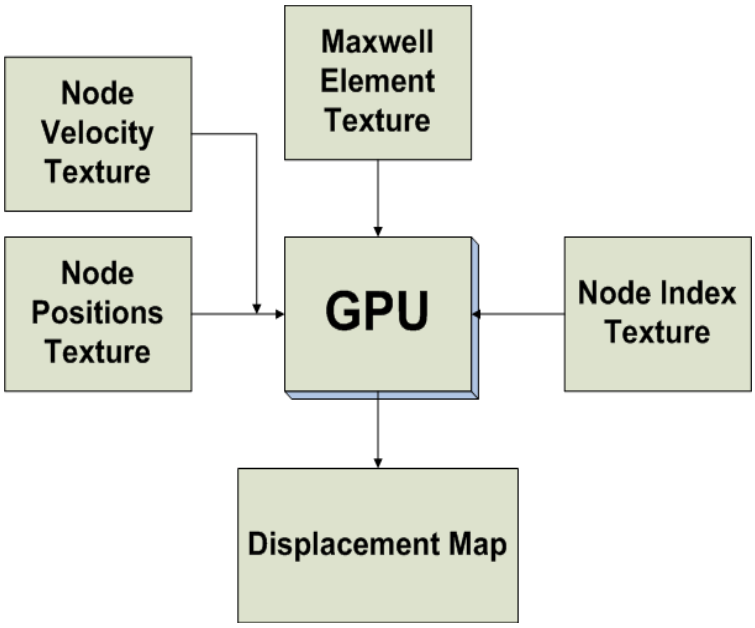


Figure 156 - GPU softbody simulation example

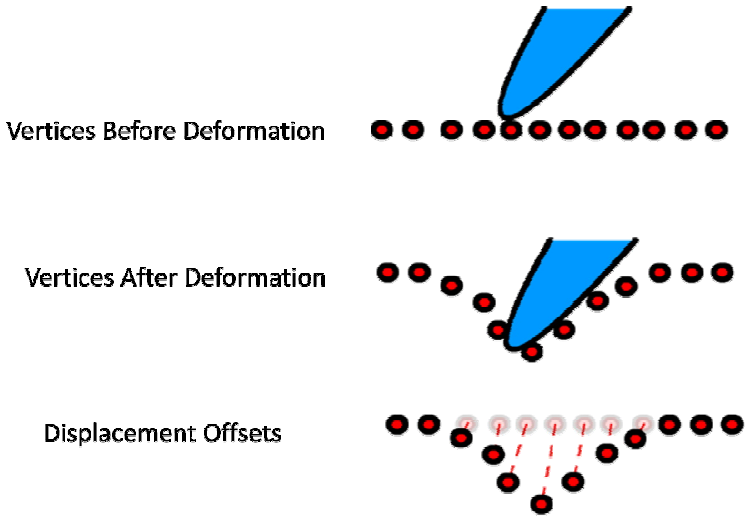


Figure 157 - Displacement mapping is nothing more than storing the change of vertices

3.6 DMIO

DMIO is responsible for managing, updating and utilizing the I/O devices. The IO/Engine architecture can be seen in Figure 158. Drivers are stored in dynamic link libraries. These drivers are linked to the interfaces as seen on Figure 159.

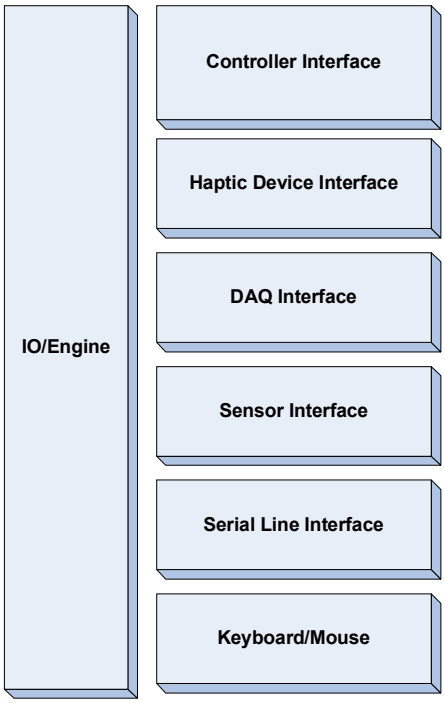


Figure 158 - I/O Engine Architecture

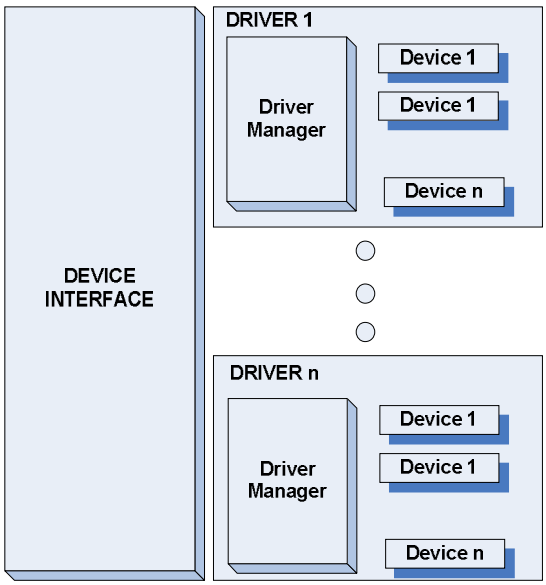


Figure 159 - Driver interface architecture

3.6.1 Controller interface

Controller interface is responsible for managing controllers like joysticks, pedals, wheels, thrusters, gamepads. Six axis analog and 128 button configuration is used as default.

3.6.2 Haptic device interface

Haptic device interface is responsible for managing haptic devices.

3.6.3 DAQ interface

DAQ interface is responsible for managing DAQ boards.

3.6.4 Sensor interface

Sensor interface is responsible for sensor operations.

3.6.5 Serial line interface

Serial Line interface is responsible for creating serial connection with devices.

3.6 DMVision

DMVision is responsible for performing the machine vision tasks. DMVision supports real-time 2D and 3D image processing on both CPU and the GPU. DMVision uses DMImage which is the image processing library. The filter interface defines a base for each image processing filters in the image processing library. Each task is connected to each other with the graph mechanism (Figure 160). The graph is responsible for taking the input image to the connected filter and storing the output image to the appropriate graph element. Figure 161 shows an example of graph and Figure 162 shows an example output screen.

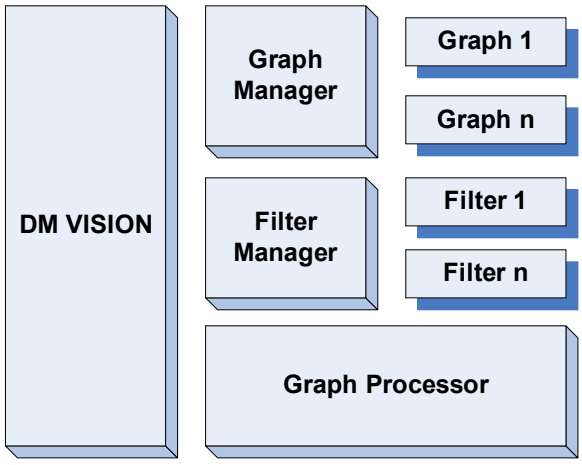


Figure 160 - Machine vision engine architecture

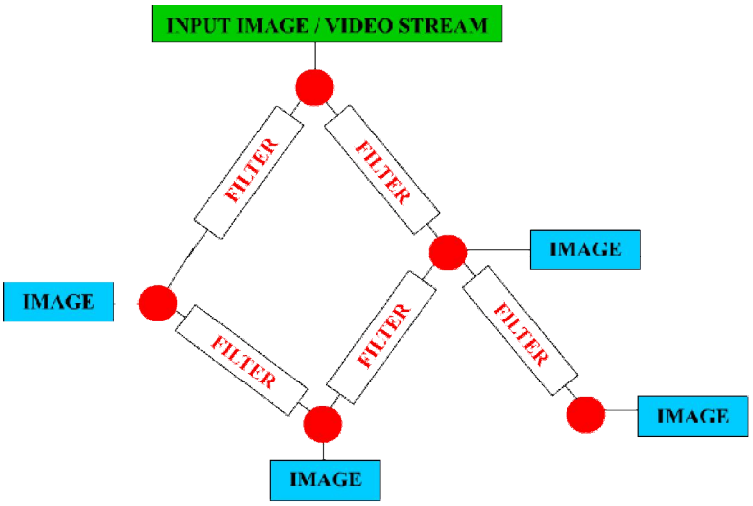


Figure 161 – An example filter graph

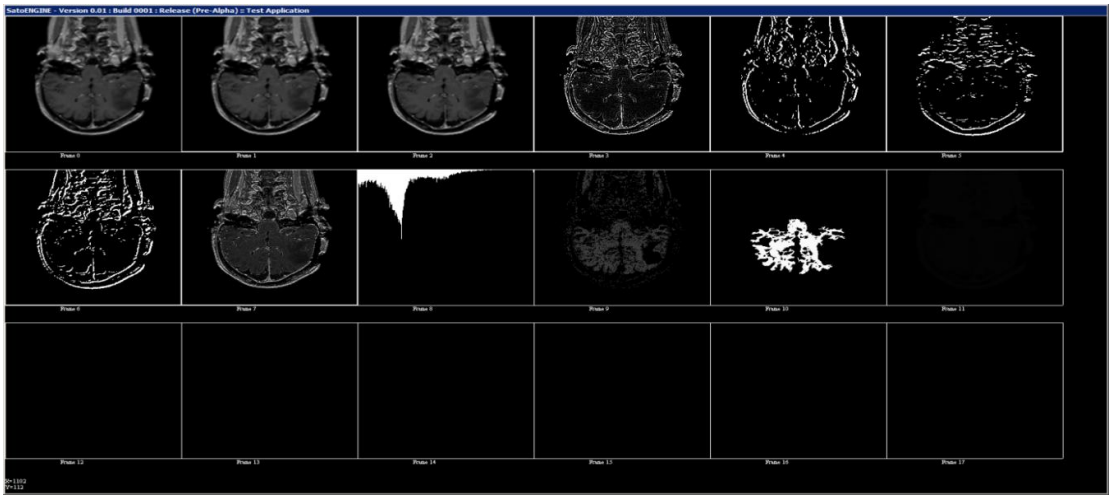


Figure 162 - Graph results stored in image slots(Note that the image slot

3.7 DMAI – AI / ML Engine

DMAI is responsible for performing Machine Learning and AI tasks. Such as Linear discriminant analysis (LDA), k nearest neighborhood (k-NN), Naïve Bayes and support vector machine (SVM).

