|  |
| --- |
| Voxelization of a Human Body from an MRI Dataset and Ultrasound Simulation with Deformable Mesh Model  MASTER’S DEGREE THESIS IN  COMPUTER SCIENCE AND ENGINEERING  Author: **Nicolo’ Stagnoli** |
|  |
| Student ID: 964718  Advisor: Marco Gribaudo  Academic Year: 2022-2023 |

# Abstract

The aim of this work is to develop a three-dimensional representation of the human body so that it contains also “internal” texture sampled from the AustinMan dataset. This internal texture is then used to build new images from the intersection of a plane with the body. The power of this concept lies in the fact that is possible to build slices of the body in any orientation, not just only horizontal. The body can be also set in different poses and the position of the internal points is recalculated.

Regarding voxelizations, two algorithms are presented. The first, very simple, is a voxelization that builds meshes triangulating cubes. It is used to build simple models for the human skin and for every internal layer of the human body. Then, a slightly more complex algorithm, Marching Cubes, is presented. It works creating a polygonal surface mesh from a 3D scalar field by “marching” (looping) through the 3D space and determining each configuration for the given cube. It builds a smoother human skin mesh that, once bone-armored, is used to generate the colliders essential to capture the deformation of single body parts, like hands, arms, feet, etc., that is one main point of this MRI simulation.

**Keywords:** Voxelization, Ultrasound Simulation, UVW mapping, UVW deformation, Mesh generation, Mesh deformation.

# Abstract in Italian

L’obiettivo di questo lavoro è di sviluppare una rappresentazione tridimensionale di un corpo umano in modo tale che esso contenga anche una texture interna campionata dal dataset AustinMan. Questa texture interna è utilizzata per creare nuove immagini dall’intersezione di un piano con il corpo. La forza di questo concetto consiste nel fatto che sia possibile creare sezioni del corpo in ogni orientamento, non solo orizzontalmente. Il corpo può essere posizionato in diverse pose e la posizione dei punti interni viene quindi ricalcolata.

Riguardo alla voxelizzazione, sono qui presentati due algoritmi. Il primo, molto semplice, implementa una voxelizzazione che crea mesh triangolando cubi. Verrà utilizzato per costruire i modelli del corpo e di ogni tessuto interno. Successivamente, un algoritmo più complesso, Marching Cubes (Cubi Marcianti), è presentato. Esso crea una superficie poligonale da un campo scalare tridimensionale, facendo scorrere un cubo nello spazio, e determinando la configurazione del cubo in ogni luogo. Verrà usato per costruire un modello del corpo più liscio che, una volta armato per la deformazione, sarà usato per generare i collider essenziali a catturare la deformazione delle singole parti del corpo, come mani, braccia, piedi, ecc., che è uno dei principali punti di questa simulazione.

**Parole chiave:**

# Contents

[Abstract i](#_Toc131350055)

[Abstract in Italian iii](#_Toc131350056)

[Contents v](#_Toc131350057)

[1 Introduction 7](#_Toc131350058)

[1.1. MRI Scan 7](#_Toc131350059)

[1.2. The AustinMan Dataset 8](#_Toc131350060)

[1.3. Voxelization 9](#_Toc131350061)

[1.4. Ultrasound Simulator 9](#_Toc131350062)

[1.5. Purpose and motivation 10](#_Toc131350063)

[1.6. Tools 11](#_Toc131350064)

[1.7. Thesis Outline 11](#_Toc131350065)

[2 State of the Art in Ultrasound Simulation 13](#_Toc131350066)

[2.1. Ultrasound Simulation Methods 13](#_Toc131350067)

[2.2. Related works 14](#_Toc131350068)

[2.3. Objective 14](#_Toc131350069)

[3 Voxelization Algorithms 15](#_Toc131350070)

[3.1. Simple Voxelization 15](#_Toc131350071)

[3.1.1. Instantiating cubes 15](#_Toc131350072)

[3.1.2. Algorithm 15](#_Toc131350073)

[3.1.3. Implementation 16](#_Toc131350074)

[3.2. Marching Cubes 18](#_Toc131350075)

[3.2.1. Algorithm 18](#_Toc131350076)

[3.2.2. Example 20](#_Toc131350077)

[3.2.3. Implementation 21](#_Toc131350078)

[4 Ultrasound simulation with Deformable Mesh Model 23](#_Toc131350079)

[4.1. 3D Texture Shader 23](#_Toc131350080)

[4.2. Mesh Model Deformation 24](#_Toc131350081)

[4.2.1. Sphere colliders 24](#_Toc131350082)

[4.2.2. Mesh collider 25](#_Toc131350083)

[4.3. Breath simulation 25](#_Toc131350084)

[5 Conclusions 27](#_Toc131350085)

[6 Further Research 28](#_Toc131350086)

[Bibliography 29](#_Toc131350087)

[List of Figures 31](#_Toc131350088)

[Acknowledgments 33](#_Toc131350089)

# Introduction

Ultrasound is a widely used, low-cost imaging modality which plays an important role in clinical diagnosis. Image acquisition and diagnosis is, however, difficult  
to learn and mainly practiced with volunteers, phantoms, or patients. A practical learning phase is required to be able to perform it correctly. However, due to the high cost of the medical equipment and the necessary presence of a patient, the training process is complicated.

Today, technology has developed a lot and it is possible to use computer simulations  
to train in the medical field. In this case the advantages are many, first the possibility of testing individual skills in a simulated environment without these having consequences in the real world. In addition, another important feature is that of repeatability and reuse of the same simulation to evaluate different aspects of the ultrasound examination.

## MRI Scan

Since its development in the 1970s and 1980s, Magnetic Resonance Imaging, or MRI has proven to be a versatile imaging technique. MRI is a noninvasive medical imaging test that produces detailed images of almost every internal structure in the human body, including the organs, bones, muscles, and blood vessels. MRI scanners create images of the body using a large magnet and radio waves. No radiation is produced during an MRI exam, unlike X-rays. These images give your medical personnel important information in diagnosing your medical condition and planning a course of treatment.

The MRI machine is a large, cylindrical (tube-shaped) machine that creates a strong magnetic field around the patient and sends pulses of radio waves from a scanner. Some MRI machines look like narrow tunnels, while others are more open. The strong magnetic field created by the MRI scanner causes the atoms in your body to align in the same direction. Radio waves are then sent from the MRI machine and move these atoms out of the original position. As the radio waves are turned off, the atoms return to their original position and send back radio signals.

These signals are received by a computer and converted into an image of the part of the body being examined. This image appears on a viewing monitor.

MRI may be used instead of computed tomography ([CT](https://www.hopkinsmedicine.org/healthlibrary/conditions/adult/radiology/computed_tomography_ct_or_cat_scan_85,p01277/), or X-Ray) when organs or soft tissue are being studied. MRI is better at telling the difference between types of soft tissues and between normal and abnormal soft tissues.

