MASTER

GPU supported medical X-ray image simulation

Xu, X.

Award date:
2018

GPU Supported Medical X-ray Image Simulation

Master Thesis

By

XIN XU

Department of Mathematics and Computer Science
EINDHOVEN UNIVERSITY OF TECHNOLOGY

OCTOBER 2018

Abstract

Nowadays, X-ray imaging system plays an important role in the medical and health-care chain, helping doctors diagnose and locate disease in organs and bones effectively. By performing simulation of such system, especially the part of X-ray imaging physics, the development of X-ray systems can be speeded up and X-ray systems can be tested and verified virtually before they are actually built. The simulation of the X-ray system involves the ray tracing of three-dimensional human body models (also known as phantoms). Currently, the commonly used data model for phantoms is voxelized model. However, simulation of voxelized phantoms is both time-consuming and cannot reach a satisfying image quality. Therefore, this thesis will explore the possibility of a new data model, Non-Uniform Rational B-Spline(NURBS).

Two approaches related to the simulation of NURBS object are proposed. The first one is X-ray tracking NURBS surface using ray tracing technique, employing Newton's iteration as the method to calculate intersection between ray and surface. The second approach is direct mathematical projection of NURBS surface, which avoids intersection calculation for each ray and directly approximates the value of each pixel. By comparison of these two approaches with the voxelized model approach and the popular triangle meshed model approach, it turns out direct mathematical projection of NURBS surface has a overall better performance (on a 9 megapixel detector plane) and flexibility to adjust its accuracy to realize different spatial resolution for the output X-ray image.

Thesis Committee:

Supervisor: Prof.dr. H. Corporaal, EE, ES, TU Eindhoven
In-company Supervisor: Dr.rer.nat. Klaus Juergen Engel, Philips Research
Committee Member: Dr.ir. A.T. Nelson, EE, ES, TU Eindhoven
Committee Member: Dr. M.A. Westenberg, CS, Algorithms&Visualization W&I, TU Eindhoven

ACKNOWLEDGEMENT

I would first like to thank my in-company supervisor Dr.rer.nat. Juergen Engel. Whenever I ran into a trouble spot or had a question about my research or writing, he steered me in the right the direction whenever he thought I needed it.

I would also like to thank Prof.dr. H. Corporaal who were involved in the whole process of this thesis. With his participation and suggestion, the research is conducted as expected.

I would also like to thank Dr.ir. A.T. Nelson for providing me an opportunity to work in this interesting field of research.

I would also like to thank Dr. Alberto Fazzi, department head of the image quality group at Philips Healthcare Best, who provided the funding for my thesis and nice working conditions.

I would also like to acknowledge Dr. M.A. Westenberg as the second reader of this thesis, and I am gratefully indebted to his very valuable comments on this thesis.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Table of Contents

CHAPTER 1

INTRODUCTION AND PROBLEM STATEMENT

In the medical and health-care domain, X-ray imaging systems play an important role as a tool to help doctors diagnose and locate diseases in organs and bones effectively. X-ray imaging refers to the physical properties of X-rays to transmit a patient in straight lines and generate a projected image of the three-dimensional mass density distribution of the human body on a detector plane. Virtualization and simulation of such system, especially the part of X-ray imaging physics, speeds up the development of X-ray systems and can virtually test and verify them before they are actually built. For example, for the verification of the X-ray system, a testing of an intended clinical workflow can be performed.

The simulation of the X-ray system involves determining intersection points between X-rays with human body model (phantom), in particular with three-dimensionally shaped volumes representing different parts and organs of the human body. Currently, the commonly used data model to represent phantom is voxelized model. However, simulation of voxelized model is both time-consuming and cannot reach a satisfying image quality. Fig.1.1 demonstrates an X-ray image simulated from a voxelized phantom, showing significant image artifacts and taking around 3 seconds to render. Therefore, this thesis will explore the possibility of a new data model with the usage of parallel hardware architecture (such as graphics processing unit).

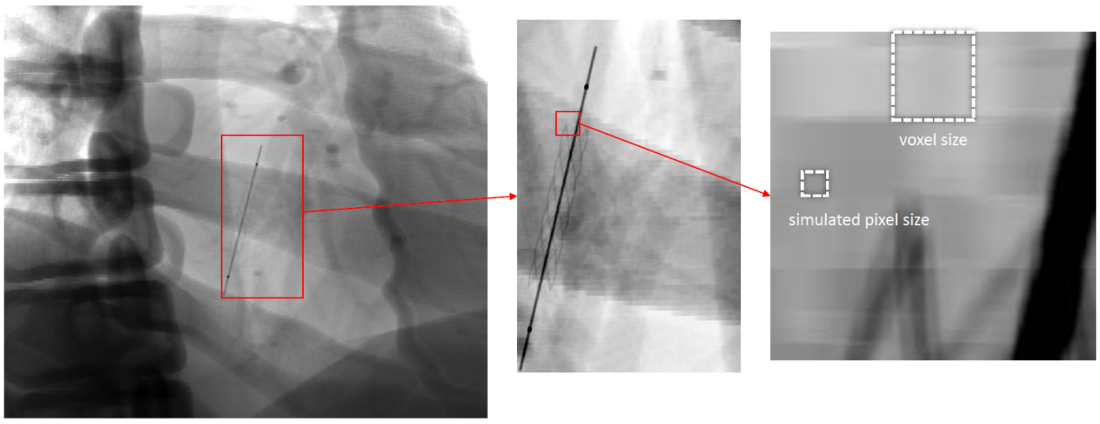


FIGURE 1.1. The result X-ray image from simulating a voxelized human phantom. When zooming in the region where the vessel stent is, some stripes appear and the edge of object is not curved. After further zooming in, blocks that actually indicate voxels show up and they are much larger than the image pixels aligned to the spatial resolution of the simulated X-ray detector.

1.1 Problem statement

There are two main use cases with different problems to solve: First, enabling real-time simulation speed, and second, realizing realistic image quality.

1. **The projection image needs to be created in real-time for X-ray simulation**
   The speed of the simulation is essential and an interactive speed should be achieved. The real-time simulation speed would be the highest priority requirement in the thesis. This problem is related to the wish of providing a "realistic user experience" by simulation. For example, a simulation user would like to see the changing of the viewing angle of the X-ray system immediately in the image. A reduced spatial resolution might be an acceptable compromise.

2. **A highly realistic image simulation is required**
   There are two aspects that indicate the definition of "highly realistic image simulation". The first is that rendered image should have a high spatial resolution. The second is that the rendered object surfaces should be smooth and without artifacts. This use case is for analysis of image quality. A resolution higher than the real-world detector resolution is absolutely required (i.e. to enable the simulation of detector blurring effects). The loss of "real-time" can be accepted as a compromise.

1.2 Contributions of the thesis

To tackle the problems, advantages and disadvantages of different data models that represent the human body has been investigated. The choice of data models is decisive for the image quality and the computation performance. Based on that, two types of data models (triangle-meshed model and NURBS model) are selected and implemented according to literature. Experiments are performed to compare and evaluate both data models from the perspectives of image quality and computation performance. Graphic processing unit is applied to further accelerate computation of the algorithms. A novel method which has a high computation efficiency and flexibility is presented based on NURBS. Tests has been done to compare this method with other existing approaches. The contributions of the thesis are listed below:

1. An existing algorithm that directly calculate intersections with NURBS using numerical approach for X-ray physics has been applied.
2. A novel method called direct mathematical projection of NURBS surface is proposed, which promises higher computation efficiency and better image quality.
3. An algorithm that tracks triangle-meshed models for X-ray physics is adapted from ray tracing algorithm.
4. Comparison among voxelized model, triangles-meshed model and the novel method (direct mathematical projection of NURBS surface) on image quality and performance has been done. Results show that the new method achieves a better image quality unless the object is represented by huge amount of triangles or voxels. It also verifies that the new approach has better scalability with the detector size than simulation with voxelized model.

In Chapter 2, more background information such as the basics of NURBS is reviewed. In Chapter 3, two approaches regarding development on NURBS are proposed. In Chapter 4, tests are conducted for two proposed approaches and existing approaches. They are compared and evaluated according to the test result. In Chapter 5, suggestions for future development and improvement are given. Chapter 6 concludes this thesis by summing up important observations.

Chapter 2

Background

In this chapter the basic concepts and influential techniques in the field of ray tracing and NURBS are introduced. Literature research has been done by reviewing related work. Additionally, the advantages and disadvantages of those techniques are discussed, both from the perspective of X-ray simulation and GPU computing.

2.1 X-ray system simulation

2.1.1 X-ray systems

X-ray imaging systems are an important part of the medical health-care chain. As an example, Fig.2.1 shows a typical "C-arm" system used for medical diagnosis and interventional treatment. The system got its name from the shape of a mechanical rotatable "C" with an X-ray source and an X-ray sensitive image detector mounted at its ends, respectively.