3.8 DMNetwork – Network Engine

DMNetwork is responsible for creating and managing network connections. DMNetwork supports three widely used networking protocols TCP/IP, UDP and SLIP. Network manager links three managers for each protocol (Figure 163).

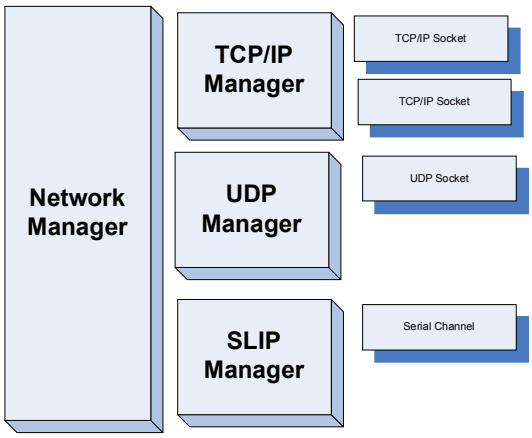


Figure 163 - Architecture of the DMNetwork

3.8.1 TCP/IP

TCP/IP (Transmission Control Protocol / Internet Protocol) is the most common protocol in network systems. Internet uses this protocol as default. TCP is a reliable protocol used for sensitive data transmission.

3.8.2 UDP

UDP (User Datagram protocol) is the other most commonly used protocol in network system. UDP doesn't have handshaking and reliable transfer protocol, it uses a simple transmission model. UDP is useful for sending non-sensitive data like video streams.

3.8.3 SLIP

SLIP (Serial Line Interface Protocol) is designed for communicating with serial channels like RS232, RS485. This protocol supports reliable data and command transfer between endpoints. This protocol is used for communicating with the embedded devices in the system like DSP Boards.

3.9 DMCore

DM Core is the Bottom layer of the Platform which links the platform with the operating system kernel and hardware. It is architecture can be seen on Figure 164.

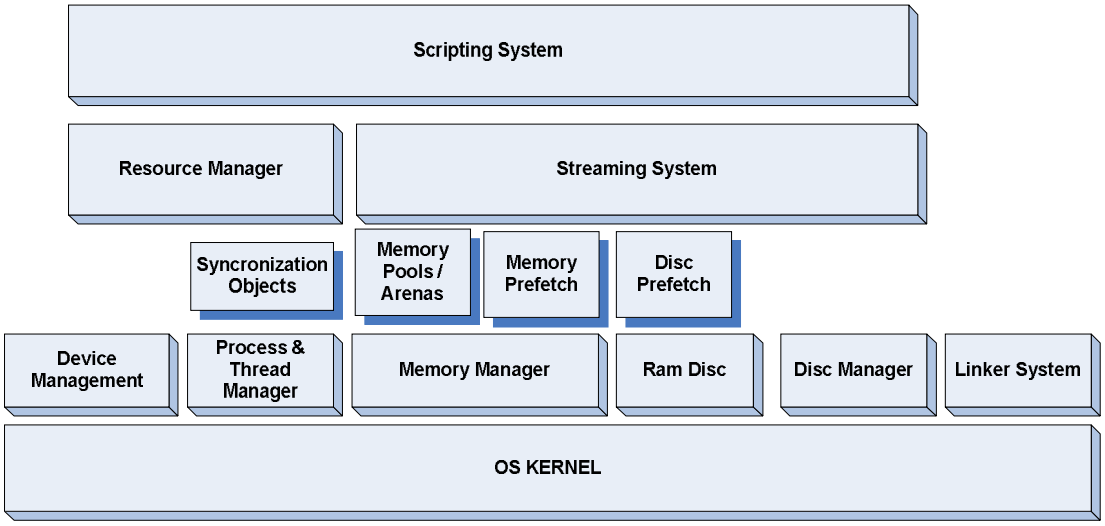


Figure 164 - Architecture of the DMCore

3.9.1 Resource manager

Resource manager is responsible for managing the shared resources by keeping only one instance and sharing the pointers for this instance. For example meshes, materials and textures.

3.9.2 Scripting system

Object properties are stored in a C language like scripts in order to change object properties easily without re-compiling the code.

3.9.3 Linker system

The device drivers, engine and library implementations are stored in dynamic link libraries. The linker system is responsible for linking the best driver for each object.

3.9.4 Memory manager

Memory manager is responsible for managing the platforms memory requirements via memory pools.

3.9.5 Ram disk

Ram Disk is a virtual hard drive that is stored in the system ram in order to increase the access speed.

3.9.6 Streaming system

Streaming system is responsible for streaming the data on disk to the ram and vice versa without interfering the user process.

3.9.7 File system

File system is responsible for managing the disk system with a platform independent manner.

3.9.8 Thread manager

Thread manager is responsible for managing, creating, suspending and synchronizing the thread and processes and managing the shared data between threads.

4. APPLICATIONS

4.1 Cancer Identification

Five pattern recognition algorithms are examined in this thesis. Firstly PCA is applied to the dataset for reducing feature space, secondly linear discriminant analysis is tested, thirdly K-NN, Naïve Bayes algorithms are tested, and finally SVM is examined.

4.1.1 Dataset

Arcene dataset from University of California Irvine Machine Learning Repository is used for testing and benchmarking the pattern recognition algorithms in this document.

Arcene dataset contains cancer versus normal patterns from mass-spectrometric data. This dataset is one of the five datasets used in the NIPS 2003 feature selection challenge. An instance of the dataset can be seen on Figure 165. Dataset features are listed in Table 10.

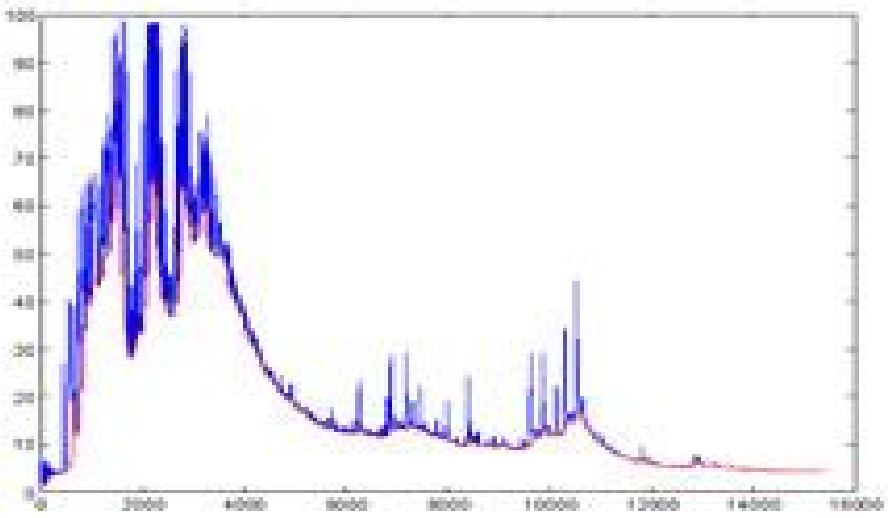


Figure 165 - 0 to 18 Khz spectrum of an instance

Table 10 - Arcene dataset features

Dataset Characteristic	Multivariate
Attribute Characteristic	Real Valued
Number of Instances	700
Number of features	10.000 per Instance
Area	Real Life
Number of Classes	2 (Cancer/Normal)

4.1.2 PCA

Principle component analysis (PCA) is the oldest and best known techniques for multivariate analysis. It was first introduced by Pearson (1901) and developed independently by Hotelling (1933). Like many multivariate methods, it was not widely used until the advent of electronic computers, but it is now well entrenched in virtually every statistical computer package [28].

The central idea of principal component analysis is to reduce the dimensionality of a data set in which there are a large number of interrelated variables, while retaining as much as possible of the variation present in the data set. This reduction is achieved by transforming to a new set of variables, the principal components, which are uncorrelated, and which are ordered so that the first *few* retain most of the variation present in *all* of the original variables [28].

Computation of the principal components reduces to the solution of an eigenvalue-eigenvector problem for a positive-semidefinite symmetric matrix. Thus, the definition and computation of principal components are straightforward but, as will be seen, this apparently simple technique has a wide variety of different applications, as well as a number of different derivations [28].

Visualizing a multivariate dataset is challenging, each feature vector component should be mapped to an axis. If there is more than three vector components, mapping becomes impossible but PCA can produce lower dimensional vector space. As seen on Figure166 dataset is mapped to 2D for visualization. This 2D set is called ‘shadow’. Shadows can be informative for some cases.

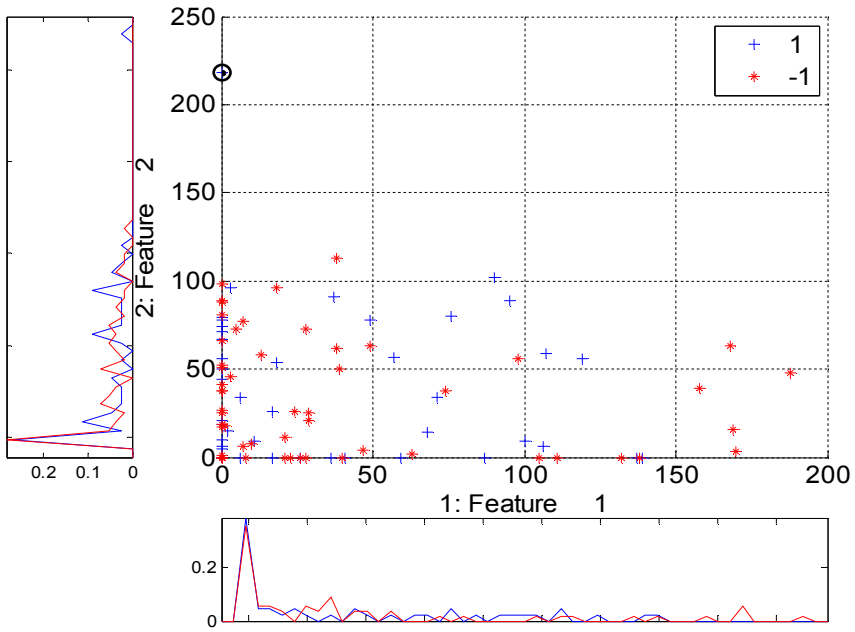


Figure 166 – Shadow of Arcene dataset - mapping Arcene dataset to 2D with PCA

4.1.3 LDA

Fisher's linear discriminant analysis (LDA) is a popular data-analysis tool for studying the relationship between a set of predictors and a categorical response.