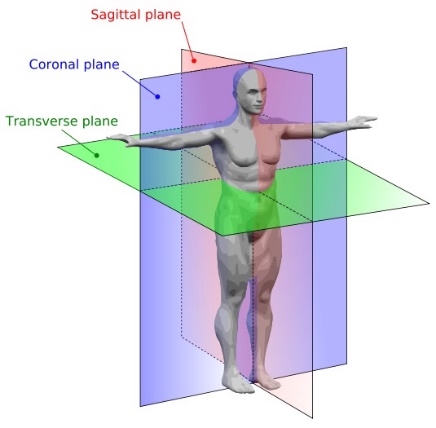
MRI provides exquisite detail of brain, spinal cord, and vascular anatomy, and has the advantage of being able to visualize anatomy in all three planes: axial, sagittal, and coronal.

Figure 1: Cutting planes of MRI.

## The AustinMan Dataset

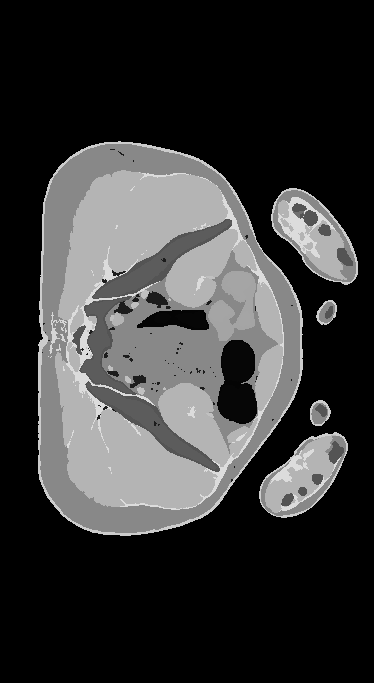
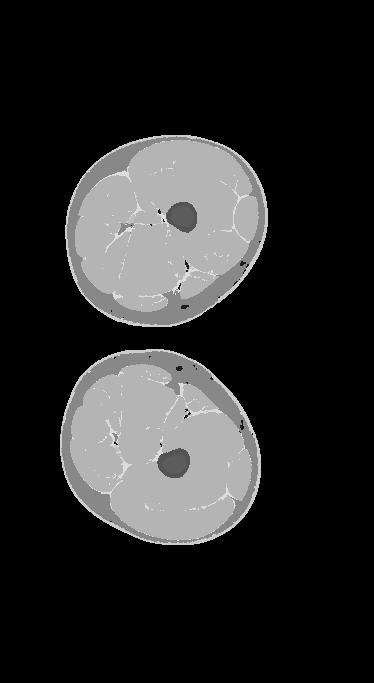
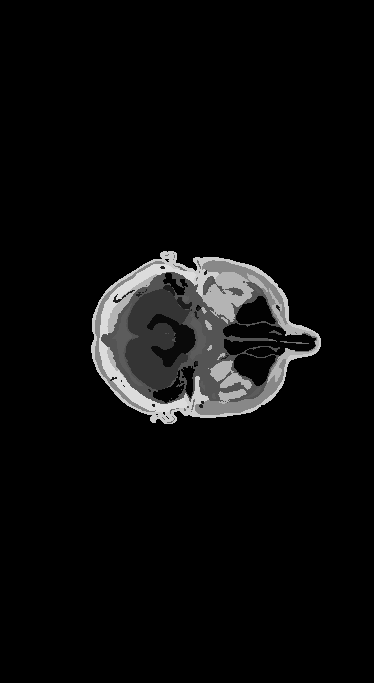
AustinMan is a voxel model of the human body that is being developed for simulations. This dataset was developed from a real MRI scan by segmenting the color cross-sectional (transverse plane) anatomical images. It contains 1878 horizontal (only) slices of a whole human body in different resolutions. The grayscale value of each pixel corresponds to a different layer of the body, such as tissues, bones, and organs, for a total of 64 layers.

Figure 2: Sample images from the AustinMan dataset

## Voxelization

The word *voxel* originated analogously to the word "pixel", with *vo* representing "volume" (instead of pixel's "picture") and *el* representing "element". A similar formation with *el* for "element" is the word "texel". The term *hypervoxel* is a generalization of voxel for higher-dimensional spaces.

In 3D computer graphics, a voxel represents a value on a regular grid in three-dimensional space. As with pixels in a 2D bitmap, voxels themselves do not typically have their position (i.e., coordinates) explicitly encoded with their values. Instead, rendering systems infer the position of a voxel based upon its position relative to other voxels (i.e., its position in the data structure that makes up a single volumetric image). Voxels are typically stored in three-dimensional matrices, but also smarter data structures can be used.

*Voxelization* is the term used to indicate the process of making three-dimensional model meshes by creating polygonal surfaces which properties depend on the voxel values.

In contrast to pixels and voxels, polygons are often explicitly represented by the coordinates of their vertices. Polygon points lie in continuous space, while voxels lie in a discrete space. A direct consequence of this difference is that polygons can efficiently represent simple 3D structures with much empty or homogeneously filled space, while voxels excel at representing regularly sampled spaces that are non-homogeneously filled.

A volume described as voxels can be visualized either by direct volume rendering or by the extraction of polygon iso-surfaces that follow the contours of given threshold values. The marching cubes algorithm is often used for isosurface extraction, however other methods exist as well.

Both ray tracing and ray casting, as well as rasterisation, can be applied to voxel data to obtain 2D raster graphics to depict on a monitor.

## Ultrasound Simulator

Medical ultrasound is a medical procedure used for therapeutic or diagnostic purposes that makes use of ultrasound waves. In diagnostic cases, the reflection of these waves is recorded and used to create an image of a patient's internal body structure, measure certain features, or record audible sounds.

This is made possible by ultrasound waves generated by piezoelectric crystals in the probe itself. When these are electrically excited, they begin to vibrate, generating waves with frequencies of more than 20 kHz that act on the patient's internal structures. The reflected waves are called back echoes, and by scanning them with a computer it is possible to obtain different images depending on the intended use of the ultrasound system.

An Ultrasound Simulator is a medical simulation training tool that enables educators and learners to practice diagnostic, therapeutic, and surgical applications as they relate to imaging interventions. A key advantage of using ultrasound simulation is that the practice makes advanced visualization, case databases and automatically generated feedback possible.

Overall, the main advantage of ultrasound simulators is that they provide new, innovative ways for learners to build up mental models. Yet, the most important feature of an ultrasound simulator is to provide feedback, which conforms to the concept of a mental model that must be updated. Another aspect that has been found critical across the realm of medical simulators is the ability to provide a range of difficulty levels for users with different skills.