The mechanical "C" can be highly flexible positioned around a patient lying on the table. The "point-like" X-ray source can emit an X-ray beam directed towards the detector. The detector itself is typically a digital version of a "photographic plate" having a size up to 40 cm x 40 cm.

FIGURE 2.1. Philips Allura Xper FD20/20 biplane neuro X-ray system

In order to accelerate the development of such systems, it is necessary to test and verify them virtually before actual construction, in especially the part of X-ray imaging physics. Its principle is demonstrated in Fig.2.3. In contrast to an optical ray, an X-ray does not reflect and refract upon a surface but transmit through all objects on its path. Due to the property of X-ray, optical effects such as refraction and reflection are not considered. Instead, the X-ray attenuation is the key concept in the simulation and the depth of each intersection on each ray is the essential variable to generate a X-ray image. The intensity of an X-ray is reduced when its depth inside a object becomes larger, known as X-ray attenuation. The definition of X-ray attenuation is shown in Equation 2.1. By obtaining the information of depth, the various intensities of X-rays that transmit through different types of objects can be transformed using Equation 2.1. The major goal of the simulation algorithm in this thesis is to obtain a depth map (in projection geometry) of the examined objects. In addition, Equ.2.2 shows the case when there are segments $depth(i)$ of multiple materials $\mu(i)$ (see Fig.2.2).

$$Intensity = Intensity_O \times e^{-\mu \times depth} \tag{2.1}$$

where $Intensity_O$ is the X-ray intensity emitted from a focal spot as detected without any object present in the beam and μ is the linear attenuation coefficient depending on the material assigned to the depth segment.

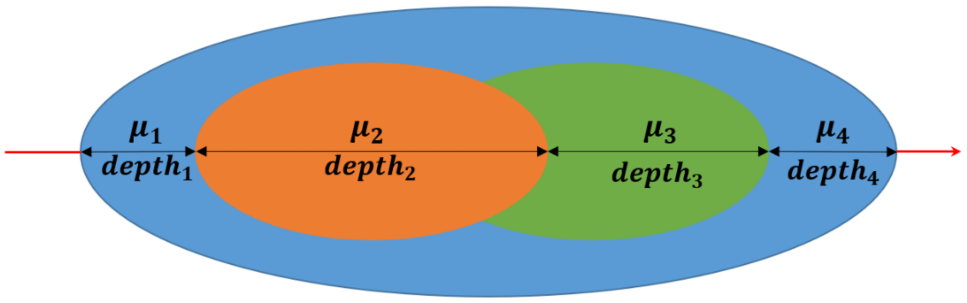
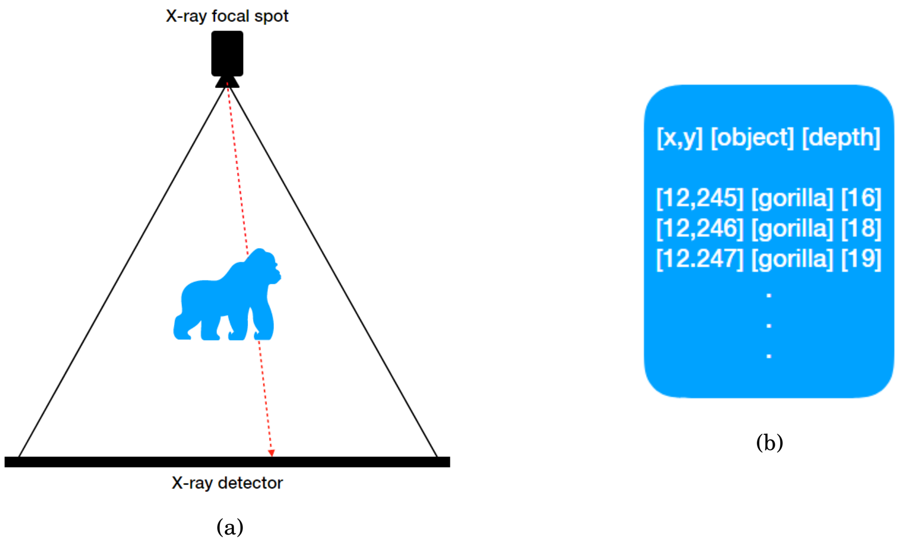


FIGURE 2.2. The situation when multiple objects overlapping each other in an X-ray scene.

$$Intensity = Intensity_O * e^{-\mu(1)*depth(1)} * e^{-\mu(2)*depth(2)} * \ldots = Intensity_O * e^{\sum(-\mu(i)*depth(i))} \tag{2.2}$$



[x,y] [object] [depth]

[12,245] [gorilla] [16]
[12,246] [gorilla] [18]
[12.247] [gorilla] [19]
.
.
.

FIGURE 2.3. (a) The setting up of an X-ray system. Objects lie between x-ray focal spot and detector plane. (b) The expected output of this application. [x,y] represents the coordinate of pixels on the detector plane. [depth] represents the distance from the x-ray focal spot to the intersection of ray with respect to the object surface.

2.1.2 Human phantom

Human phantoms are the artificial models of human body commonly used for computerized analysis in the field of health-care (Fig.2.4). As the evolution of computer technology, phantom has developed to higher complexity, from representation by elementary geometrical shapes like ellipsoids and cylinders to voxelized model, and currently represented by more advanced mathematics such as Non-uniform rational B-spline (NURBS) and triangle meshes. Human phantoms have been developed to represent a wide range of humans, children and adults, male and female. The requirement for the precision of human phantoms has also grown since they are needed to perform test simulations of imaging systems and treatment procedures during the development phase. A human phantom composed by free-form and naturally curved surfaces is favored in almost all use cases.

Current X-ray simulation is commonly realized by rendering phantoms represented by voxelized model, which is time-consuming, has image artifacts and additionally needs huge memory to store its data. A ray tracing algorithm combined with NURBS human phantom can hopefully reduce memory usage, increase performance and image quality (i.e. produce natural and smooth object surfaces even in high resolution).

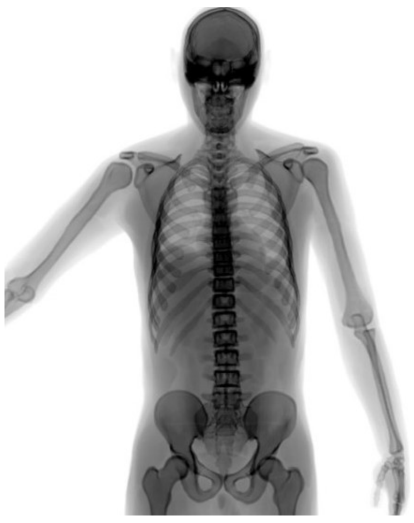


FIGURE 2.4. An example of human phantom scanned from patient[SW].

2.2 Phantom model representation

2.2.1 Voxelized model

A voxel is a unit of graphic information in 3D space that defined by a 3D coordinate and the information (color, intensity) at that coordinate. Voxels are commonly used for representing the output data of a Computed Tomography Scan (CT scan), Magnetic Resonance Imaging (MRI), or Ultrasound. Voxels was selected as the data model for human phantom mainly because of it is easy to be handled in software and almost all medical scans exports voxelized model data. Voxelized model can contain a large amount of data that specifies every coordinate in the 3D space (Fig.2.5), however adding up to a large amount memory at the same time. Data size of surfaces represented by two dimensions scales quadratically to resolution while that of voxels represented by three dimensions scales cubically to resolution. The other disadvantage is that everything in a voxelized model is blocky, because every voxel is actually a tiny cube. Therefore, it is almost impossible for it to represent a free-form and curved model at a high spatial resolution.

FIGURE 2.5. A game character represented by voxels

2.2.2 Triangle meshed model

In the field of game rendering scenes, triangle mesh is commonly used thanks to its simplicity. A triangle meshed object is a collection of triangles composed by several components (vertices, edges, faces). The traditional way to visualize a model would first tessellate the curved surface into triangle/triangle meshes in order to obtain a tight approximation of the surface. As shown in Fig.2.6(right plot), tessellation is actually approximating the surface by filling small triangles. The triangles generated by tessellation still occupy a large portion of memory when a high spatial resolution is required. Besides, artifacts and cracks may result from errors in approximation (Fig.2.7).

Triangle mesh is already a mature method and being widely used because of its flexibility in expanding and appending more details. The gaming industry already provided efficient code libraries for tracking. Additionally, the calculation with triangle is analytic and simple, with no iterative approach required. On the other hand, dynamic scaling (i.e. zoom-in on-the-fly) is difficult for triangle meshed scene because the fixed number of triangles indicates a fixed resolution. When there are no sufficient number of triangles, objects' surface would convert original curvatures in edgy and tiled looking segments of planes.