LDA finds a linear combination of features which characterize or separate two or more classes of objects or events.

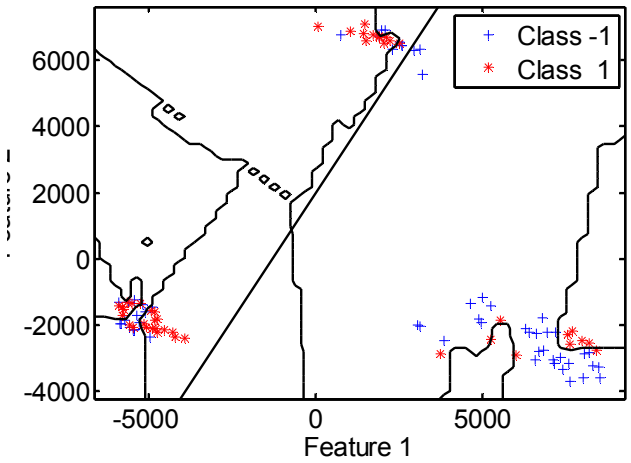


Figure 167 - Example LDA output

4.1.4 KNN

k-nearest neighbor algorithm (*k*-NN) is a classification method based on closest features in the feature space. The object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its *k* nearest neighbors.

If there are infinitely many sample points, then the density estimate converges to the actual density function. The classifier becomes the Bayesian classifier if the large scale sample is provided. But in practice, given a small sample, the Bayesian classifier usually fails in the estimation of the Bayes error, especially in a high-dimensional space, which is called the disaster of dimension. The methods of Parzen and *k*-NN are often used in the case of small sample [28].

4.1.5 Naïve Bayes

Naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem (from Bayesian statistics) with strong (naive) independence assumptions.

In simple terms, a naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature.

4.1.6 SVM

Support vector machine (SVM) takes a set of input data and predicts the output, for each given input, which of two possible classes.

Support vector machine constructs a hyper plane or set of hyper planes in a high- or infinite- dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyper plane that has the largest distance to the nearest training data points of any class.

4.2 Experiment Results

In this part, experiment results will be evaluated. It will be presented that SVM and K-NN give better results than LDA and Naïve Bayes Classifiers.

4.2.1 PCA

After applying PCA, feature size is reduced from 10000 to 5055 maintaining %95 of variance.

4.2.2 LDA

Due to the large feature vector size and non linearity, LDA classifier did not work for this dataset (Table 11).

Table 11 - LDA results for Arcene dataset

	Decisions		
True Labels	-1	1	Totals
1	44	0	44
-1	56	0	56
Totals	100	0	100

4.2.3 KNN

For distance measures Euclidian Distance is selected. And algorithm is run for K=1 to 20

K=1 gave the best results because larger values of *k* reduced the effect of noise on the classification, but made boundaries between classes less distinct (Table 12).

Table 12 - KNN results for Arcene dataset

K = 1			
	Decisions		
True Labels	-1	1	Totals
1	17	27	44
-1	55	1	56
Totals	72	28	100

4.2.4 NAÏVE BAYES

Gaussian distribution is chosen for MLE because mean and variance can be easily calculated from the Spectrum data.

However Naïve Bayes classifier did not work for this dataset (Table 13).

Table 13 - Naive Bayes results for Arcene dataset

	Decisions		
True Labels	-1	1	Totals
1	0	44	44
-1	0	56	56
Totals	0	100	100

4.2.5 SVM

With linear kernel model better results are achieved compared to the radial kernel model. Smaller regulation gave better results (Table 14-15).

Table 14 - SVM (Linear) results for Arcene dataset

Regularization parameter P = 2			
	Decisions		
True Labels	-1	1	Totals
-1	51	5	56
1	7	37	44
Totals	58	42	100

Table 15 - SVM (radial) results for Arcene dataset

Regularization parameter **P = 2**			
	Decisions		
True Labels	-1	1	Totals
-1	49	7	56
1	7	37	44
Totals	56	44	100

4.3 Discussions

As it is expected, SVM has the best results, and linearity makes it fast on runtime decision making. Secondly K-NN gives good results compared to Naïve Bayes and LDA.

Numbers of training samples (100 samples for training) were too small compared to the large number (10.000 features). Therefore, random correlation between input and features may be formed. That could be the reason why Naïve Bayesian did not work for this set.

The nonlinear structure of the mass spectrum data and the large number of features were not suitable for LDA. LDA projects the feature space to lower dimensions but in mass spectrum data, large feature space should be projected to a higher dimension. That's why SVM gave better results on experiments.

4.2 Soft-Tissue Sampling

“Mechanical simulation of viscoelastic materials and assigning a viscoelastic material to the modeled parts in the simulations are difficult task. For the simulations, material model should be well chosen and material coefficients of the chosen models should be known. In order to obtain accurate simulations, hyperelastic characteristics of the viscoelastic materials should be investigated and hyperelastic model should be incorporated in the solutions.

Material models and material model’s coefficients are chosen with the help of mechanical tests/experiments for these situations. The main goal of this application is to optimize material model’s coefficients by using an indenter test setup results and inverse finite element modeling. To achieve this, firstly by using a haptic device and other required equipments an indenter setup was prepared to test the materials mechanically.

Inverse finite element modeling method is used in order to model the materials according to their viscoelastic and hyperelastic characteristics. The model obtained from analysis was optimized by using the results obtained from indenter setup according to experimental test data. By doing this, the correctness of the model chosen by inverse finite element modeling was proved for the tested material and material model coefficients were calculated correctly.” [29]