Unfortunately, there are some disadvantages associated with this simulation   
systems.

First, these systems are now very expensive, and it is not possible to provide one to every student to train. These systems are generally quite bulky and require a very high computational power, which leads to being forced to use them in places set up for this purpose.

Furthermore, many of these systems consist in a “static” simulation, meaning that the models involved in the simulation are fixed and cannot be deformed. These models are often hand-crafted. A lot of effort is made to produce and validate these models, increasing the total cost of the work.

## Purpose and motivation

The solution chosen to overcome the disadvantages described above is to provide a  
reliable simulation of an ultrasound activity, using an ultrasound probe to perform the examination.

All the three-dimensional models of the human body skin and its internal tissues and organs involved in this simulation are generated with voxelization algorithms. These voxelization algorithm in general performs triangulations of surfaces by sampling values from a discrete field of three-dimensional points.

When the ultrasound probe intersects the body, the “internal” texture of it is shown on the screen in the scene.

The main point of the developed simulation is the possibility to deform the human body to set it in a general pose. Arms and legs can be oriented in any direction. Doing this, a technique to map every part of the body to the “internal” texture, and deform it, accordingly, is required.

## Tools

To develop the proposed solution, the *Unity* game engine framework is used, along with C# as programming language.

Unity is a cross-platform game engine developed by Unity Technologies; it can be used to create three-dimensional, two-dimensional, Virtual Reality and Augmented Reality simulations and games.

C# is an object-oriented programming language developed and maintained by Microsoft, and it is used as a scripting language in Unity.

Models generated with the voxelization algorithms are post-processed using Blender, which is a free and open-source 3D computer graphics software used for creating animated films, visual effects, art, 3D printed models, motion graphics, interactive 3D applications, Virtual Reality and computer simulations and games.

## Thesis Outline

Here a brief overview of this thesis content:

* Chapter 2 - State of the art in Ultrasound Simulation
* Chapter 3 - Voxelization Algorithms
* Chapter 4 - Ultrasound Echography simulation
* Chapter 5 - Conclusions
* Chapter 6 - Further work

# State of the Art in Ultrasound Simulation

The evolution and development of technologies and computer knowledge in recent years have radically changed the way we look at things in many ways and have brought many benefits in the working environment.

One of the sectors that has been affected by these benefits, and that is still evolving today, is the medical sector, especially the field of medical simulations. These are used in many areas of medicine, such as medical ultrasound, to identify or reproduce specific case studies that can be used for learning.

## Ultrasound Simulation Methods

In achieving educational outcomes, computer-based simulators mimic the ultrasound image produced within a computer. The methods to simulate ultrasound images can be categorized into:

* Interpolative
* Generative image-based
* Generative model-based

The interpolative approach uses prerecorded three-dimensional ultrasound volumes and slicing techniques, that can be combined with postprocessing like deformations and artificial shadow insertion in the final image. [3] [6]

The generative approach simulates ultrasound images using geometry from imaging systems like computed tomography (CT) and magnetic resonance (MRI), or it is based on mesh models. Generative model-based ultrasound simulators create an image by extracting a bi-dimensional slice from the model and texturizing it. Although this method is very appealing for generating and depicting cases involving different pathologies, modeling, and confirming that the model is correct can present challenges. The model creation for this approach is more complex than for the interpolative approach because each model needs some preprocessing. [7] [8]

Another generative model-based technique is based on ray tracing. Multiple ray emissions from the probe are simulated and the intersection points with the three-dimensional mesh model are calculated. Artifacts like shadowing and refractions can be added with postprocessing.

## Related works

Despite the numerous proposed approaches and the current presence of very accurate commercial applications, current implementations require a lot of computational power and/or a lot of previous manual work for the creation and validation of the models.

We therefore want to investigate whether it is possible to obtain similar results in a computationally lighter environment, also accessible from portable devices such as laptops, tablets, and smartphones. For this reason, the Unity game engine has been chosen as working environment.

In the past year, here at Politecnico di Milano, Diego Zucca presented a lightweight reliable simulation of an ultrasound activity, using an ultrasound scanner and an  
ultrasound probe to perform the examination, running on VR (Virtual Reality headset). Thanks to this simulation, the sensation of immersion of the environment in which the user finds himself and the fidelity of the simulation of an ultrasound examination achieve a good result of realism compared to the cost necessary for the use of the application, thus finding a good quality-price ratio. [citazione]

## Objective

The aim of this work is to enhance the previous work by Diego Zucca to account for model deformation. Setting the model in different poses could be interesting to see what happens to internal body part when the pose is changed, or to see what the effect of involuntary body movements are, like breath, on the internal texture. For this reason, a technique to map every part of the body to the “internal” texture, and deform it, accordingly, will be investigated.

# Voxelization Algorithms

In this chapter the voxelization algorithms used to generate the meshes of the human body are presented.

## Simple Voxelization

### Instantiating cubes

The simplest method to construct a three-dimensional model from the stack of images of the dataset is to instantiate a cube for each colored pixel, since black color corresponds to air. However, due to the high number of pixels in the dataset, this simple approach is not feasible because memory overload. The number of vertices is too high, and most of them are inside the body, so not even rendered.

### Algorithm

The idea is to algorithmically create vertices and triangulate them to make simple cubes. Each pixel in the dataset images corresponds to a cube. If one face of a cube being drawn is covered by another cube, the face is inside the mesh, and it doesn’t need to be rendered. Doing so, only the external part, the skin, of the human body is drawn, while the inside is empty.

A statue of a person

Description automatically generated with medium confidenceThis simple approach is used also the build each layer of the human body. Different meshes are built iterating each time on the whole dataset, taking only pixels with a specific value of the grayscale value.