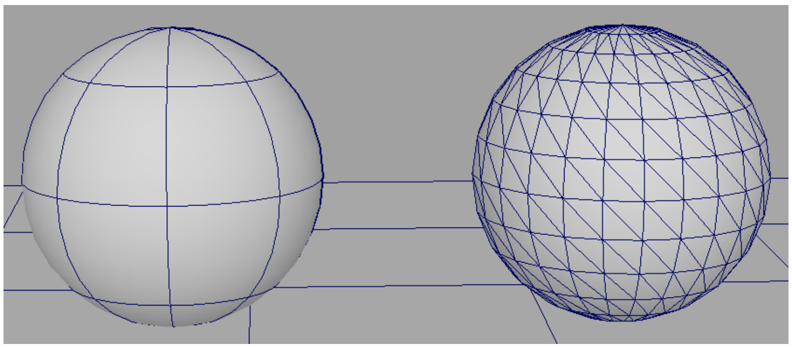


FIGURE 2.6. Sphere on the left is represented by NURBS, presenting free-form mathematical surface. Sphere on the right is tessellated, presenting surface composed by triangle segments.

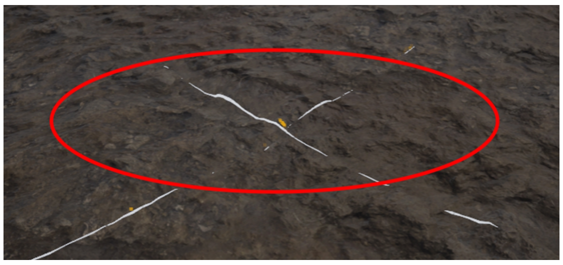


FIGURE 2.7. Cracks appear when modeling the ground in the scene with Unreal 4 game engine.

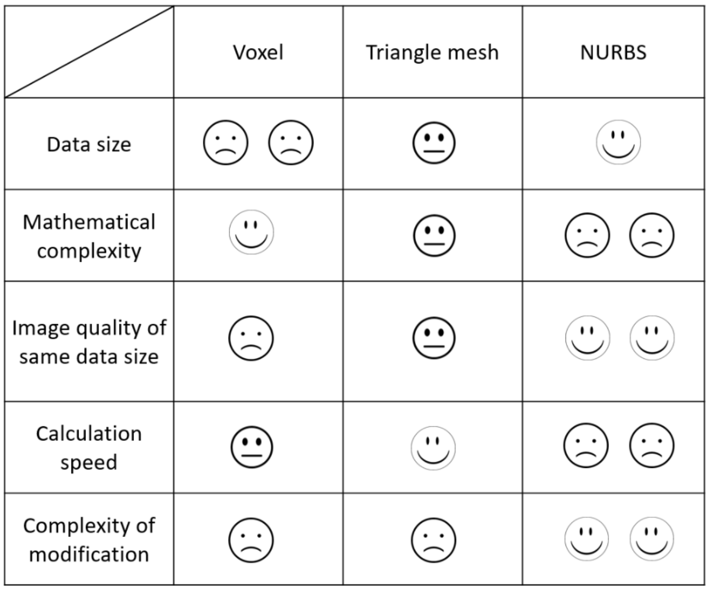
2.2.3 NURBS model

In the field of computer aided design (CAD), smooth free-form surfaces are commonly used to represent complex industrial design models such as automobiles and air crafts[PT12]. Human phantoms are also represented by them in the field of health care to obtain the highly precise and free-form models[SW]. Non-uniform rational basis spline (NURBS) surface is favored due to its compact representation and support for local control. With the upcoming of much higher computation power and algorithmic experience in data segmentation, NURBS is becoming a more promising option for its flexible usage of phantom. One of the advantages of NURBS is that modifications with NURBS model can be implemented much more easily and independently on spatial resolution. However, this is a complicated task for voxelized or triangle-meshed model. For instance, visualizing the narrowing of vessels in different resolution requires voxelized or triangle-meshed model apply changes constantly to all voxels/triangles. This is surely not a stable usage of phantom data. On the other hand, it is flexible for NURBS to change particular objects locally and combine them with the scene again.

2.2.4 Model comparison

In terms of a free-form surface, the number of triangles that represents the surface influences the error with the original shape. The error would be acceptable or even neglectable when there are sufficient triangles representing an object. In the case of ray tracing, the efficiency of representation by either NURBS or triangle mesh depends on the size of the image, the complexity of the object, and the tolerated error. In general, NURBS can store the information of objects more efficiently than triangle mesh, therefore lead to a smaller data size when comparable image quality. NURBS can always represent free-form surface while triangle mesh needs sufficient number of triangles to achieve it. The other advantage that NURBS has is its possibility of dynamic scaling (varying spatial resolution on-the-fly). Nevertheless, the high mathematical and computational complexity is a significant drawback that would lower the performance.

The artifact levels are reduced by using more triangles, while computational effort increases. It is unknown for now which one is the more efficient representation given a certain level for minimized artifact level and spatial resolution. The thesis will study the trade-off between those two representations and determine if NURBS are worthy to represent human phantom with expensive computation. Another possibility is combining the advantages of both approaches, to be specific, using NURBS to dynamically create triangle meshes on-the-fly with optimal spatial resolution. This method will overcome issues of static spatial resolution with triangle mesh and computation complexity with NURBS, and theoretically provides the best performance. Fig.2.8 demonstrates the three discussed data models' advantages and disadvantages from different perspectives.



	Voxel	Triangle mesh	NURBS
Data size	🙁 🙁	😐	🙂
Mathematical complexity	🙂	😐	🙁 🙁
Image quality of same data size	🙁	😐	🙂 🙂
Calculation speed	😐	🙂	🙁 🙁
Complexity of modification	🙁	🙁	🙂 🙂

FIGURE 2.8. Comparison among three different data models on different aspects.

2.3 NURBS basics

Implicit equation and parametric function are the two most common methods to represent curves and surfaces in the field of geometric modeling. Parametric function is preferred to represent free-form surfaces thanks to its intuitive geometrical properties[PT12]. the Non-Uniform Rational B-Spline surface or NURBS surface is the most used type of parametric free-form surface.

A NURBS surface $S(u,v)$ of degree p in u parameter direction and degree q in v parameter direction defined in the parameter domain $[a,b]\times[c,d]$ is represented in 3D(x,y,z) space by the following equation[PT12] Chapter 4:

$$S(u,v)=\frac{\sum_{i=0}^{n}\sum_{j=0}^{m}N_i^p(u)N_j^q(v)w_{i,j}C_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m}N_i^p(u)N_j^q(v)w_{i,j}} \tag{2.3}$$

where $a\le u\le b$, $c\le v\le d$, $C_{i,j}\in R^3$ are control points and $w_{i,j}\in R$ are weights assigned to their corresponding control points. The convex hull property is the most important property of the NURBS surfaces, the entire NURBS surface is enclosed in the convex hull of its control mesh formed by its $(n+1)\times(m+1)$ control points (Fig.2.9).

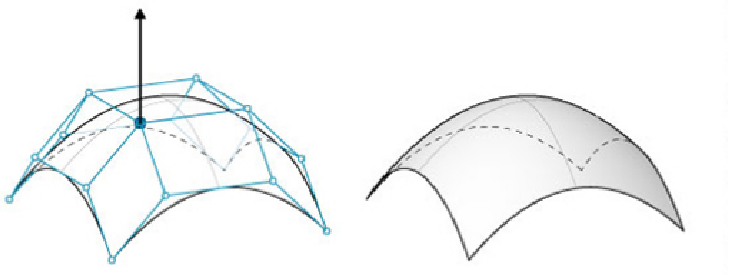


FIGURE 2.9. The original NURBS surface is on the right and its control mesh is shown on the left.

Two *knotvectors* ([PT12] Chapter2.2) corresponding to u and v directions define the NURBS surface as well:

$$\begin{aligned}r=n+p+1;s=m+q+1;\\ u_i\le u_{i+1}\text{ for }0\le i<r\text{ and }v_i\le v_{i+1}\text{ for }0\le i<s;\\ \vec{u}=\{u_0,u_1,...,u_r\}\text{with }u_i=a\text{ for }i\le p\text{ and }u_i=b\text{ for }i\ge r-p;\\ \vec{v}=\{v_0,v_1,...,v_s\}\text{with }v_i=c\text{ for }i\le q\text{ and }v_i=d\text{ for }i\ge s-q;\end{aligned} \tag{2.4}$$

where there are $p+1$ replicates of each of a and b respectively and $q+1$ replicates of each of c and d respectively. Additionally, $r=n+p+1$ and $s=m+q+1$. All the elements in the knot vectors are in an ascending sequence. Fig.2.10 shows the impact of different knot vectors on the same set of control points of a NURBS curve.