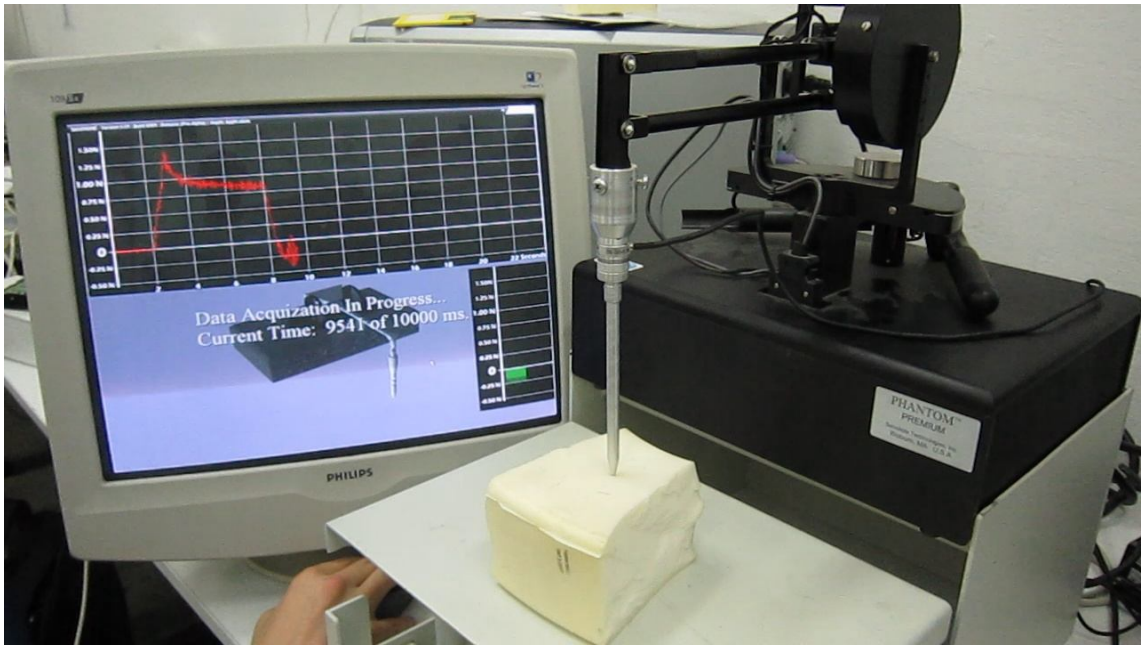


Figure 168 - Data acquisition

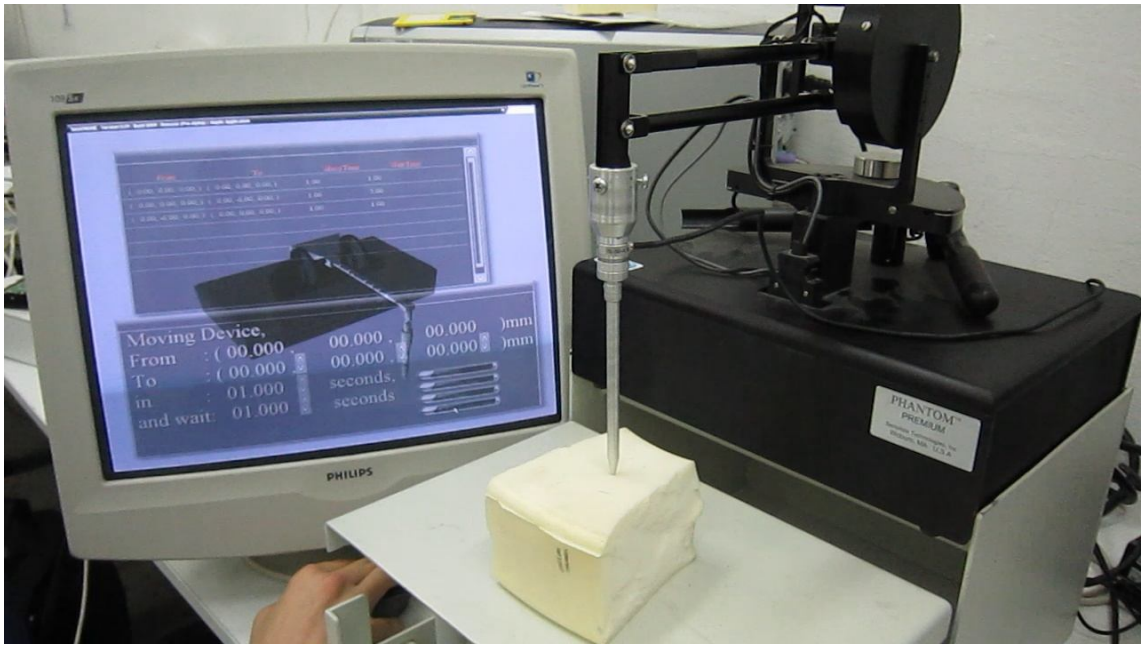


Figure 169 – Path planning

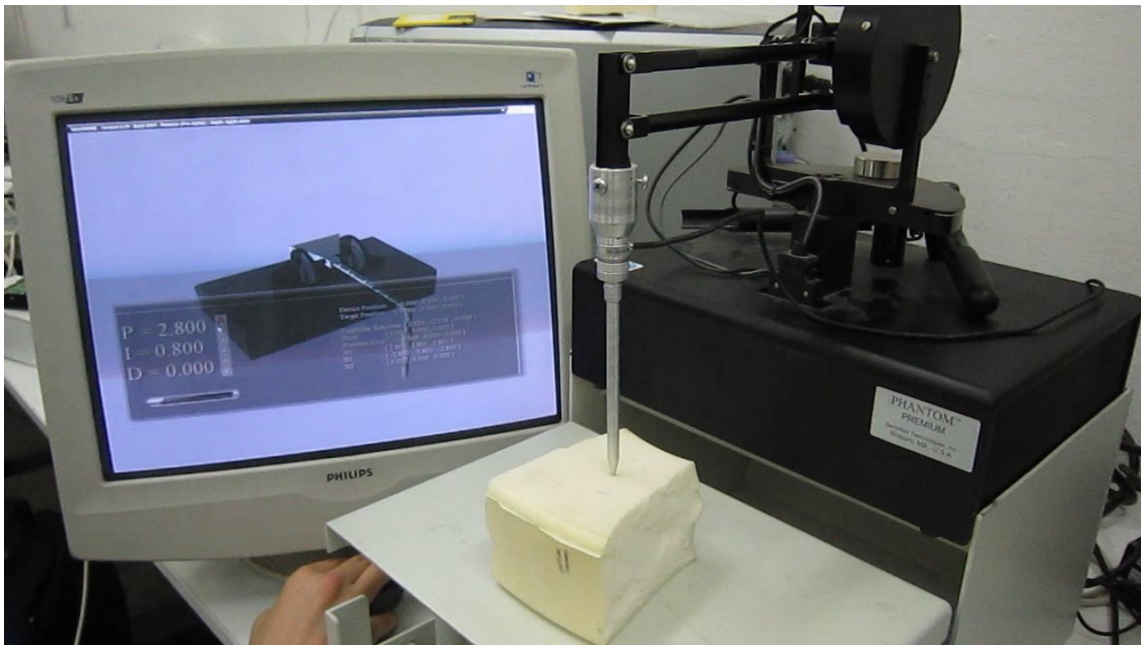


Figure 170 - Tuning the PID controller coefficients

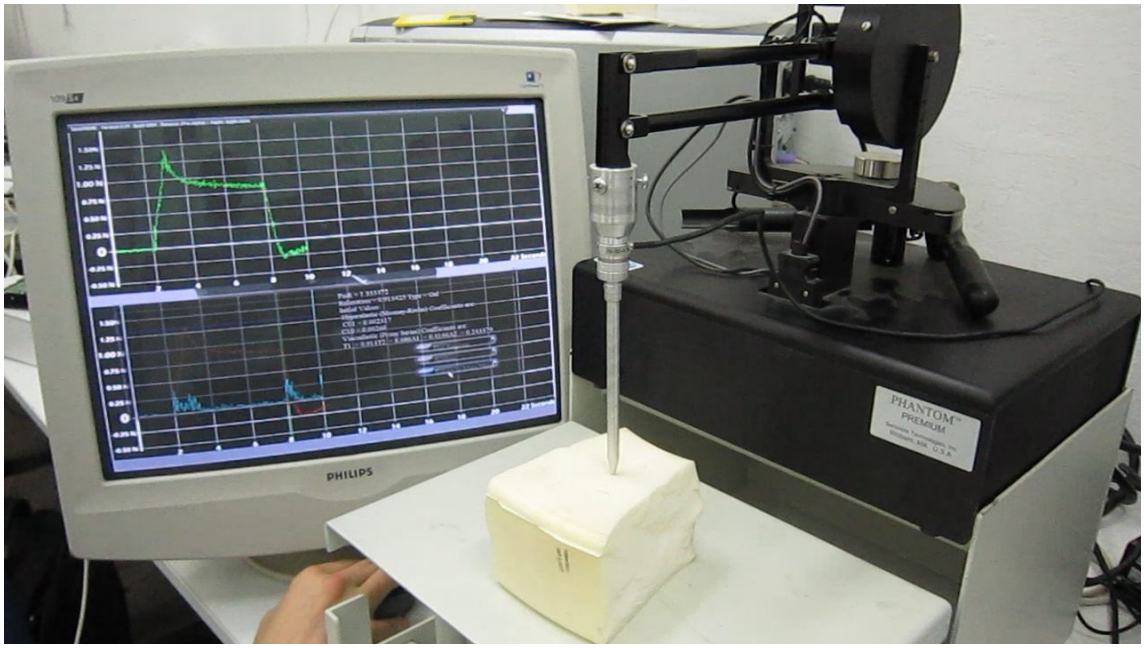


Figure 171 - Analyzing the material properties

4.3 MRI – Haptic Segmentation

In this study, a novel virtual reality based interactive method combined with the application of graphical processing unit (GPU) is proposed for the semi-automatic segmentation of 3D magnetic resonance images (MRI) of the brain. The key point of our approach is to use haptic force feedback guidance for the selection of seed points in a bounded volume with similar intensity and gradient. For the automatic determination of bounded volume of segmentation in real time the parallel computation on GPU is used. Automatic segmentation procedure is applied in this adjustable bounded spherical volume with a variable diameter which is controlled according to the edge map acquired from the gradient map. The haptic force feedback is used in order to guide the user to remain in a volume, where the intensity and gradient change are under a defined threshold range. After each seed point selection, the segmentation algorithm works inside the bounded volume of the ball with an adjusted diameter. Figure 172 – 181.