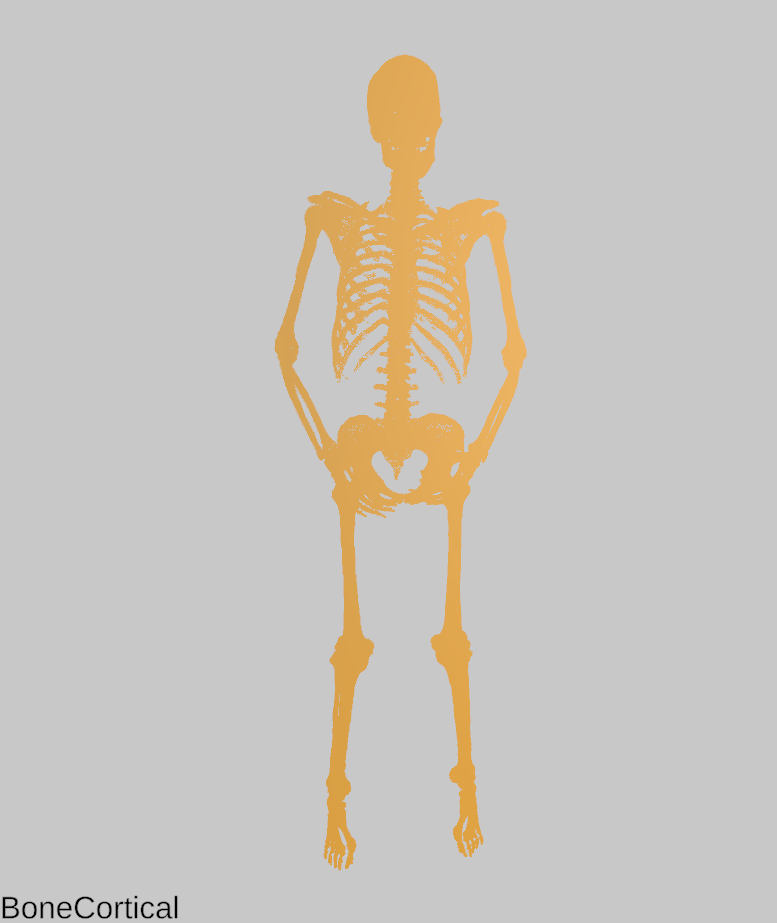
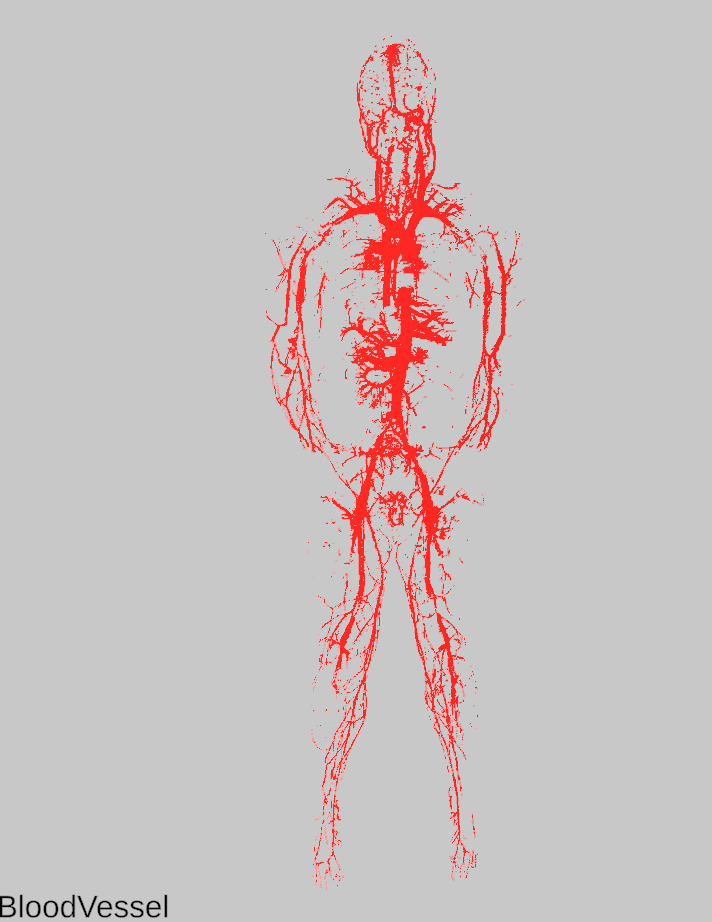
Figure 3: The mesh generated with the first algorithm.

### Implementation

The implementation of this algorithm can be found in the *Layers\_Voxelization* scene in the Unity project. The *AustinMan* game object is set with an initial empty MeshFilter a standard MeshRenderer, and a script component called *Voxel\_Grid\_Layers*.

This script initializes the three-dimensional voxel matrix with the grayscale pixel values of the AustinMan dataset images, with values from 0 to 255. After this, an array of Unity mesh is initialized, one for each grayscale. By looping all the voxels, a cube is added to the mesh of the corresponding layer. Each cube is set with world coordinates corresponding to the position of the voxel in the matrix. Cubes are built by creating faces with the *AddQuad* function. For each mesh (layer), a face of a specific cube is currently added only if there is no other cube (of the same layer) directly next to it. In this way, only visible faces are added to the mesh and rendered, making the voxelization feasible.

|  |  |
| --- | --- |
| int[,] Faces = new int[6, 9]{  {0, 1, 2, 3, 0, 1, 0}, //top  {7, 6, 5, 4, 0, -1, 0}, //bottom  {2, 1, 5, 6, 0, 0, 1}, //right  {0, 3, 7, 4, 0, 0, -1}, //left  {3, 2, 6, 7, 1, 0, 0}, //front  {1, 0, 4, 5, -1, 0, 0} //back  }; | Vector3[] VertPos = new Vector3[8]{  new Vector3(-1, 1, -1), new Vector3(-1, 1, 1),  new Vector3(1, 1, 1), new Vector3(1, 1, -1),  new Vector3(-1, -1, -1), new Vector3(-1, -1, 1),  new Vector3(1, -1, 1), new Vector3(1, -1, -1),  }; |
| void AddQuad(int facenum, int v, int meshIndex) {  for (int i = 0; i < 4; i++) Vertices[meshIndex].Add(new Vector3(x, y, z) + VertPos[Faces[facenum, i]] / 2f);  Triangles[meshIndex].AddRange(new List<int>() { v, v + 1, v + 2, v, v + 2, v + 3 });  }  for (int x = 1; x < Dimensions.x - 1; x++) {  for (int y = 1; y < Dimensions.y - 1; y++) {  for (int z = 1; z < Dimensions.z - 1; z++) {  int layer = Voxels[x, y, z];  int meshIndex = Array.IndexOf(layers, layer);  for (int o = 0; o < 6; o++)  if (Voxels[x + Faces[o, 4], y + Faces[o, 5], z + Faces[o, 6]] != layer)  AddQuad(o, Vertices[meshIndex].Count, meshIndex);  }  }  } | |