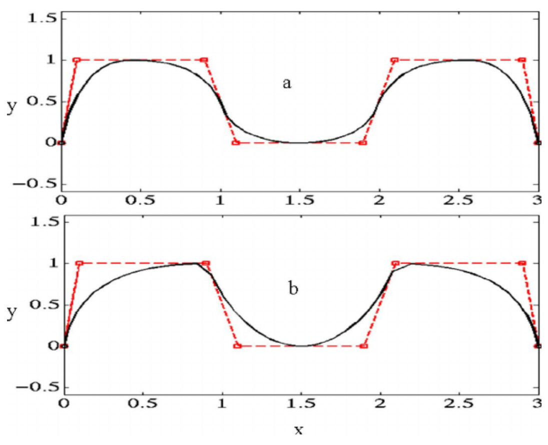


FIGURE 2.10. (a)The knot vector is {0, 0, 0, 0.2, 0.4, 0.5, 0.6, 0.8, 1, 1, 1}. (b)The knot vector is {0, 0, 0, 0.2, 0.2, 0.5, 0.8, 0.8, 1, 1, 1}. a produces smooth curve while b has sharp corner visible.

$N_i^p(u)$ and $N_j^q(v)$ are B-Spline basis functions of degree p and q, which are derived from knot vectors using the Cox-de Boor Recurrence formula. Equation.2.5 demonstrates the Cox-de Boor Recurrence formula for calculating $N_i^k(u)$, which is similar to $N_j^k(v)$. In the formula, k is iteratively incremented from $k = 0$ to $k = p$ or $k = q$. Furthermore mention that 0<=i<=r-p-1

(2.5)

$$N_i^k(u) = \begin{cases} 1 & \text{if } k = 0 \cap u \in [u_i, u_{i+1}) \\ 0 & \text{if } k = 0 \cap u \notin [u_i, u_{i+1}) \\ a_i(u) \cdot N_i^{k-1}(u) + (1 - a_{i+1}(u)) \cdot N_{i+1}^{k-1}(u) & \text{otherwise} \end{cases}$$

$$\text{where } \mathrm{a}_i(u) = \begin{cases} 0 & \text{if } u_{i+k} - u_i = 0 \\ \frac{u-u_i}{u_{i+k}-u_i} & \text{if } u_{i+k} - u_i \neq 0 \end{cases}$$

where $\vec{u}$ is the knot vector and $[u_i, u_{i+1})$ is the knot interval extracted from the knot vector. A none zero degree basis function is the linear combination of its two lower degree basis functions, and it is zero if $u \notin [u_i, u_{i+p+1})$. This means that each control point $C_{i,j}$ has local support on the domain $[u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$, moving the control points would not change the surface shape globally.

The first-order derivatives of surface points are essential for Newton's iteration (Section 2.4.1), and they are computed with the algorithm proposed by myself (See List.1) based on Equation.2.6[PT12]. Despite the fact that [PT12] already provides general algorithm to calculate derivatives of surface points of any order, it is complicated to be implemented and redundant when only the first-order derivatives are required in Newton's iteration.

(2.6)
$$N_i^k(u)' = \frac{k}{u_{i+k} - u_i} N_i^{k-1}(u) - \frac{k}{u_{i+k+1} - u_{i+1}} N_{i+1}^{k-1}(u)$$

```
float3 SurfaceDerU(float u, float v, size_t index_object)
{
//initialize basis functions for u and v
float Nu[10];
float Nv[10];
//calculate basis functions for u and v
unsigned int uspan = FindSpan(number_points_n, degree_p, u, knot_u);
DerBasisFuns(uspan, u, degree_p, knot_u, Nu);
unsigned int vspan = FindSpan(number_points_m, degree_q, v, knot_v);
BasisFuns(vspan, v, degree_q, knot_v, Nv);
//iteratively sum up to 1st order derivative on u
unsigned int uind = uspan - degree_p;
float3 surface_der_u = optix::make_float3 (0);
for(int i = 0; i <= degree_q; i++)
{
unsigned int vind = vspan - degree_q + i;
for(int j = 0; j <= degree_p; j++)
{
surface_der_u=surface_der_u+Nv[i]*Nu[j]*control_points[(uind+j)*number_points_m+vind];
}
}
return(surface_der_u);
}
```

Listing 1: The function that calculates the first-order derivatives of surface points on u. The calculation on v is similar.

2.4 Ray tracing

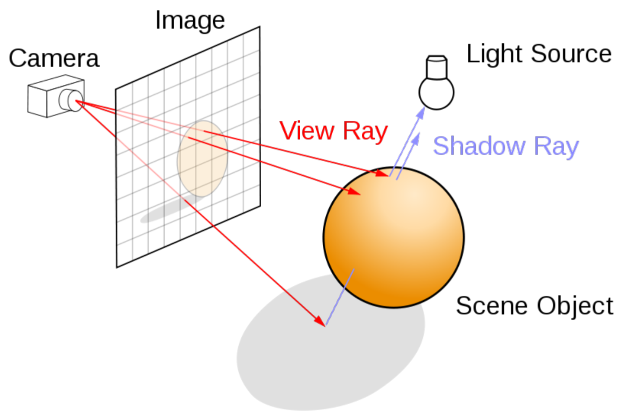
2.4.1 Ray tracing for visible optics

Since X-rays are related to visible light (both is physically described by electromagnetic radiation), it is worth looking into the much more common usage of ray tracing for visible optical light simulations. Ray tracing is a relatively simple algorithm, which can generate an image by tracing the path of light from light sources and simulate visual effects when encountering objects in between. By ray tracing, pixel values can be calculated directly from the model, through which the memory usage is reduced and artifacts are avoided (Fig.2.12).

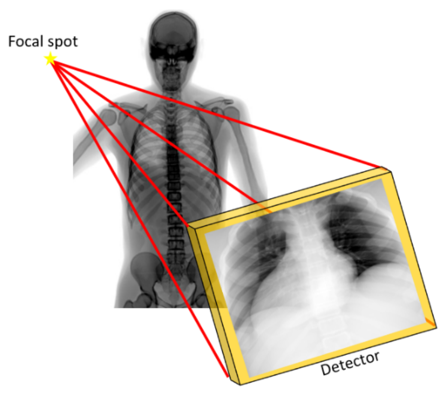
The principle of ray tracing is shown in Fig.2.11(a). The color of each pixel is determined by shooting a primary ray from the camera passing the pixel and calculating the nearest intersection with an object. If no object appears as an obstacle on the path of the ray, the color of the corresponding pixel will be set to the default background color. If the ray intersects an object, the color of the corresponding pixel will be calculated by spawning secondary rays. For a realistic image, the simulation of interface physics (reflection and refraction) is mandatory.

2.4.2 Ray tracing for X-ray

Nevertheless, tracing X-rays is different from tracing common optical rays. First of all, an X-ray will not spawn any secondary X-ray when encountering a surface. (X-ray scattering might occur in the volume. However, its treatment is not part of this thesis due to its physical complexity.) Instead, it transmits straight forward through the whole object (see Fig.2.11(b)). Therefore, X-ray tracing does not generate any significant visual effects like reflection or refraction. The other difference is that the information of depth is the key in X-ray tracing as discussed in X-ray system, in which depth is the distance that a X-ray travels on its path inside an object. Eventually, the depth list for each pixel in the detector plane will be used to generate a depth image. Equation.2.1 already shows the definition of X-ray attenuation and Equation.2.2 explain the case of multiple overlapping objects, indicating that segment lengths (depths) along a straight trajectory are needed.



(a)



(b)

FIGURE 2.11. (a) The principle of ray tracing in optical world. (b) The X-ray tracking in the X-ray world.

2.5 Ray tracing NURBS

Since ray tracing does not require extra information and preprocessing, NURBS surface can be directly traced. However, NURBS represents free-form looking curved surfaces with direct mathematical formulas and ray tracing would need a mathematical formula to find intersection between ray and surface. It turns out that an inverse calculation of the formulas (i.e. a derivation of the u and v from a given (x,y,z) point in space) is analytically not possible. With such mathematical complexity, the current approach is iterative via root finding. Using this method, the rendered image can be visualized without any artifact (see Fig.2.12)., however requires a huge computation power. This process can be speed up by two approaches: reducing the number of ray-patch intersection calculation and optimizing calculation of intersection. Such two approaches together with acceleration data structure are discussed in the following sections.

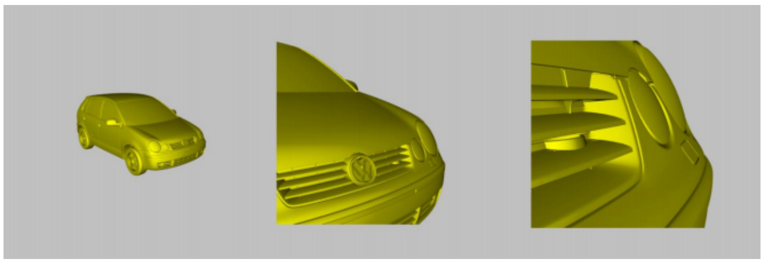


FIGURE 2.12. A model of Volkswagen POLO from different viewing distances. The details of the model remains perfectly curved due to direct ray tracing of NURBS. This figure is retrieved from [AGM06] and [Val10].