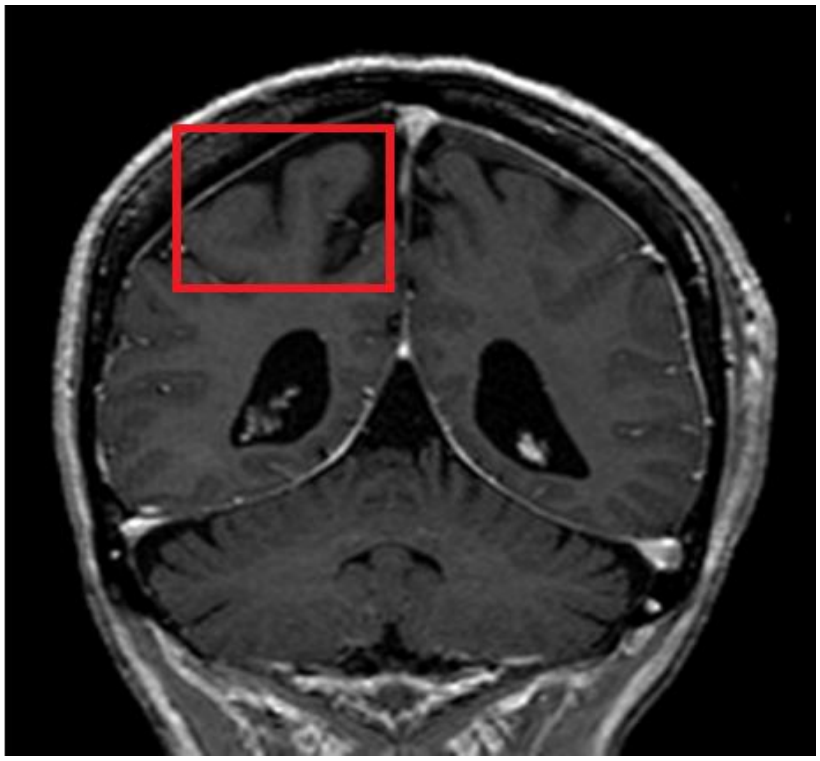


Figure 172 - Section of a brain MRI Slice

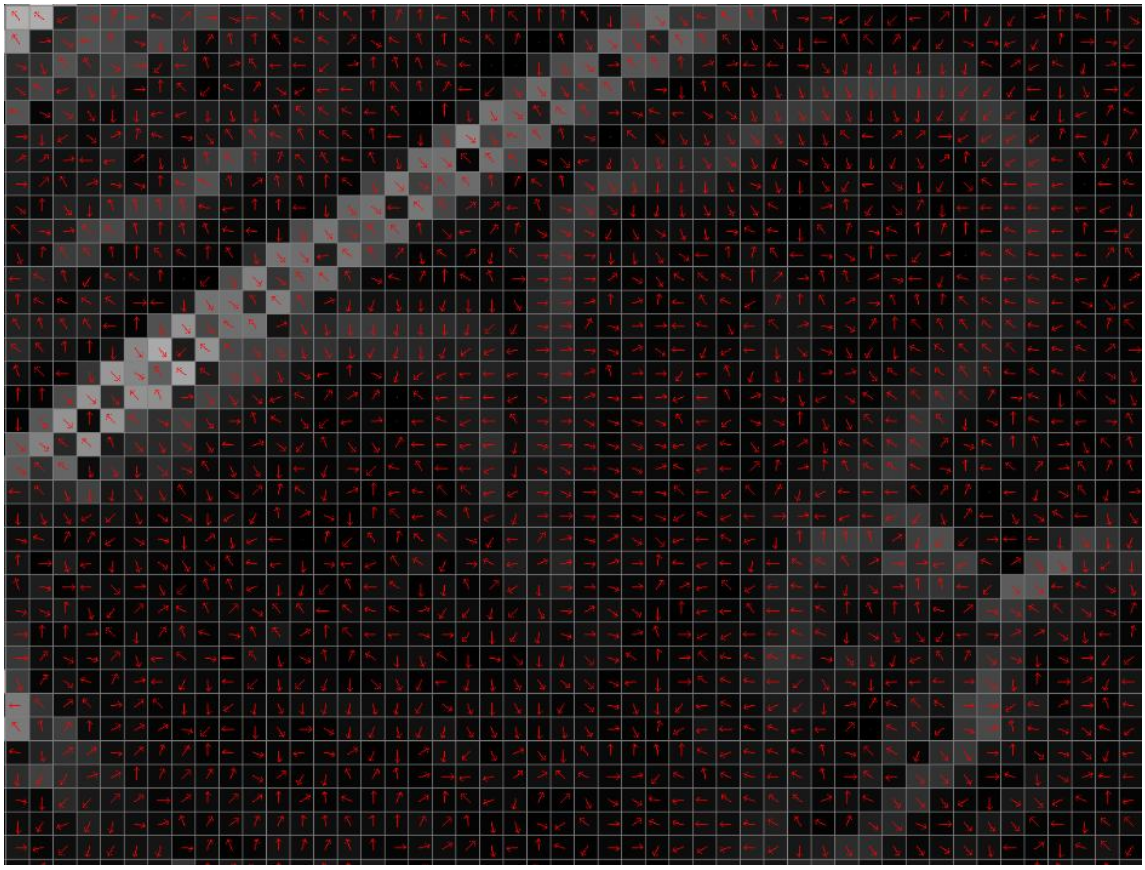


Figure 173 - Gradients of the section marked in previous Figure (Gradient Directions are represented by arrows; gradient magnitudes are represented by gray scale 0=Black white = max)

In preprocessing stage three maps are created, the Intensity map which contains the MRI data, the Gradient map which contains the change of the scalar field and the Edge map which contains the edge information. The ball size is calculated via edge map and the haptic force is calculated by casting rays through the gradient map by using the equation in Figure 178. Where $\vec{F}$ is the force vector. The coefficient a is experimental force regulator constant that is between 0 and 1, $\nabla\vec{m}$ is gradient vector of the MRI data and $\overrightarrow{Rayn}$ is related to the current ray. The term $\overrightarrow{Rayn}\cdot\nabla\vec{m}$ is dot product of current ray and gradient vector and $\left\|\overrightarrow{Rayn}\right\|$ is magnitude of Current Ray that is used for normalization of the current ray.

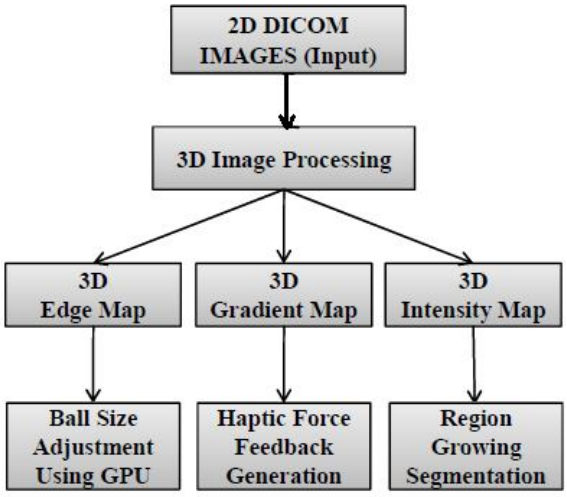


Figure 174 - Pre-Processing of the MRI Data

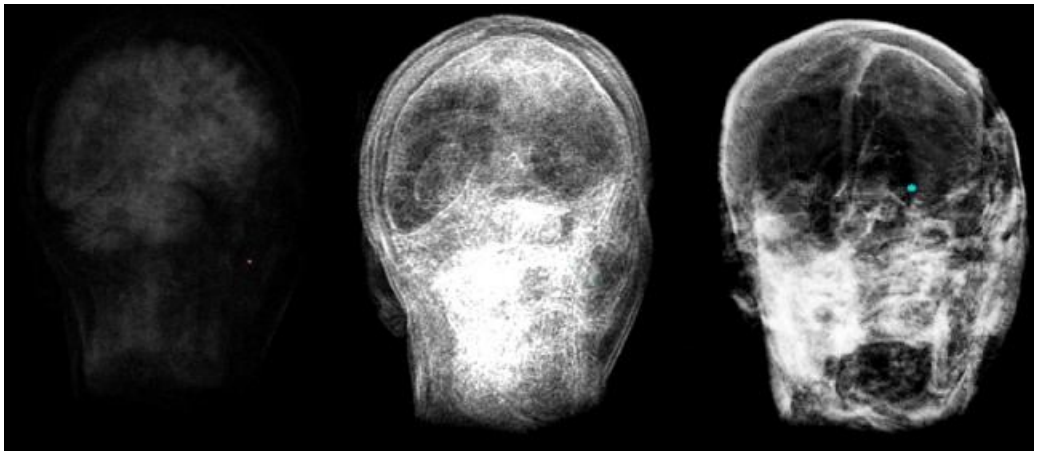


Figure 175 - Intensity map (left); gradient map (Middle); edge map (right)

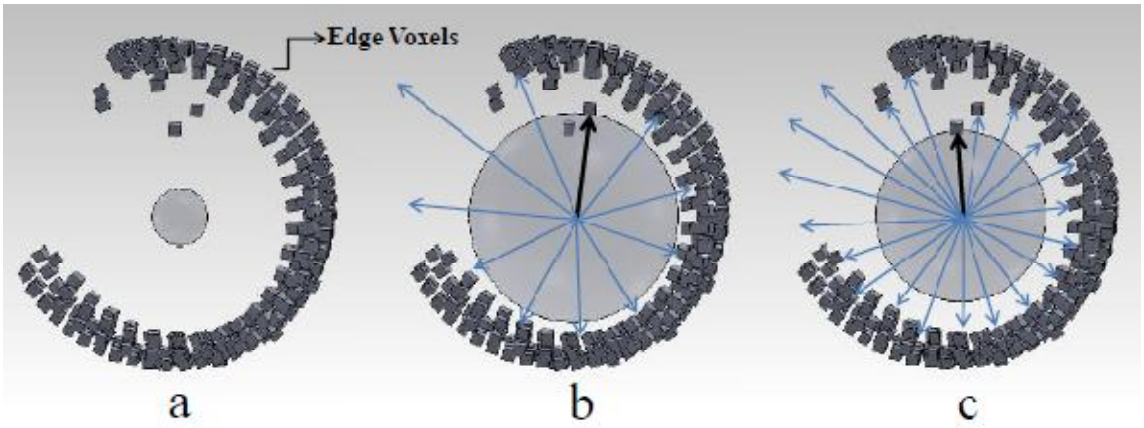


Figure 176 - effect of ray count; initial ball size (a); ball size adjustment using less rays can miss the edge voxels (b) , using more rays increases accuracy.

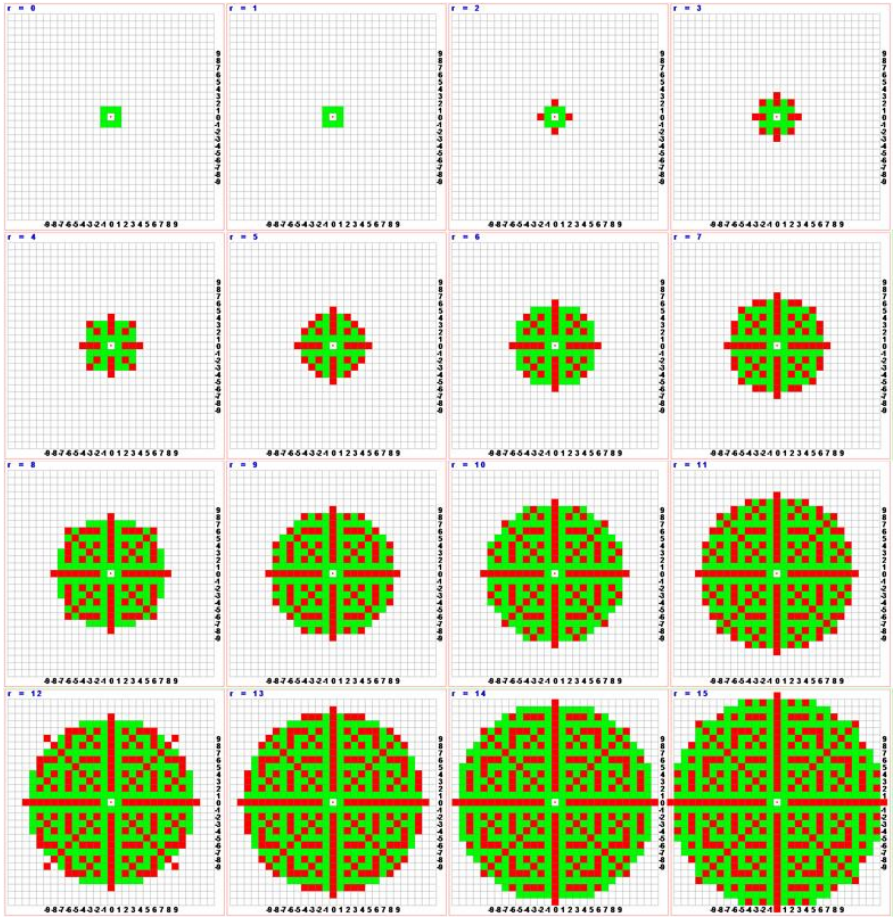


Figure 177 - Active Rays for 2D (Red indicates cancelled, green indicates Ray vectors)

$$\vec{F} = a \times \frac{1}{Number of\ Rays} \times \sum_{n=1}^{Number of\ Rays} \left[\left[\sum_{m=1}^{Number of\ Voxels\ Through\ Ray} \left(\overrightarrow{Rayn} \cdot \nabla \vec{m} \right) \right] \times \frac{\overrightarrow{Rayn}}{\left\| \overrightarrow{Rayn} \right\|} \right]$$