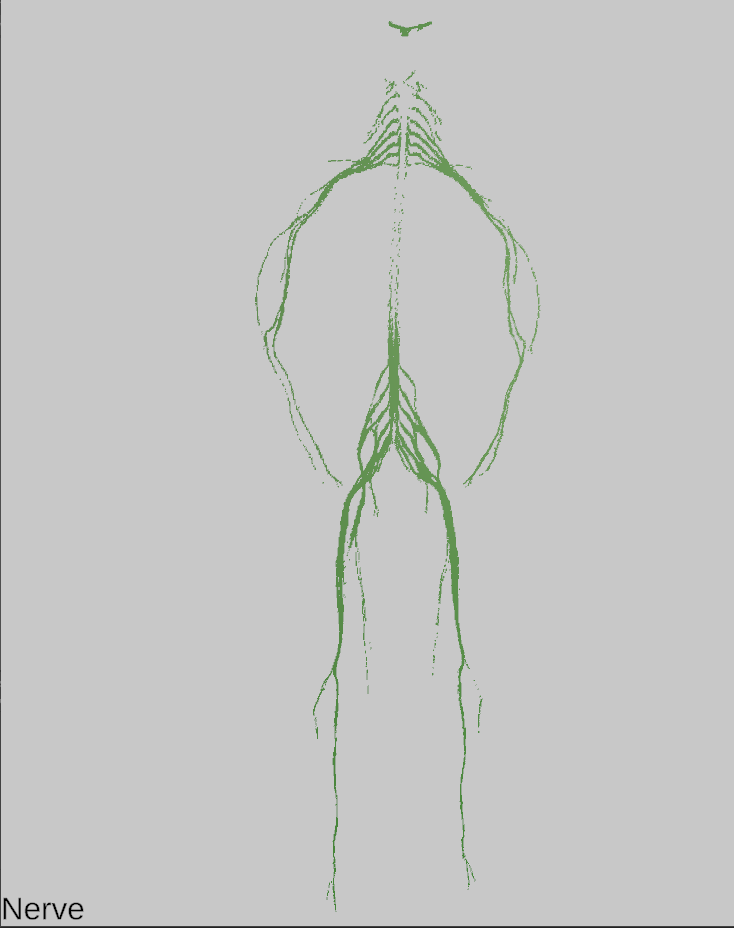


Figure 4: Some of the layers generated with the first algorithm.

## Marching Cubes

The marching cubes algorithm is mainly used to procedurally generate terrains (meshes, in general) from a discrete field of values, which can be randomly generated from noise or taken by image maps. The marching cubes algorithm creates a polygonal surface mesh from a 3D scalar field by “marching” (looping) through the 3D space and determining each configuration for the given cube.

### Algorithm

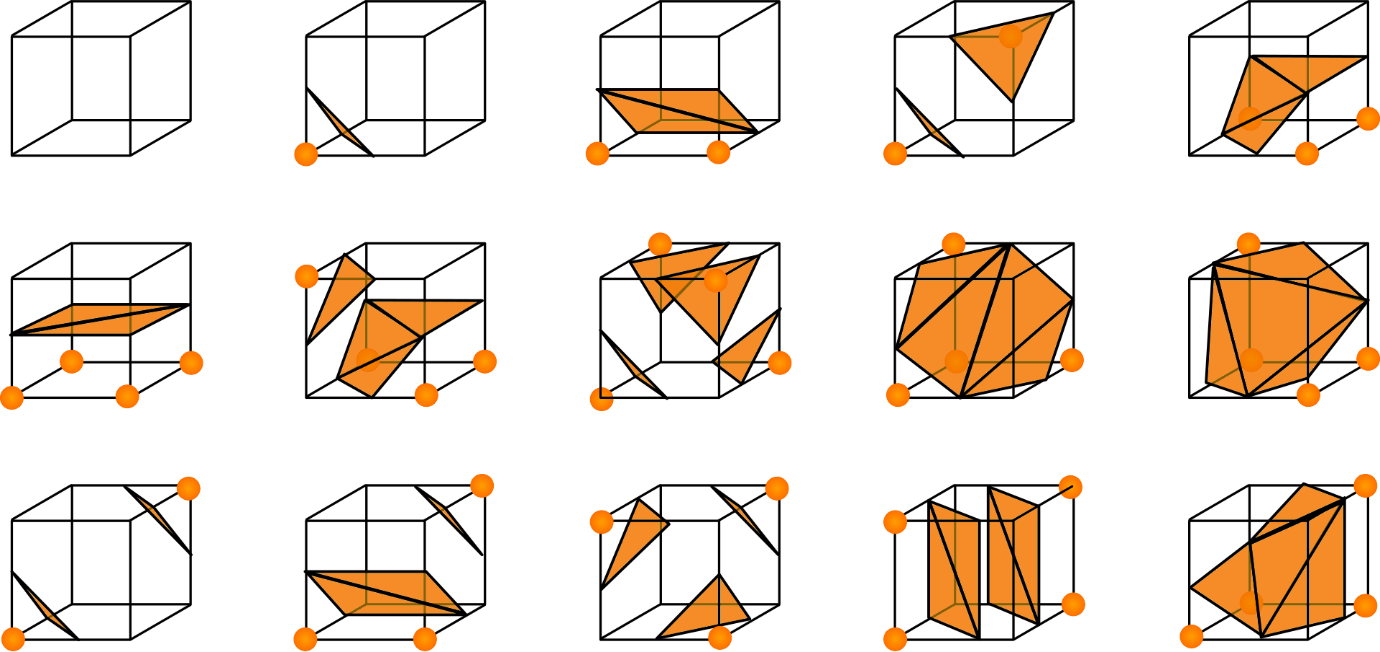
Every point in the 3D world is a value from 0 to 1, where 0 is black and above ground, and 1 is white and underground, or vice versa. We march a single cube through the 3D space and construct a mesh. When the value at a vertex is below a given threshold, also called isosurface, we can say that this vertex of our cube in the terrain is underground, and we want to hide it by drawing a face. A configuration is chosen by determining which of those vertices are below the isosurface, and which are not. In total there are 256 such combinations that can be formed by looking at the values of our vertices since cubes have 8 corners with each 2 possible states. These 256 configurations can be reduced to only 15 since most cases are symmetries.

Figure 5: Configurations of a single cube in the marching cubes algorithm.

The algorithm begins by determining the configuration of the cube, by comparing the value of our cube at every corner vertex with the isosurface level. Cube configuration is then found with a lookup table, which contains lists of edges for each of the 256 possible vertex configurations. The configuration corresponds to an 8-bit word, one bit for each vertex. Bits corresponding to vertices below the isosurface level are set to 1.

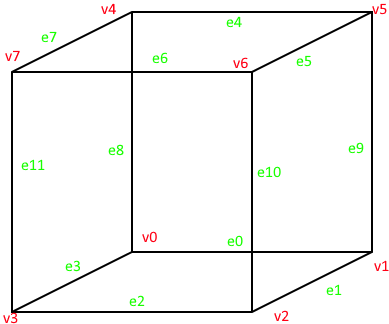
The cube configuration gives us the list of “active” edges on the cube. The vertices of the triangles generated lie on these edges. Given the edges indexes, another lookup table is used to find the two cube vertices that this edge is between. So, there will be two cube vertices for each “active” edge.

Figure 6: Numbering of vertices and edges on a single cube.