2.5.1 Ray-patch intersection

The most important part of ray tracing NURBS surface is an intersection test between a ray and the surface. For standard primitives like triangles and quadrilaterals, the intersection calculation can be done analytically with low computational effort. However, for NURBS, the analytic formula to solve the intersection calculation is too complicated to be implemented and the method for fast intersection test is still open. The numerical method is discussed below:

Numerical method The approach is the numerical method to approximate the intersection iteratively, also known as Newton's iteration [PTVF96]. This method iteratively finds better approximation of the root of a non-linear function. For each subsequent iteration, the error between the approximation and the actual root decreases quadratically (Fig.2.13) and the computation of each iteration only depends on the previous one. The algorithm needs the function, its derivative and an initial guess as the inputs (Equation 2.7). Moreover, a good initial guess that is close enough to the actual root is extremely important for Newton's iteration converging to the correct solution.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2.7}$$

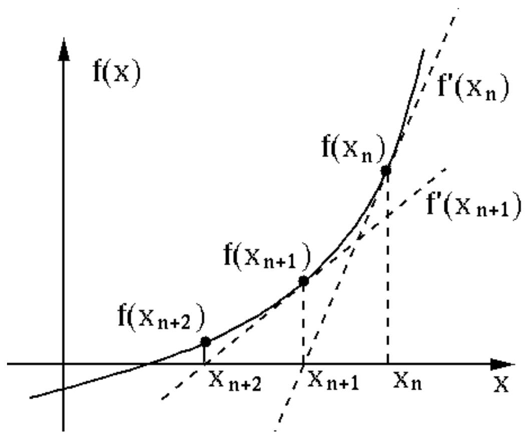


FIGURE 2.13. In Newton's iteration, approximation become closer to the actual root in each subsequent iteration.

Error measurement in Newton's iteration The idea of calculating of error function in Newton's iteration approach refers to [NSK90] as shown in Fig.2.14. The author devised that every ray is represented as the intersection of two orthogonal planes. In each Newton's iteration, a candidate point is proposed by the algorithm. The error function of such point between the actual root is calculated by summing the distances from the candidate points to both orthogonal planes. Then, the error function will be used as the reference to calculate the candidate point for the next iteration.

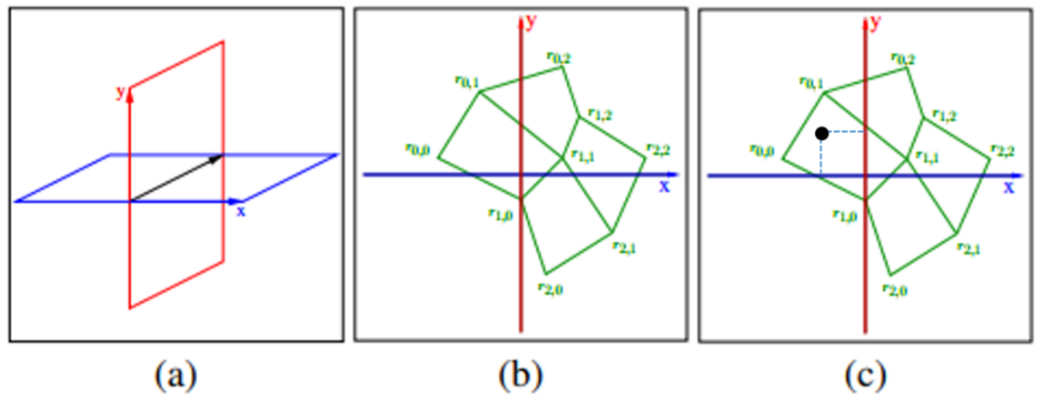


FIGURE 2.14. Figures retrieved from [EHS05]. (a) The ray is represented as the intersection of two orthogonal planes. (b) The topview along the X-ray trajectory. (c) The error function of a candidate point is calculated by summing the distance towards both planes.

2.6 General-purpose GPU computing

The performance of graphics processing unit (GPU) has significantly grown during recent years. Modern GPUs provide not only computation parallelism in multiple processing cores, but also offer a considerable bandwidth, can be up to 600GB/s or more. A huge amount of GPU memory (up to 12GB) together with such high bandwidth can provide a dramatic computation power[Nvi]. Fig.2.15 gives the performance trends of latest GPUs and CPUs. The main reason for fast evolution of GPUs is that they are specially designed for highly parallel computation and has a more simplified processing model than typical CPUs (Fig.2.16)

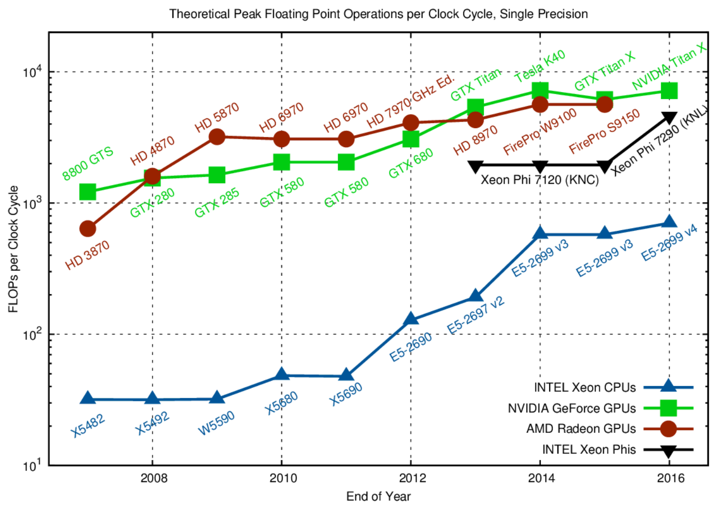


FIGURE 2.15. Performance trend of latest CPUs and GPUs[Rup]

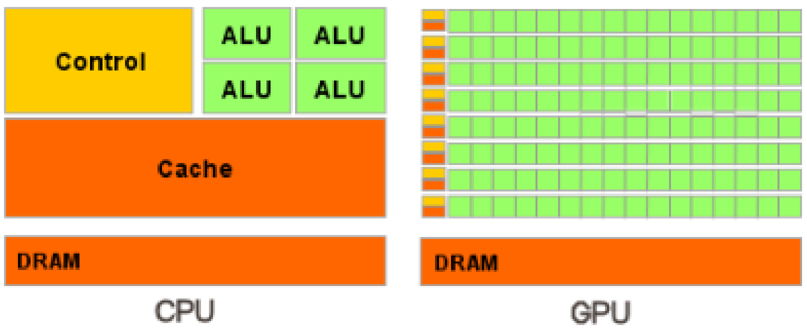


FIGURE 2.16. GPU has a simplified processing model.

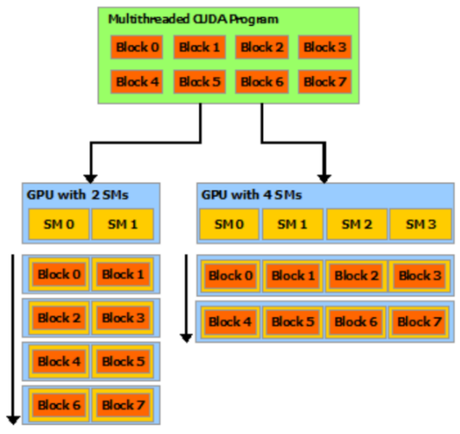
A GPU is especially suitable for applications that require the same set of operations for each data point. With single instruction multiple data (SIMD) style, the requirement for flow control is lower and the memory access latency can be hidden with calculations instead of big data caches. A GPU can be programmed by using special programming language or API such as OpenGL, DirectX, CUDA. CUDA are considered to be used in this thesis due to the wide range of GPUs it supports and the functions it contains.

2.6.1 CUDA programming model

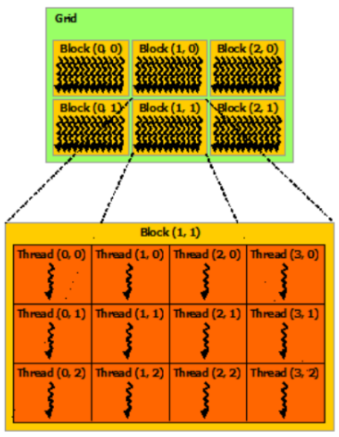
A GPU is built around an array of Streaming Multiprocessors (SMs). A multithreaded program is partitioned into blocks of threads that execute independently from each other, so that a GPU with more multiprocessors will automatically execute the program in less time than a GPU with fewer multiprocessors.