Figure 178 - Force calculation

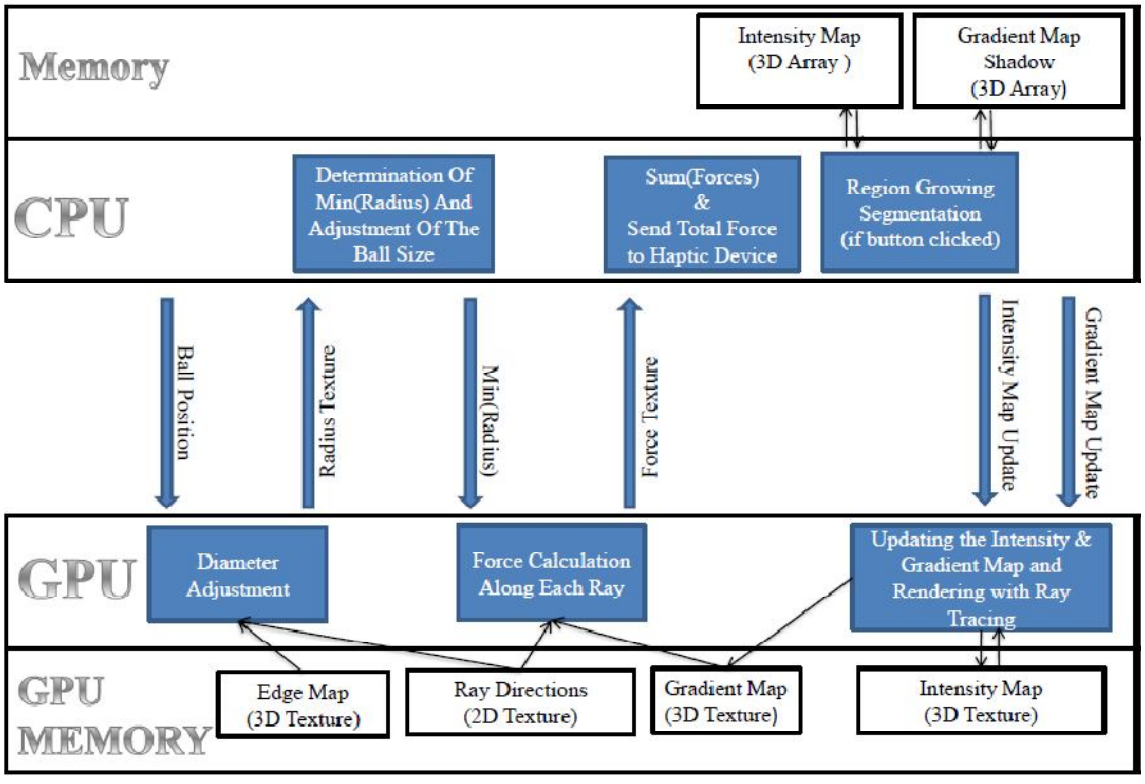


Figure 179 –CPU and GPU states on runtime

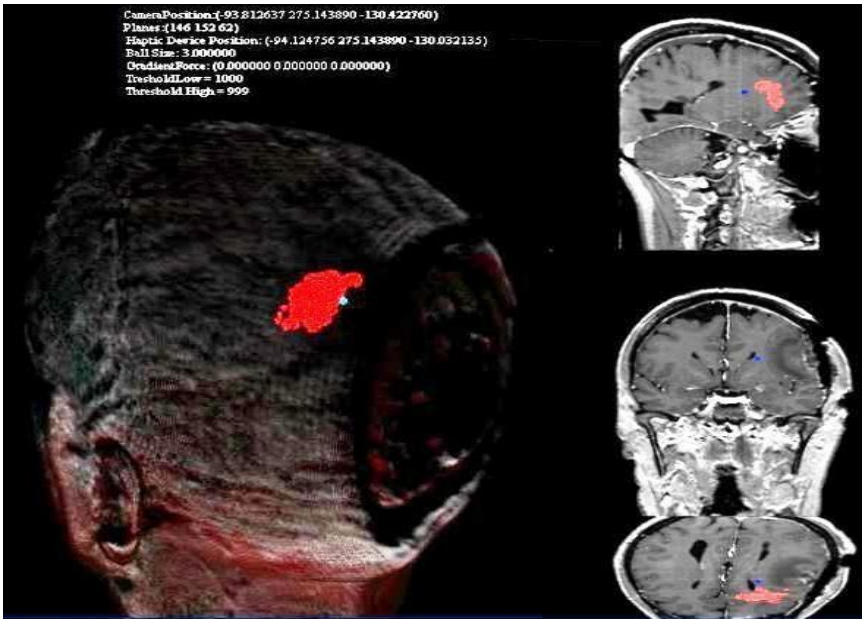


Figure 180 - screenshot from the application

Figure 181 - Haptic Segmentation in action

4.3.2 Results

The proposed segmentation method based on force and visual feedback with the advantage of adjustable bounded volume is not only accurate and effective in narrow spaces near the boundaries of different layers; but also fast in large homogenous spaces since the radius of the ball increases in such regions. Parallel programming on GPU is used for computing gradient change in selected directions which is needed for the self-adjustment of the sphere diameter. Gradient values are used in order to calculate the haptic force on CPU in real-time. In this study, two haptic devices are used, one for getting haptic force feedback and the other one for camera guidance during 3D visualization. Our self tests showed 4 times faster segmentation with the same accuracy compared to manual segmentation method.

5. CONCLUSIONS

Building a surgery robotics system is a very complex and difficult task for both surgeon side network and patient side network. The tasks need to be handled are controlling and preventing collisions of the robots, tools and manipulators attached to the robots, acquiring data from various sensors by DAQ Systems, filtering and processing this acquired data, applying pattern recognition techniques, or feeding the acquired data to fuse the sensors, communicating with the FPGA, DSP boards, acquiring data from imaging devices such as endoscopes, image processing with multi view cameras and registration of the patient, displaying and analyzing the Medical Imaging Data, driving the controllers for surgeon, running a 3D vision system, creating a mathematical model for the force feed-back for the haptic devices, running physics processing for that model, rendering the processed results, driving the haptic devices, creating a network communication, and processing a huge amount of data. However, there is no all in one solution for doing all these tasks in an integrated way.

There are some commercial systems available on the market. Such as ZEUS (Computer Motion), Da Vinci (Intuitive Surgical), in all of these robots the surgeon commands are mimicked by robots. But these systems have some deficiencies and problems. First problem is the absence of haptic feedback which is crucial for surgeons, used to operate with the presence of tissue forces. Moreover, as seen from the example surgery setup in Figure 13, second problem is the network and processing delays. In order to simulate forces and compensate the delays we need to create a mathematical model of the tissue and simulate it in a virtual reality environment. Since the size of the model data is huge, we need to process this data carefully.

The above mentioned problems can be overcome by tools supplied by manufacturers like open haptics academic edition, ghost library, cyclops, sensor software packages and more. In addition to these tools, there are some open source projects like physics engines, rendering engines, machine vision libraries. However, these software packages remain limited for overcoming above problems because of their incompatibilities like use of different coordinate systems, different data types, different endienness, different data packing, different handling

mechanisms, different code space and no shared resource. Due to these problems, there is necessity to create a platform which uses the same coordinate space, same data types, same endienness, the same data types, the same data packing therefore every data, shared between engines and libraries are compatible and reusable. Throughout this thesis, a Platform was tried to build from the scratch and its components were tried to be explained.

The most important property of the Platform is the use of same handling mechanism by all its components to become less confusing. In other words, all the components uses the same resources, for example the mesh used in the physics simulation can be used for rendering or the images captured from video stream can be used as textures in rendering engine without any type conversions. They all use the same resources. The algorithms are stored in the libraries, thus the common used methods are shared, and memory waste is minimized. This approach removed the data and type conversion process. Moreover, every component shares the same data structures, same resources, same handling mechanisms, same coordinate axes, same code space and providing a very flexible and complete architecture. Hence a stable, portable, platform independent, highly scalable, expandable platform was built from the scratch and named DMPlatform.

DMPlatform consists of a low level abstraction layer for platform independency and libraries and engines designed for specific tasks given below.

DMCore is developed as a low level layer for platform and responsible for API abstraction, resource management, memory management, disk management, thread and process management, stream and script management.

DMPhysics is developed as a very detailed physics engine in order to model the tissue information mathematically and to simulate rigid bodies, soft bodies, fluids that are under any kind of force effects.

DMCollide is developed for detecting collisions as fast as possible. The challenges of collision detection process are achieved by constructing smarter collision detection hierarchies and by dividing collision detection process.

DMRenderer is built for photo-realistic rendering and visualization. It is powerful as the commercial rendering engines and supports stereo rendering, detailed lighting physics, high-dynamic range rendering, soft-shadows, tessellation, per pixel lighting techniques, volume rendering techniques and massive rendering techniques.

DMIO is developed for interfacing the platform with various sensor and device interfaces using plug-in architecture driver system. It supports all kinds of controllers, keypads, pedals, joysticks, game-pads, haptic devices, DAQ systems, sensors and more.

DMNetwork is developed for creating and handling common network connections and protocols easily. It supports TCP/IP, world's most used reliable network connection protocol, for the sensitive data and UDP, for stream data such as video stream and SLIP for communicating with the external devices from serial channels.

DMVision is developed in order to be more user friendly and easy to use image processing experience. It does not only supports 2D image processing techniques but also it extended these techniques to 3D image processing which is essential for image processing in volumetric data.

The libraries and engines are developed as plug in architecture, which means the platform is capable of linking the best optimized version of the library or the engine in runtime and this architecture makes it extendable for the newest hardware.

The whole applications and algorithms are developed and integrated as all-in-one approach. The overall platform is easy to use, highly portable, platform abstracted, open architecture, overridable, highly scalable, extendable and optimized. The platform supports multi core processing & synchronization which is essential for today's multi-core CPUs and supports symmetric processing with the GPU which means utilizing the GPU with CPU.