The index of these two cube vertices is then used with another lookup table to get the local 3D coordinates of the cube vertices.

The last step consists of interpolating between those found vertices to estimate where along the edge the final vertex is, and this is done to give a smoother look. Interpolation is done considering the isolevel value of each vertex. Interpolation is done with the following formula:

Where is the final triangle vertex, and are the 3d coordinates of the cube vertices to interpolate, v1 and v2 are the isolevel values of the cube vertices, and k is the considered isosurface level. Notice that, since v2 corresponds to the second cube vertex of the “active” edge, its isolevel value is always greater than v1.

### Example

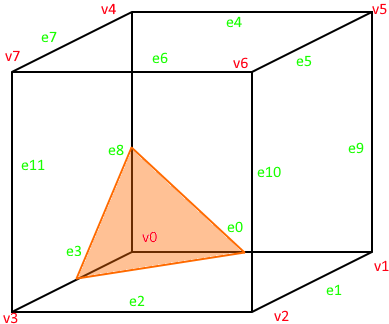
For example, if vertex 0 has a value of 0, and all other vertices have a value of 1.0, given that the isosurface level is 0.5, we can conclude that since vertex 0 is the only vertex below the threshold, so we want to “hide” this vertex by creating a triangle in front of it by connecting edges 0, 3 and 8.

Figure 7: Example of a triangulated face with the marching cubes algorithm.

The cube configuration index is built by setting the least significant bit of the 8-bit word to 1. So, the cube configuration index we get is 1. From the first lookup table at this index, we get these edge indexes: {0, 8, 3}. These means that the vertices of the triangle lies on e0, e8, and e3.

From the second lookup table at these indexes, we get the following edge connections: {0, 1}, {0, 4}, {3, 0}. Indeed, v1 is the other vertex of e0, v4 of e8, and v3 of e3.

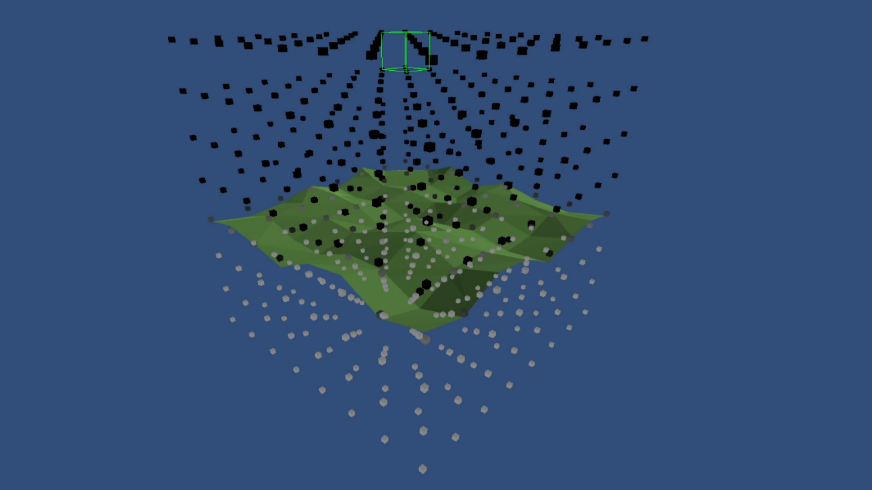
With these vertex indexes, we get the local cube coordinates of the vertices, which are then interpolated with the previous formula to get the final triangle vertices.

Figure 8: An example of a terrain generated with the marching cubes algorithm.

### Implementation

The implementation of this algorithm can be found in the *Marching\_Cubes* scene in the Unity project. The *AustinMan* game object is set with an initial empty MeshFilter a standard MeshRenderer, and a script component called *Marching\_Cubes*.

As in the previous scene, this script initializes the three-dimensional voxel matrix with the grayscale pixel values of the AustinMan dataset images, this time with floating point values from 0 to 1. Voxel values corresponding to the air surrounding the body (all the black color in the dataset images) are set to 1, while all the voxel values corresponding to body parts are set to 0. The Isolevel value is set to 1. In this way, only the outer skin of the body will be generated by the algorithm.

|  |  |
| --- | --- |
| currentCubePosition = new Vector3(x + .5f, y + .5f, z + .5f);  // Set values at the corners of the cube  float[] cubeValues = new float[] {  pointValues[x, y, z + 1],  pointValues[x + 1, y, z + 1],  pointValues[x + 1, y, z],  pointValues[x, y, z],  pointValues[x, y + 1, z + 1],  pointValues[x + 1, y + 1, z + 1],  pointValues[x + 1, y + 1, z],  pointValues[x, y + 1, z]  };  // Find the triangulation index  int cubeIndex = 0;  if (cubeValues[0] < isoLevel) cubeIndex |= 1;  if (cubeValues[1] < isoLevel) cubeIndex |= 2;  if (cubeValues[2] < isoLevel) cubeIndex |= 4;  if (cubeValues[3] < isoLevel) cubeIndex |= 8;  if (cubeValues[4] < isoLevel) cubeIndex |= 16;  if (cubeValues[5] < isoLevel) cubeIndex |= 32;  if (cubeValues[6] < isoLevel) cubeIndex |= 64;  if (cubeValues[7] < isoLevel) cubeIndex |= 1  // Get the intersecting edges  int[] edges = MarchingCubesTables.triTable[cubeIndex];  Vector3 worldPos = new Vector3(x, y, z);  int triCount = triangles.Count; | // Triangulate  for (int i = 0; edges[i] != -1; i += 3) {  int e00 = MarchingCubesTables.edgeConnections[edges[i]][0];  int e01 = MarchingCubesTables.edgeConnections[edges[i]][1];  int e10 = MarchingCubesTables.edgeConnections[edges[i + 1]][0];  int e11 = MarchingCubesTables.edgeConnections[edges[i + 1]][1];  int e20 = MarchingCubesTables.edgeConnections[edges[i + 2]][0];  int e21 = MarchingCubesTables.edgeConnections[edges[i + 2]][1];  Vector3 a = Interp(MarchingCubesTables.cubeCorners[e00], cubeValues[e00], MarchingCubesTables.cubeCorners[e01], cubeValues[e01]) + worldPos;  Vector3 b = Interp(MarchingCubesTables.cubeCorners[e10], cubeValues[e10], MarchingCubesTables.cubeCorners[e11], cubeValues[e11]) + worldPos;  Vector3 c = Interp(MarchingCubesTables.cubeCorners[e20], cubeValues[e20], MarchingCubesTables.cubeCorners[e21], cubeValues[e21]) + worldPos;  AddTriangle(a, b, c);  } |



Figure 9: The mesh generated with the marching cubes algorithm.

This smoother mesh for the outer skin will be subsequently used in the ultrasound simulation as a “container” for the “internal” texture.