The key of the CUDA parallel programming model is a hierarchy of thread groups that are simply exposed to the programmer as a minimal set of language extensions. The hierarchy provides fine-grained data parallelism and thread parallelism, nested within coarse-grained data parallelism and task parallelism. They guide the programmer to partition the problem into coarse sub-problems that can be solved independently in parallel by blocks of threads, and each sub-problem into finer pieces that can be solved cooperatively in parallel by all threads within the block.

Blocks are respectively organized into a one-dimensional, two-dimensional, or three-dimensional grid of thread blocks (2.17(b)). The number of thread blocks in a grid is usually dictated by the size of the data being processed or the number of processors in the system, which it can greatly exceed.



(a)



(b)

FIGURE 2.17. (a) A compiled CUDA program can execute on any number of multiprocessors, concurrently or sequentially base on schedule of blocks of threads. (b) The hierarchy of grid, block and thread.

2.6.2 Optimization with GPU

The ray tracing algorithm is perfectly suitable to be implemented on GPU because it can be fully adapted with GPU parallelism. For example, the tracing of a beam consisting of rays from source to each detector pixel, i.e. $N_x * N_y$ rays, can be distributed on each core of GPU and executed by parallel. Since there is not data dependency among rays' tracing, this approach is already much more effective than a CPU version.

2.7 Related Work

This section includes a literature review on topics which are not further used in the main part of the thesis, but nevertheless are noted to get a better understanding. Those related works focus on ray tracing algorithm and NURBS surface, and some of them include GPU usage.

State of the Art in Ray Tracing Animated Scenes[WMG$^+$09] This paper is useful for determining which acceleration data structure to be used although it is not applied in this thesis. It discusses the general design decisions in ray tracing algorithms and their trade-offs of acceleration data structures with different types of scenes (static or dynamic), however with respective of surfaces composed of triangles. The general idea is compatible for NURBS.

Efficient Ray Tracing of Trimmed NURBS Surfaces[Efr05] The trade-off between performance and image quality of the two root-finding approaches could be the reference of determination of the root finding method in my thesis. The author presents two influential methods to find root of intersection between ray and rational Bezier surface: Bezier clipping and Newton's iteration. It proposes an efficient choice for termination criteria for Bezier clipping, which makes the algorithm more intelligent. However, both of its approaches require pre-transformation from NURBS to rational Bezier surface due to its faster evaluation routines. Hierarchical axis aligned bounding boxes are used as bounding volume for acceleration data structure. Its test shows that Newton's iteration is faster than Bezier clipping, however is unacceptable for complex scenes because it produces image artifacts.

Interactive Ray Tracing of NURBS Surfaces by Using SIMD Instructions and GPU in Parallel[Abe05] This thesis discusses about the memory layout for SIMD and GPU use, which could be useful for my GPU implementation. Its tests show that ray tracing a non-rational variant is faster than ray tracing a rational one. Moreover, the author compared the results on different PCs and different bounding box hierarchy creation strategies. This would be a nice reference for us to choose how to construct acceleration data structure.

Ray Tracing NURBS Surfaces using CUDA[Val10] This thesis is the first one directly ray tracing NURBS using CUDA, providing some effective techniques, among which measurement of errors in Newton's iteration is applied in my thesis. Instead of tracing rays sequentially, the author traces a packet of rays each time to take the advantage of memory coherence among rays. He also developed a CPU/GPU hybrid variant, which accelerating the primary intersection stage, which is highly relevant to our topic. Tiling is applied as the technique for efficiently using GPU memory by dividing images into evenly-sized rectangular pixel blocks, which can be taken as a optimization technique for our GPU implementation. Results show that the version of CPU/GPU hybrid by tracing packets of rays with cache has the best performance with primary rays in low-complexity images.

Acceleration Data Structure Construction for Ray Tracing[Vin13] Construction of acceleration data structures both on CPU and GPU is discussed in the paper, which is helpful for future implementation of acceleration data structure in my topic. This is a thesis proposal that specializes in acceleration data structure construction, however its follow-up thesis is not found. It deeply discusses three most influential acceleration data structures: grid, kd-tree, bounding volume hierarchy.

Interactive Rendering of NURBS Surfaces[CAPD14] A new approach to refine NURBS surfaces is proposed in the paper despite the fact that refinement and subdivision is not applied in my thesis. Intersections of smaller NURBS surfaces after refinement and subdivision can be faster calculated. An interesting concept called KSQuad is introduced in this paper to replace the knot refinement technique (NURBS surface are decomposed into rational Bezier patches) as the partitioning technique for subdividing NURBS surface. KSQuad requires less memory usage and can achieve interactive speed of deforming NURBS surface, which is suitable for GPU implementation. It is valuable when partitioning NURBS is needed in our thesis.

Direct and Fast Ray Tracing of NURBS Surfaces[AGM06] Replacement of formulas with SIMD instructions is helpful for implementation on CPU since only a limited number of SIMD instructions are available in CUDA. The most inspiring point of this paper is that the author has found out a way to rewrite the Cox-de Boor recursion formula2.5 into a more efficient and memory-friendly form using SIMD instructions. The co-efficient in the formula are called CBDItem which is dealt in the preprocessing stage.

Practical Ray Tracing of Trimmed NURBS Surfaces[MCFS00] Another subdivision method is proposed in the paper to help Newton's iteration converge faster, which is constructive for future development. The author employs a flattening step to subdivide the control mesh such that each knot span meets some flatness criteria, by which the Newton's iteration can converge swiftly. Additionally, their bounding volume hierarchy is built based on the newly refined control mesh generated in the flattening step. Their implementation have achieved interactive frame rates with scenes with moderate complexity.

Robust and Numerically Stable Bezier Clipping Method for Ray Tracing NURBS Surfaces[EHS05] The alternative of Newton's iteration called Bezier clipping is proposed here. Part of the approach is deployed in the error measurement of Newton's iteration in my thesis. The author of this paper choose to transform NURBS into rational Bezier patches in the preprocessing stage due to its faster evaluation routine. The Bezier clipping method is introduced to help speed up the root finding of intersections. Rays are represented by intersection of two orthogonal planes and error can be easily calculated from that.

Chapter 3

Development

Two new methods regarding realization of X-ray simulation with NURBS object are proposed in this chapter. The first method applies techniques related to ray tracing NURBS (discussed in related work) for reference, realizing direct X-ray tracking NURBS object. However, it involves a huge amount of computation for intersection calculation between each ray and the surface of the object, due to the complexity of NURBS. The second method is a fully innovative approach that reduces the amount of intersection calculations and achieves the flexibility to adjust image quality. This approach is essentially an approximation of the original object by interpolation among sampled surface points, which has two main procedures: projection and rendering. It can effectively cooperate with GPU thanks to the high parallelism in the algorithm.

3.1 X-Ray tracking NURBS surface with Newton's iteration

3.1.1 Introduction

As discussed in Section 2.5.1, even the equations of the rays from X-ray source to detector and the mathematical definitions of the NURBS surface are known, no direct mathematical approach (analytic formula) is available since 3rd-degree polynomials is not practically invertible. Thus, the algorithmic approach (Newton's iteration) is taken. In this section, the approach using a conventional ray tracing technique combined with Newton's iteration root finding is proposed. Since the output image quality is vital in this thesis, the advantage of ray tracing that produces realistic image quality is considered. However, the calculation of intersection between ray and NURBS surface could be computationally expensive and become the bottleneck of the entire application. After investigation, OptiX developed by Nvidia[Opt], a ray tracing framework cooperating with Nvidia GPU, is used to avoid redundant coding.

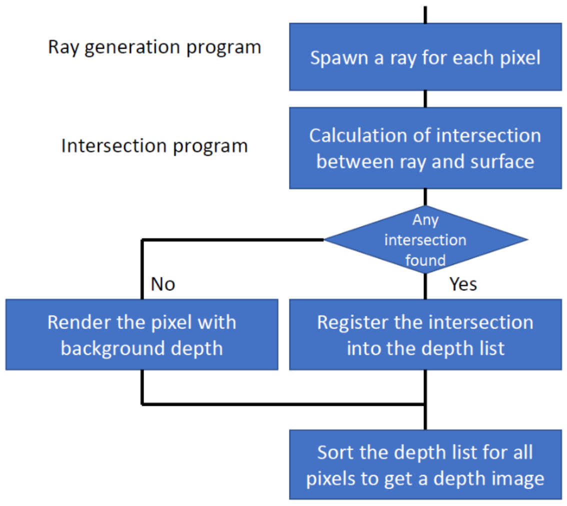

Ray generation program
Spawn a ray for each pixel
Intersection program
Calculation of intersection between ray and surface
Any intersection found
No
Yes
Render the pixel with background depth
Register the intersection into the depth list
Sort the depth list for all pixels to get a depth image

FIGURE 3.1. The pipeline of X-ray tracking algorithm using Newton's iteration.