To demonstrate the above mentioned features of the platform and its capabilties, some applications like soft tissue sampling and haptic segmentation are provided in this research. These applications proved how the platform components are

compatible with each other. To specify, most well known pattern recognition algorithms are compared for the cancer detection for real-time usage by the platform and it is completed easily. In addition to these, a soft-tissue sampling application is developed which is used for calculating the finite element coefficients by applying the inverse finite element method, with using these coefficient the mathematical model and the Maxwell grids are developed more accurately. Last but not least, a haptic driven semi automatic MRI segmentation method is developed, in order to speed-up and ease the tumor marking process for our upcoming brain surgery robot.

In the near future, with the experience gained from the upcoming brain surgery robot the platform will be extended as needed. A distributed processing architecture is planning to be added to the platform in order to distribute the processing load to other computers which will allow us to model the meshes with higher resolution that is directly effects the simulation accuracy.

REFERENCES

[1] KWOH, Y. S., HOU, J., JONCKHEERE, E. A. and HAYALL, S. A., Robot with improved absolute positioning accuracy for CT guided stereotactic brain surgery. IEEE Trans. Biomed. Eng, vol. 35, no.2, p. 153–161, 1988.

[2] NATHOO, N., CAVUŞOĞLU, M.C., VOGELBAUM, M.A., BARNETT, G.H., In touch with robotics: neurosurgery for the future, Neurosurgery, vol.56, no.3, p. 421-433, 2005.

[3] HILLS, J. W., JENSEN, J. F., Telepresence technology in medicine: principles and applications, Proceedings of the IEEE, vol.86, no.3, p.569-580, 1998.

[4] FALK, V., MCLOUGHIN, J., GUTHART, G., SALISBURY Jr, J. K., WALTHER, T., GUMMERT, J., and MOHR, F.W, Dexterity enhancement in endoscopic surgery by a computer controlled mechanical wrist, Minimally Invasive Theraphy and Allied Technologies, vol.8, no.4, p.235-242, 1999.

[5] Taylor, R., Jensen, P., Whitcomb, L., Barnes, A., Kumar, R., Stoianovici, D., Gupta, P., Wang, Z. X., DeJuan, E., Kavoussi, L., A steady-hand robotic system for microsurgical augmentation. Medical Image Computing and Computer-Assisted Intervention - MICCAI'99. Taylor, C. and Colchester, A. (eds.), Springer-Verlag, Cambridge. pp.1031-1041, 1999.

[6] Dewan, M., Marayong, P., Okamura, A. M., Hager, G. D., Vision-based assistance for ophthalmic microsurgery. Medical Image Computing and Computer-Assisted Intervention - MICCAI 2004. 7th International Conference Proceedings, vol. 2.Barillot, C., Haynor, D. R., Hellier, P. (eds), Springer-Verlag, Saint-Malo, France, pp. 49-57, 2004.

[7] HEILBRUN, M.P., MCDONALD, J.D., The future of image-guided surgery, Clin. Neurosurg., vol. 46, p. 89-101, 2000.

[8] APUZZO, MLJ, In the realm of ideas: The advent of advanced surgery of the human cerebrum and neurosurgical education, Acta Neurochir Suppl., vol. 69, p. 145-150, 1997.

[9] CAVUSOGLU, M.C., SHERMAN, A., TENDICK, F., Design of bilateral teleoperation controllers for haptic exploration and telemanipulation of soft environments, IEEE Transactions on Robotics and Automation, vol. 18, no.4, p.641–647, 2002.

[10] GRAY, B.L., FEARING, R.S., A surface micromachined microtactile sensor array, IEEE International Conference on Robotics and Automation, Minneapolis, vol.1, p.1-6, 1996.

[11] RUURDA, J. P., BROEDERS, I. A., PULLES, B., KAPPELHOF, F. M., WERKEN, C. van der, Manual robot assisted endoscopic suturing: Time-action analysis in an experimental model, Surgical Endoscopy, vol. 18, no. 8, p. 1249-1252, 2004.

[12] XIN, H., ZELEK, J. S., CARNAHAN, H., Laparoscopic surgery, perceptual limitations and force: A review. First Canadian Student Conference on Biomedical Computing, Ontario, Canada, 2006.

[13] NEUBAUER, A., WOLFSBERGER, S., FORSTER, M.T., MROZ, L., WEGENKITTL, R., BUHLER, K., Advanced virtual endoscopic pituitary surgery, IEEE Transactions on Visualization and Computer Graphics, vol.11, no.5, p.497-507, 2005.

[14] NEUMANN, P., SIEBERT, D., SCHULZ, A., FAULKNER, G., TOLXDOR, T., Using virtual reality techniques in maxillofacial surgery planning, Virtual Reality, vol.4, no.3, p.213-222, 1999.

[15] MORRIS D., Sewell C., Blevins N., Barbagli F., and Salisbury K., A collaborative virtual environment for the simulation of temporal bone surgery. Medical Image Computing and Computer-Assisted Intervention - 7th International Conference. Proceedings, MICCAI 2004. Barillot, D. R.; Haynor, and P.; Hellier, (eds.), vol. 2, p. 319-327, Springer-Verlag, Saint-Malo, France, 2004.

[16] SRIMATHVEERAVALLI, G. and THENKURUSSI, K.. Motor skill training assistance using haptic attributes, First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. IEEE Comput. Soc., World Haptics Conference, Pisa, Italy, p. 452_457, 2005.

[17] BASDOGAN C., HO C. H., SRINIVASAN M. A., SMALL S. D., and DAWSON S. L., Force interactions in laparoscopic simulations: haptic rendering of soft tissues, Stud Health Technol.Inform., vol. 50, p. 385-91, 1998.

[18] DE S., MANIVANNAN M., KIM J., SRINIVASAN M. A., and RATTNER D., Multimodal simulation of laparoscopic heller myotomy using a meshless technique. Stud Health Technology Inform, vol 85, p. 127-32, 2002.

[19] HUTTER R., SCHMITT K. U., and NIEDERER P. Mechanical modeling of soft biological tissues for application in virtual reality based laparoscopy simulators. Technology and Health Care, vol 8(1), p. 15-24, 2000.

[20] Ayache, N., Cotin, S., and Delingette, H., Surgery simulation with visual and haptic feedback, Proceedings of Eighth International Symposium on Robotics Research, Shirai Y and Hirose S., (eds.), Springer-Verlag, London, pp. 311-316, 1998.

[21] NEALEN, A., MÜLLER, M., KEISER, R., BOXERMAN, E. and CARLSON, M., Physically based deformable models in computer graphics. Computer Graphics Forum, Vol. 25, issue 4, p. 809-836, 2006.

[22] QUINLAN S., Efficient distance computation between non-convex objects, Proc. IEEE Int. Conf. On Robotics and Automation, p. 3324–3329, 1994.

[23] ZACGMANN, G., Rapid collision detection by dynamically aligned dop-trees, Proceedings of the IEEE Virtual Reality Annual International Symposium, pp. 90–97, March 1998.

[24] LIN, M. C., and GOTTSCHALK, Collision detection between geometric models: a survey, 1998.

[25] JIMENEZ, P., THOMAS, F., and TORRAS, C., 3D Collision Detection: A Survey, Computers and Graphics, vol 25(2), p. 269-285, 2001.

[26] SPICER, MA, APUZZO, MLJ, Virtual reality surgery: Neurosurgery and the contemporary landscape. Neurosurgery, vol 52 p 489–497, 2003.

[27] Tendick, F, DOWNES, M, GOKTEKIN, T, CAVUSOGLU, MC, FEYGIN, D, WU, X, EYAL, R, HEGARTY, M, WAY, LW: A virtual environment test bed for training laparoscopic surgical skills, Presence Teleoperators and Virtual Environments, vol 9, p 236–255, 2000.

[28] DUDA, Richard O., HART, Peter E. and STORK, David G., Pattern Classification 2nd Edition, Wiley - Interscience, 654p, 2000.

[29] TABAKÇI, Ali Can, Mechanical properties identification of viscoelastic / hyperelastic materials based on experimental data, M.Sc. Thesis, METU, Ankara, 133p, 2010.

[30] ERICSSON, Christer, Real time Collision detection, Morgan Kaufmann, 632p, 2005.

[31] BERGEN, Gino van den, Collision Detection in Interactive 3D Environments, Morgan Kaufmann, 277p, 2003

[32] EBERLY, David H., Game Physics 2nd Edition, Morgan Kaufmann, 944p 2003.

[33] ERLEBEN, Kenny, SPORRING, Jon, HENRIKSEN, Knud, and DOHLMANN, Henrik, Physics-Based Animation, Charles River Media, 817p, 2005.

[34] PHARR, Matt, and HUMPHREYS, Greg, Physically Based Rendering: From Theory to Implementation, Morgan Kaufmann, 2004

[35] MOLLER, Tomas A., HAINES, Eric and HOFFMAN, Naty, Real-Time Rendering, AK Peters, 1045p, 2008

APPENDIX

APPENDIX A – ABBREVIATIONS

Scalar Multiplication:

$$S = a * b$$

Vectors

Vector:

$$\vec{\boldsymbol{v}} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ v_w \end{bmatrix}$$

Unit Vector:

$$\hat{\boldsymbol{v}} = \frac{\vec{\boldsymbol{v}}}{|\vec{\boldsymbol{v}}|}$$

Component Product

$$\vec{\boldsymbol{v}} = \vec{\boldsymbol{v}}_{\boldsymbol{1}} \odot \vec{\boldsymbol{v}}_{\boldsymbol{2}}$$

$$\begin{bmatrix} v_x \\ v_y \\ v_z \\ v_w \end{bmatrix} = \begin{bmatrix} v_{1x} \\ v_{1y} \\ v_{1z} \\ v_{1w} \end{bmatrix} \odot \begin{bmatrix} v_{2x} \\ v_{2y} \\ v_{2z} \\ v_{2w} \end{bmatrix} = \begin{bmatrix} v_{1x} * v_{2x} \\ v_{1y} * v_{2y} \\ v_{1z} * v_{2z} \\ v_{1w} * v_{2w} \end{bmatrix}$$

Dot (Scalar) Product

$$s = \vec{\boldsymbol{v}}_{\boldsymbol{1}} \odot \vec{\boldsymbol{v}}_{\boldsymbol{2}}$$

$$s = \begin{bmatrix} v_{1x} \\ v_{1y} \\ v_{1z} \\ v_{1w} \end{bmatrix} \odot \begin{bmatrix} v_{2x} \\ v_{2y} \\ v_{2z} \\ v_{2w} \end{bmatrix} = v_{1x} * v_{2x} + v_{1y} * v_{2y} + v_{1z} * v_{2z} + v_{1w} * v_{2w}$$