# Ultrasound simulation with Deformable Mesh Model

This chapter describes all the techniques used to make the simulation.

## 3D Texture Shader

A 3D texture is a bitmap image that contains information in three dimensions rather than the standard two. 3D textures are commonly used to simulate volumetric effects such as fog or smoke or to approximate a volumetric 3D mesh. In this work, a 3D texture is made by building a three-dimensional matrix from the dataset images. Each horizontal slice of the matrix corresponds to a slice image in the dataset.

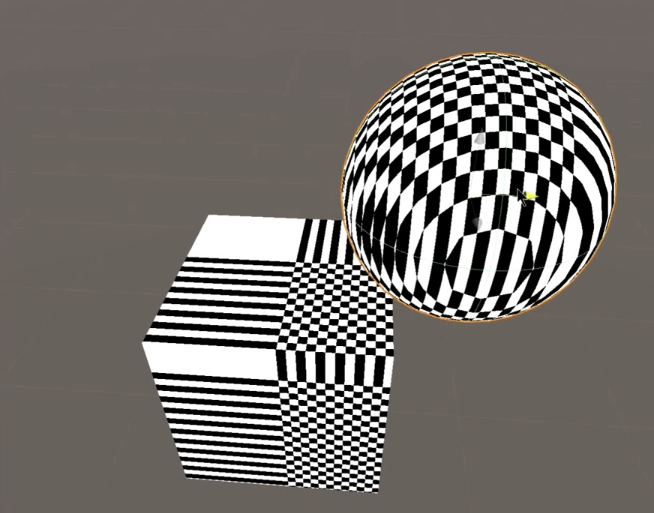
3D texture can be rendered just like normal 2D textures, with the same types of filtering and interpolations to calculate final pixel values; the only difference is, obviously, the presence of one extra dimension for texture coordinates: there will be UVW coordinates. 3D texture can be applied to any mesh.

Figure 10: Example of a three-dimensional checkerboard texture applied to a cube and to a sphere.

In this work, to simulate the cut/slice effect of the MRI, a simple plane is used. This plane is set in the world origin. Every time the plane position or rotation changes, the UVW coordinates of each vertex of the plane are set to the world position coordinates of the vertex itself. To render the same image on the back screen, that is a plane just like the one used to slice the texture, the same UVW coordinates are applied to vertices of the screen. This is just a “static” mapping of the 3D texture.

## Mesh Model Deformation

**One of the main points of this work is to make the human model deformable, (just rotation and translation of body parts), with the mesh carrying the “internal” texture along the way. To do this, two approaches are presented.**

### **Sphere colliders**

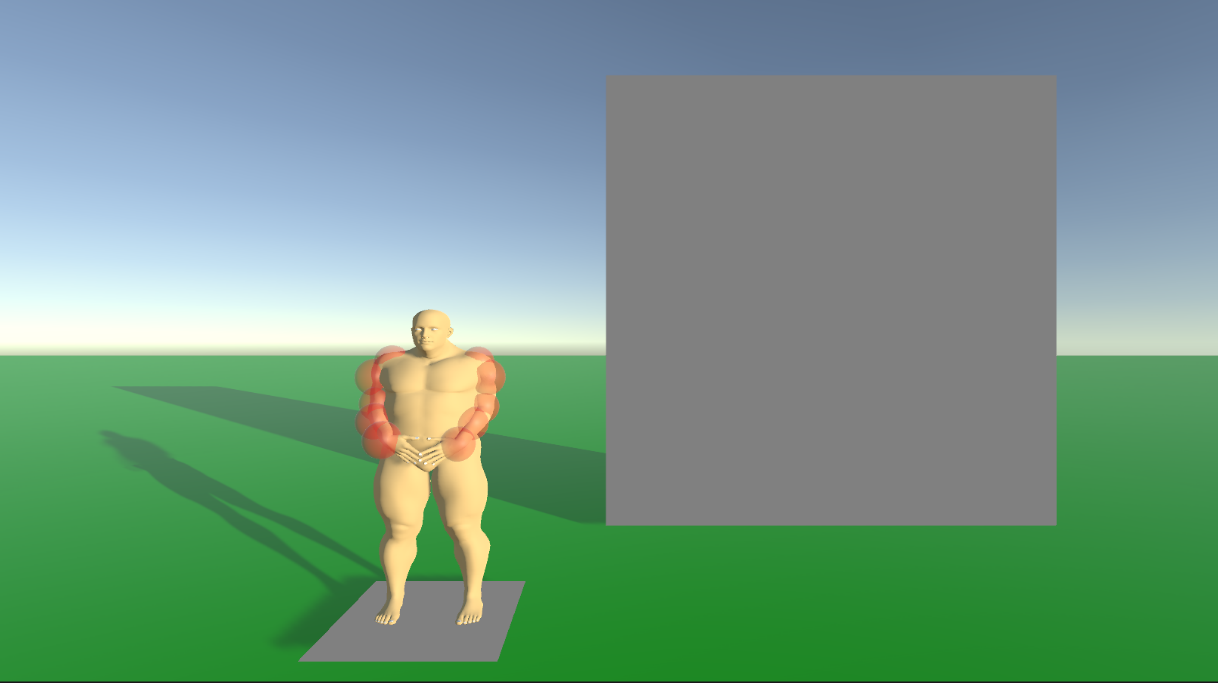
**The first approach consists in manually placing spheres along arms and legs. These spheres are attached to the model rig. The initial position of this spheres is stored at the start. the idea is that, whenever the model rig is deformed, each point position of the cutting plane is compared with all the spheres: if the point is inside one of them, the inverse of the sphere rotation and translation is applied to the UVW coordinates of that vertex. If the involved sphere is still in the original position, there is no rotation and translation, and the result is the same of the “static” version.**

Figure 11: Spheres set on the arms used to check the collision with the cutting plane.