Virtually, ray generation program, intersection program, any-hit program and miss program of OptiX framework[Opt] are deployed in our algorithm. For example, closest hit program is not necessary since it is only effective when it comes to an optical application like a game engine. Otherwise, finding the closest intersection is trivial. Programs and procedures for traversal are not fully used because every provided NURBS object is represented by a whole NURBS patch. It is only worthy to spend effort to take advantage of traversal acceleration when the intersection calculation does not limit the performance. The pipeline of this approach is shown in Fig.3.1, in which there are four steps.

The pipeline for the X-ray tracking algorithm is relatively simple: the ray generation program is called to spawn a ray for each pixel of the detector plane, variables like ray direction and camera setting are calculated.

Next, the intersection program is invoked to calculate the intersection between ray and NURBS object. By applying Newton's iteration root finding, an intersection is normally calculated in at most three iterations. However, there must be an essential prerequisite: a reasonable initial guess as the input for Newton's iteration. The detail of how the initial guess is obtained is described in the next section. The surface points are computed according to the algorithms in [PT12] Chapter 2 including the knot span finding and basis function calculation.

If no intersection is found, the miss program is called to render the pixel with the background depth. Otherwise, the any hit program is called to register the depth of the intersection (distance between X-ray source and the intersection) into the depth list of the corresponding pixel. Fig.2.2 provides an example of overlapping objects, which is the reason why a depth list is necessary. Such a pipeline is executed on GPU in parallelism for every pixel.

Eventually, after all pixels on the detector plane are rendered (all intersections' depth are registered into depth lists), a sorting algorithm is applied in order to generate an sorted depth map. The organized depth map includes every pixel's depth list, and the depth in the list is sorted in a ascending order, with a pointer to indicate which object the intersection is entering or exiting. An simulated X-ray image can be generated with the sorted depth map using the existing X-ray simulation software of Philips.

3.1.2 Initial guess finding

For Newton's iteration method root finding, a reasonable initial guess is a must. The definition of "reasonable" here is that the guess should have an acceptable error with actual root so that the Newton's iteration could converge. The error between the ray and the initial guess is measured with the technique in [EHS05] and [Val10], representing a ray by two perpendicular planes and calculating the distance of the initial guess and those two planes respectively (see Section 2.4.1). Such error measurement is applied in the Newton's iteration as well.

Currently, the initial guess is obtained by spatial-equally sampling 100*100 surface points on the UV region and picking the least-error one. Fig.3.2 demonstrates a mesh composed of sampled surface points that equally distributed in the UV parametric region. By sampling the points on the mesh and calculating their errors to the ray, the point with least error will be used as Newton's iteration initial guess.

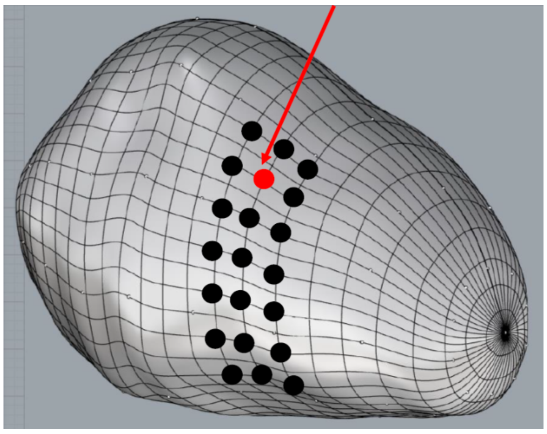


FIGURE 3.2. Sampling over the object surface to find out the initial guess. Red arrow represents the ray and black dots represent the sampled surface points in the region. The algorithm calculates the errors for all black dots, and use the red dot (the least error one) as the initial guess for Newton's iteration.

Nevertheless, even sampling 100*100 surface points still cannot provide a reliable guess. It is mainly because of the complexity of the human body organs formed by the NURBS surfaces. First, the function of NURBS surface is two-dimensional based on U and V. Second, the function could be non-linear as shown in Fig.3.3 and it is almost impossible for Newton's iteration to converge to a root of a close-spaced local function where minimum and maximum locate close to each other. One solution would be increasing sampling density to obtain a more accurate initial guess. This will definitely increase the workload of GPU and further lower the performance.

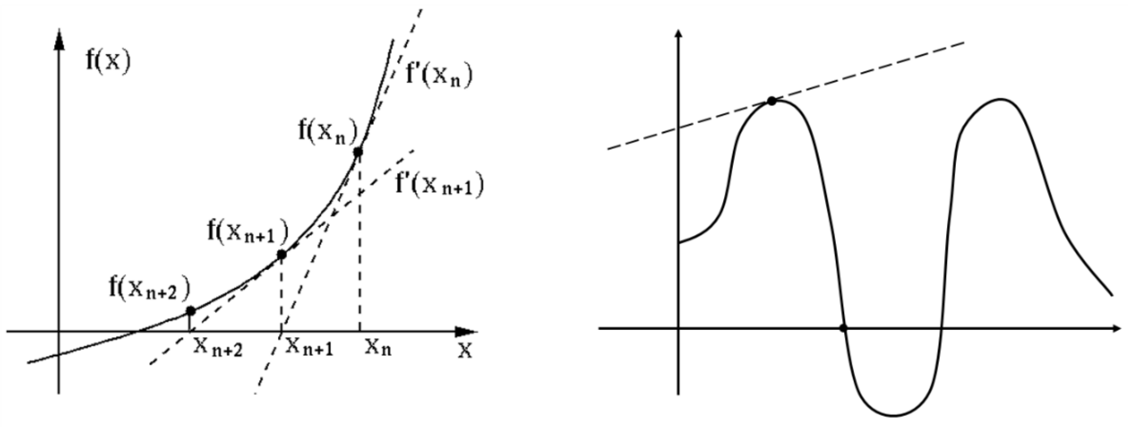


FIGURE 3.3. The figure on the left shows a normal linear function that Newton's iteration can easily converge in two or three iterations. The figure on the right shows a possible non-linear function of NURBS surface. Even when the initial guess is close to the actual root, the Newton's iteration can hardly converge to it.

3.1.3 Discussion

This approach is highly restrained by initial guess finding. First, Newton's iteration only supports single-root finding, which means it converges to only one intersection even when a ray hits multiple objects. Therefore, a correct number of initial guesses exactly as the number of intersections should be provided, in order to calculate all the intersections. Such a problem makes the X-ray tracking even more complicated unless the NURBS object is partitioned into patches to ensure only one intersection exists for a ray and a patch. Second, obtaining the initial guess costs a large portion of computation power. [Val10] used the center of the bounding volume from his acceleration data structure as the initial guess for Newtons' iteration and provided a decent accuracy. Such method will also require the algorithm to preprocess NURBS data such as partition of objects and construction of a acceleration data structure. Due to limited time, the version without adaption with preprocessing will be used for comparison with the other approach.

3.2 Direct mathematical projection of NURBS surface

3.2.1 Introduction

In this section, a new approach is discussed which is completely different with the previously discussed methods regarding intersection calculation. The idea is to generate a depth map as a mathematical projection directly on the detector plane for a single NURBS object, in which every pixel contains a list of depth of intersections that a ray hits on its path.

The first step of such approach is sampling 2D points by sampling 3D surface points equally in the object's parametric region and projecting them onto the detector plane as 2D points (Fig.3.4), e.g. 100*100 sampling density in the $[0.0, 1.0] \times [0.0, 1.0]$ UV region. Each projected 2D point contains information of the depth of the sampled surface point.

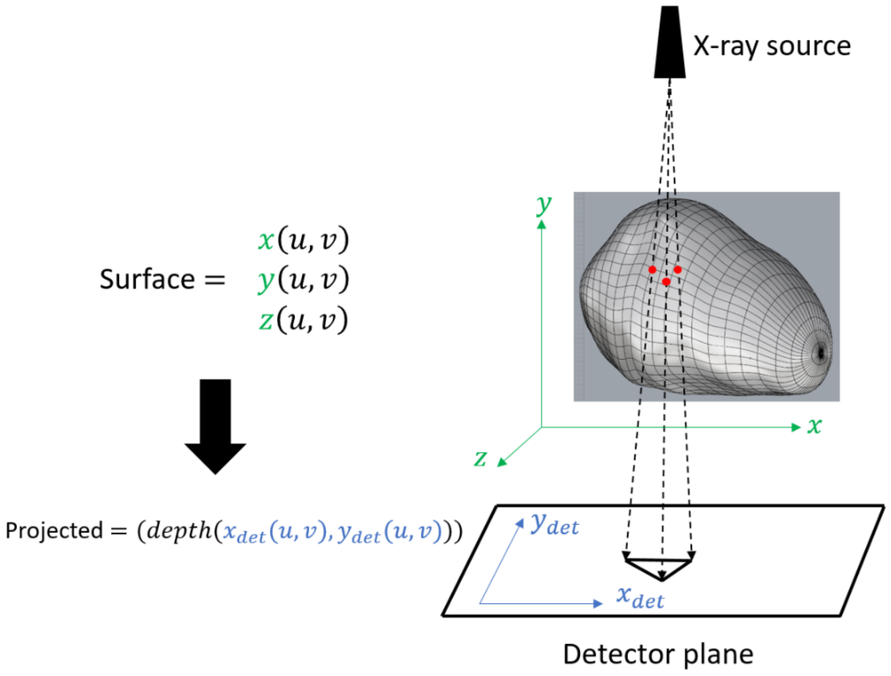

X-ray source
Surface = $x(u,v)$ $y(u,v)$ $z(u,v)$
y
x
z
Projected = $(depth(x_{det}(u,v), y_{det}(u,v)))$
y_{det}
x_{det}
Detector plane

FIGURE 3.4. Projection of 3D formula into 2D formula on the detector plane.