Vectoral Product

$$\vec{\boldsymbol{v}} = \vec{\boldsymbol{v}}_{\boldsymbol{1}} \otimes \vec{\boldsymbol{v}}_{2}$$

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} v_{1x} \\ v_{1y} \\ v_{1z} \end{bmatrix} \otimes \begin{bmatrix} v_{2x} \\ v_{2y} \\ v_{2z} \end{bmatrix} = \begin{bmatrix} v_{1y} * v_{2z} - v_{1z} * v_{2y} \\ v_{1z} * v_{2x} - v_{1x} * v_{2z} \\ v_{1x} * v_{2y} - v_{1y} * v_{2x} \end{bmatrix}$$

Matrix

Matrix

Bold and Uppercase

$$\mathbf{M}$$

Column Matrix:

$$\mathbf{M} = \begin{bmatrix} v_{1x} & v_{2x} & v_{3x} & v_{4x} \\ v_{1y} & v_{2y} & v_{3y} & v_{4y} \\ v_{1z} & v_{2z} & v_{3z} & v_{4z} \\ v_{1w} & v_{2w} & v_{3w} & v_{4w} \end{bmatrix} = [\, \overrightarrow{\boldsymbol{v_1}} \quad \overrightarrow{\boldsymbol{v_2}} \quad \overrightarrow{\boldsymbol{v_3}} \quad \overrightarrow{\boldsymbol{v_4}} \,]$$

Special Vectors:

Uppercase,Bold and italic:

$$\boldsymbol{F} = Force$$

Special Constants:

Italic

$$k_{stiff}$$

Convolution:

$$y(t) = x(t) \circledast u(t)$$

APPENDIX B - TRANSFORMATION EQUATIONS

Homogeneous Coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{(B.1)}$$

Translation

$$T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(B.2)}$$

$$\acute{P} = T.P \quad \text{(B.3)}$$

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -T_x \\ 0 & 1 & 0 & -T_y \\ 0 & 0 & 1 & -T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(B.4)}$$

$$P = \acute{P}.T^{-1} \quad \text{(B.5)}$$

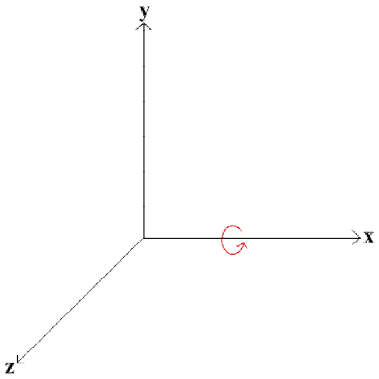
Scaling

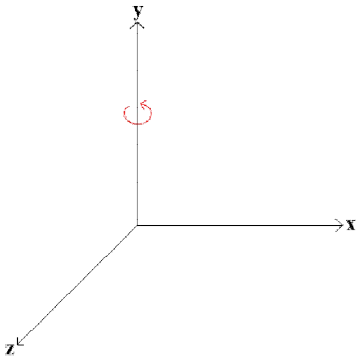
$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(B.6)}$$

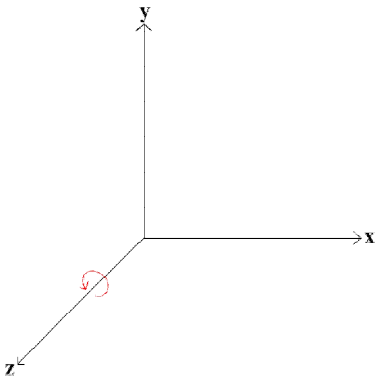
$$S^{-1} = \begin{bmatrix} \frac{1}{S_x} & 0 & 0 & 0 \\ 0 & \frac{1}{S_y} & 0 & 0 \\ 0 & 0 & \frac{1}{S_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(B.7)}$$

$$S' = 0 \quad \text{(B.8)}$$

Rotation







Rotation in Z Axis:

$$x' = xcos\theta - ysin\theta \quad \text{(B.9)}$$

$$y' = xsin\theta + ycos\theta \quad \text{(B.10)}$$

$$z' = z \quad \text{(B.11)}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{(B.12)}$$

$$R_z(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(B.13)}$$

$$P' = R_z(\theta).P \quad \text{(B.14)}$$

$$R_z{}^{-1}(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 & 0 \\ -sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(B.15)}$$

$$P = P'.R_z{}^{-1}(\theta) \quad \text{(B.16)}$$

Rotation in X axis:

$$y' = ycos\theta - zsin\theta \tag{B.17}$$

$$z' = ysin\theta + zcos\theta \tag{B.18}$$

$$x' = x \tag{B.19}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{B.20}$$

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.21}$$

$$P' = R_x(\theta).P \tag{B.22}$$

$$R_x{}^{-1}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & sin\theta & 0 \\ 0 & -sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(B.23)}$$

$$P = P'.R_x{}^{-1}(\theta) \quad \text{(B.24)}$$

Rotation in Y axis:

$$z' = zcos\theta - xsin\theta \quad \text{(B.25)}$$

$$x' = zsin\theta + xcos\theta \quad \text{(B.26)}$$

$$y' = y \quad \text{(B.27)}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{(B.28)}$$

$$R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(B.29)}$$

$$P' = R_y(\theta).P \quad \text{(B.30)}$$

$$R_y{}^{-1}(\theta) = \begin{bmatrix} cos\theta & 0 & -sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(B.31)}$$

$$P = P'.R_y{}^{-1}(\theta) \quad \text{(B.32)}$$

APPENDIX C - PHYSICS EQUATIONS

Equations of Motion

Velocity:

$$v = \frac{dx}{dt} = \frac{\Delta x}{\Delta t} \quad \text{(C.1)}$$

Where, v is the velocity, x is the position, t is the time and Δt is the time delta.

$$v = \lim_{t \to 0} \frac{\Delta x}{\Delta t} = \frac{x - x_0}{\Delta t} \quad \text{(C.2)}$$

Where v is the velocity, x is the current position, x_0 is the previous position and Δt is the time delta.

$$\vec{v} = \dot{x} = s.\vec{d} \quad \text{(C.3)}$$

Where, v is the velocity vector, s is the scalar speed and d is the direction vector.

Acceleration:

$$a = \frac{d^2x}{dt^2} = \frac{dv}{dt} \qquad \text{(C.4)}$$

Where, a is the acceleration, x is the position, v is the velocity, t is the time and Δt is the time delta.

Momentum

$$J = mv \qquad \text{(C.5)}$$

Newton 1:

An Object Continues with a constant velocity unless a force acts upon it. (momentum).

Ex : Drag Forces.

Newton 2:

A force acting on an object produces acceleration which is proportional with objects mass

$$F = m\,a \qquad \text{(C.6)}$$

Where, F is the Force acting on the body, m is the mass of the body and a is the acceleration produced by F.

$$F = ma = m\dot{v} = m\ddot{x} \tag{C.7}$$

Gravitation

$$F = G\frac{m_1 m_2}{r^2} \tag{C.8}$$

Where F is the Gravitation force, G is the universal gravitation constant which is XXX, r is the distance of objects m_1 , m_2 is the object masses of object 1 and object 2.

D'Alembert's Principle

The Net Force acting on a body is equal to the sum of all forces acting on that body.

$$F_{Net} = \sum_i F_i \tag{C.9}$$

Where F_{Net} is the Net foce and F_i is the i^{th} force acting on a body.

Equations of Angular Motion

$$\ddot{\boldsymbol{x}} = a = m^{-1}\boldsymbol{F} \tag{C.10}$$

$$\ddot{\boldsymbol{\theta}} = I^{-1}\boldsymbol{\tau} \tag{C.11}$$

$$\boldsymbol{\tau} = P_c \; x \; \boldsymbol{F} \tag{C.12}$$

$$\tau = a\hat{d} \tag{C.13}$$

$$w = \dot{\theta} = r\,\alpha \tag{C.14}$$

$$\acute{\theta} = \theta \dot{+} w \tag{C.15}$$

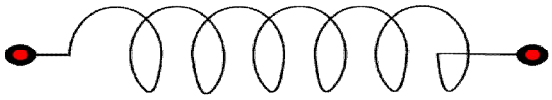
$$Quarternion = \dot{\theta} = \theta + \frac{\Delta x}{2} w\theta \tag{C.16}$$

$$w = \begin{bmatrix} 0 \\ \dot{\theta}_x \\ \dot{\theta}_y \\ \dot{\theta}_z \end{bmatrix} \tag{C.17}$$

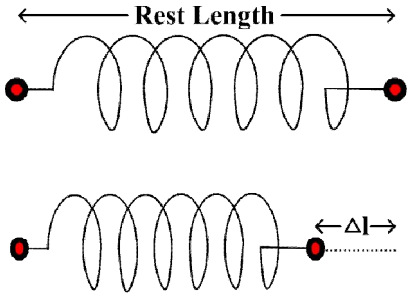
D'ALEMBERT'S PRINCIPLE FOR TORQUE

$$\tau_{NET} = \sum_{i} \tau_i \tag{C.18}$$

Springs



Hooks law:



$$F = -k\Delta l \tag{C.19}$$

Where F is the Force generated by compressing or XXXXX the spring, k is the spring constant, and ∆l is the displacement of the spring.

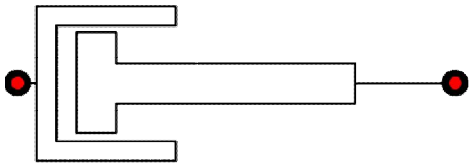
$$F = -k(l - l_r) \tag{C.20}$$

Where F is the Force generated by compressing or XXXXX the spring, k is the spring constant, l is the current position and l_r is the Rest length of the spring.

$$F = -k(|d| - l_r)\hat{d} \tag{C.21}$$

Where F is the Force generated by compressing or XXXXX the spring, k is the spring constant, l is the current position and l_r is the Rest length of the spring.

Damper



$$F_{d=} {}_{-}k_d \frac{dx}{dt} \tag{C.22}$$

Mass Spring Damper