**These methods work well, but results applied to this precise human model is not satisfying, due to the nonstandard base position of the mesh (so also of the 3d texture), the initial overlapping of some body parts, and the high inaccuracy of the spheres to approximate body parts. The approach, however, is working.**

### **Mesh collider**

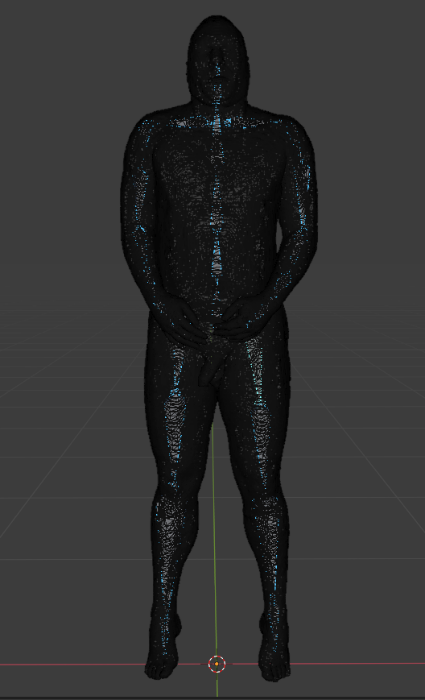
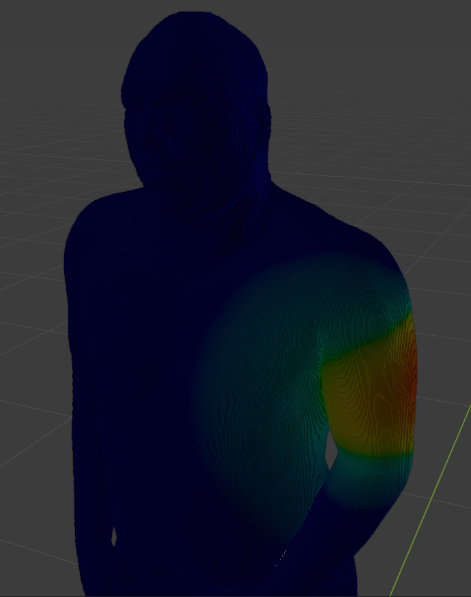
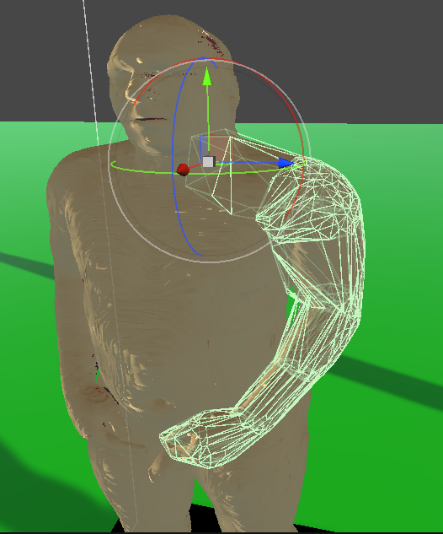
The final approach consists in automatically generating small convex mesh colliders for each part of the body to better approximate its bounding. Exploiting the bone weights of the human model rig, the model is subdivided into sub-meshes by considering which vertex is attached to which bone. These sub-meshes colliders are regenerated every time the model rig is deformed. As in the previous method, the initial position and orientation of each collider is stored at the start to subsequently apply the inverse transform to plane points.

Figure 12: Colliders of the left arm generated from the bone weights.

Figure 13: An example of bone weighting.

Figure 14: The model rig.

Notice that, for both approaches, the plane has been subdivided: it contains about 16.000 vertices. This is done because, differently from the “static” version of the MRI, this time different parts of the 3D texture needs to be rendered on the plane, and these parts in general are no more “aligned” (in the UVW space), due to the possibility of deforming the model rig. This would not be possible if the plane had only 4 vertices.

## Breath simulation

To simulate the breathing movement of the body, the chest bone is animated. For a period of about 4 seconds, it’s scale is increasing and decreasing. Applying scale to a bone result in scaling all the vertices assigned to that bone. The problem is that the scale is applied also to all the child bones, so also the head and the arms are enlarged. To prevent this, the same animation, but with inverse scale factor, is applied to all the child bones that doesn’t need to be enlarged. Colliders are automatically recalculated each frame. The local scale of each bone/collider is also used to calculate the inverse transform to be applied to plane points.

# Conclusions

# Further Research

# Bibliography

|  |  |
| --- | --- |
| [1] | J. W. Massey and A. E. Yilmaz, "[AustinMan and AustinWoman: High-fidelity, anatomical voxel models developed from the VHP color images](http://ieeexplore.ieee.org/document/7591444/)," in Proc. 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (IEEE EMBC), Orlando, FL, Aug. 2016. |
| [2] | https://polycoding.net/marching-cubes/part-1/ |

https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/magnetic-resonance-imaging-mri

https://www.healthysimulation.com/ultrasound-simulator/

IEEE TRANSACTIONS ON MEDICAL IMAGING, VOL. 32, NO. 3, MARCH 2013 609  
Real-Time GPU-Based Ultrasound Simulation Using Deformable Mesh Models  
Benny Bürger\*, Sascha Bettinghausen, Matthias Rädle, and Jürgen Hesser

# List of Figures

[Figure 1: Cutting planes of MRI. 8](file:///C:\Users\n\Desktop\Thesis%20template_W.docx#_Toc131176088)

[Figure 2: Sample images from the AustinMan dataset 8](file:///C:\Users\n\Desktop\Thesis%20template_W.docx#_Toc131176089)

[Figure 3: The mesh generated with the first algorithm. 15](file:///C:\Users\n\Desktop\Thesis%20template_W.docx#_Toc131176090)

[Figure 4: Some of the layers generated with the first algorithm. 17](file:///C:\Users\n\Desktop\Thesis%20template_W.docx#_Toc131176091)

[Figure 5: Configurations of a single cube in the marching cubes algorithm. 18](file:///C:\Users\n\Desktop\Thesis%20template_W.docx#_Toc131176092)

[Figure 6: Numbering of vertices and edges on a single cube. 19](file:///C:\Users\n\Desktop\Thesis%20template_W.docx#_Toc131176093)

[Figure 7: Example of a triangulated face with the marching cubes algorithm. 20](file:///C:\Users\n\Desktop\Thesis%20template_W.docx#_Toc131176094)

[Figure 8: An example of a terrain generated with the marching cubes algorithm. 20](file:///C:\Users\n\Desktop\Thesis%20template_W.docx#_Toc131176095)

[Figure 9: The mesh generated with the marching cubes algorithm. 22](file:///C:\Users\n\Desktop\Thesis%20template_W.docx#_Toc131176096)

[Figure 10: Example of a three-dimensional checkerboard texture applied to a cube and to a sphere. 23](file:///C:\Users\n\Desktop\Thesis%20template_W.docx#_Toc131176097)

[Figure 11: Spheres set on the arms used to check the collision with the cutting plane. 24](file:///C:\Users\n\Desktop\Thesis%20template_W.docx#_Toc131176098)

[Figure 12: Colliders of the left arm generated from the bone weights. 25](file:///C:\Users\n\Desktop\Thesis%20template_W.docx#_Toc131176099)

[Figure 13: An example of bone weighting. 25](file:///C:\Users\n\Desktop\Thesis%20template_W.docx#_Toc131176100)

[Figure 14: The model rig. 25](file:///C:\Users\n\Desktop\Thesis%20template_W.docx#_Toc131176101)

# Acknowledgments

Here you might want to acknowledge someone.