The second step is to generate a depth map for each detector pixel by interpolating the depth of neighboring projected 2D points. Those points may irregularly and non-equidistantly distribute on the detector plane even they are sampled equally in the object's parametric region. Thus, for each pixel, several projected points may be located around the center of it. And the depth of this pixel is calculated by interpolating those neighboring projected 2D points. Fig.3.5 shows an example of a 4*4 pixel-size plane, on which pixel centers resides inside the triangles formed by neighboring projected 2D points. The other possibility is that multiple pixel centers reside in the same triangle region depending on the sampling density and the detector plane size.

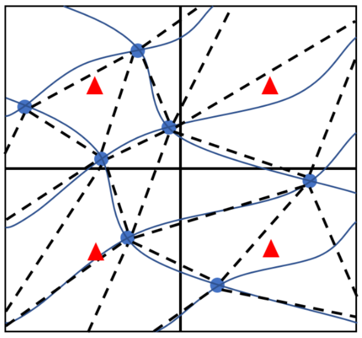


FIGURE 3.5. Top view of interpolation that renders pixels' value on a 4*4 pixel size plane. Red triangles are centers of pixels and blue dots are projected points located on the surface represented by blue curves. A mesh is constructed by connecting projected points. The pixel will be rendered by interpolation if the center of it is located inside of the triangle formed by neighboring projected points.

3.2.2 Pipeline

In this section, an execution pipeline for the X-ray tracking algorithm using direct mathematical projection is proposed (see Fig.3.6). The setting of X-ray source and detector plane as well as the parameters of NURBS object including knot vectors, control points, degrees are loaded before the pipeline.

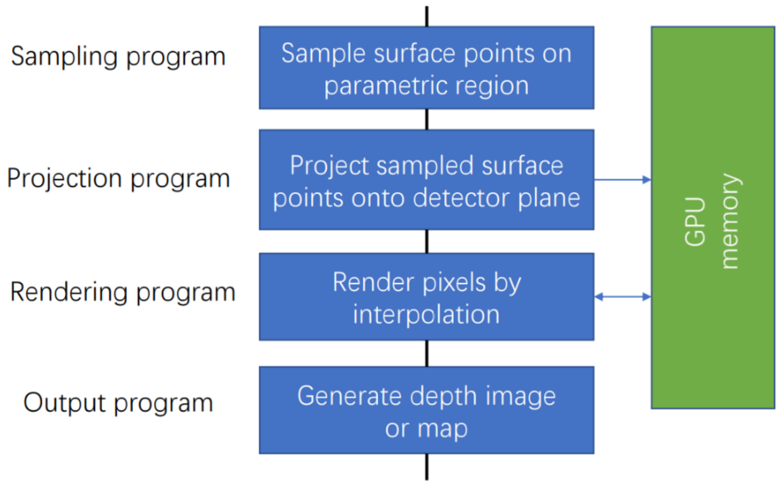

Sampling program
Sample surface points on parametric region
Projection program
Project sampled surface points onto detector plane
Rendering program
Render pixels by interpolation
Output program
Generate depth image or map
GPU memory

FIGURE 3.6. Overview of the algorithm stages and their relation to GPU.

A data structure called *sampled_payload* (see List.2) is responsible of carrying all the necessary data of each sampled surface point through the pipeline and it is transferred to GPU as well. Two essential parameters are contained in such a structure: *projected_coordinate*, the coordinate of a sampled surface point projected on the detector plane; $depth$, the distance between the X-ray source and the sampled surface point.

```
struct sampled_payload{
     float3 projected_coordinate;
     float depth;
}
```

Listing 2: Definition of $sampled\_payload$ struct.

The sampling program calculates the surface points in the object's parametric region based on the arbitrary sampling density, which further influences the rendering process. Here the advantage of NURBS is taken: the sample density can be dynamically (on-the-fly) defined such that the projected points can already get a suitable distance from each other on the detector plane. The surface points are computed according to the algorithms in [PT12] Chapter 2 including the knot span finding and basis function calculation.

The projection program is the key of such approach since it maps surface points onto the detector plane based on sampling in the parametric region. An array of 2D projected points with corresponding depth is generated from the projection program and then transferred to GPU.

The rendering program is regarded as the kernel of GPU execution which searches triangle regions formed by three consecutive projected 2D points from the array, for instance array[i,j], array[i+1,j], array[i,j+1], checking if there is any pixel center residing in such a region. If no, the pixel will not be rendered. If yes, it will be rendered the interpolation value among those three points' depth and this intersection is added to a list of already calculated depths of other layers. After all triangle regions from the array are searched and corresponding pixels are rendered, an output array containing all pixels' depth lists is transferred back to the output program. For example (see Fig.3.7), 2D projected points predefined as P (i,j) corresponding to UV pairings (u_i, v_j) are stored in an array and triangle is assigned to (i,j) by the points P (i,j), P (i-1, j) and P (i, j+1). In the case of Fig.3.7, it is P (3,1), P (3,2) and P (4,1). For a detector plane of 18*12 pixels size, there are three pixel centers that reside in the region. Eventually, they are rendered with corresponding 2D projected points' depth and the depth will be added into the depth list of each pixel.

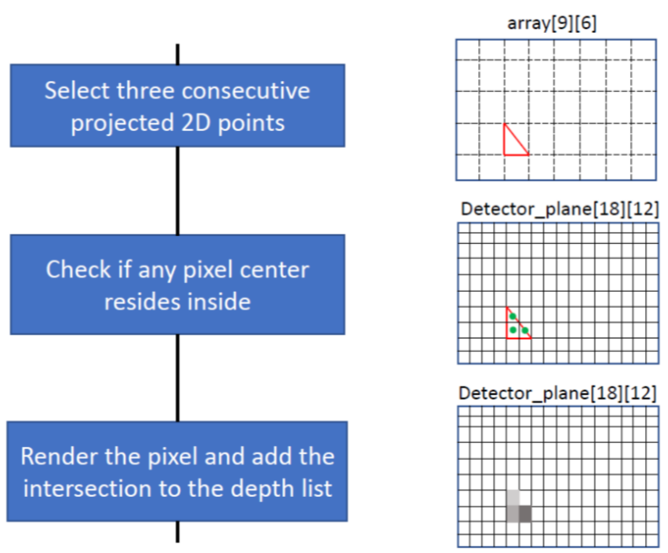

Select three consecutive projected 2D points
Check if any pixel center resides inside
Render the pixel and add the intersection to the depth list
array[9][6]
Detector_plane[18][12]
Detector_plane[18][12]

FIGURE 3.7. An example of rendering program searching the triangle region formed by three projected 2D points.

With the output array, the output program applies a sorting algorithm to generate an organized depth list for each pixel which indicates all intersections' depth and whether an intersection is entering an object or exiting an object.

3.2.3 Discussion

This approach is an approximation of the NURBS surface projected on the detector plane. The sampling density of surface points directly influences the accuracy of the approximation, which enables the approach to be flexible to adjust the image quality. The other advantage is that the computation of intersection between ray and NURBS surface in a conventional ray tracing approach is avoided, therefore the performance is highly increased. The bottleneck of the approach could be the computation in the rendering program that checks whether pixel centers reside in the triangle region. The sorting algorithm could be another potential bottleneck, because the size of the depth list for each pixel must scales with the number of objects in the scene and the complexity of the scene (objects overlapping randomly each other). The bottlenecks are not issues with image quality but only with performance. In addition, memory usage would not be the concern either for the rendering program or the sorting algorithm. The rendering program only accesses three elements of the projected points every time. For the sorting algorithm, the maximum number of objects in the depth list of each pixel could be 50, and the total memory usage would not exceed 500MB.

Chapter 4

Results

In this chapter, the run-time performance and output image quality of the X-ray tracking algorithm in different approaches will be tested and compared.

4.1 Experimental Setup

In the tests, two different approaches are investigated, namely X-ray tracking NURBS surface with Newton's iteration and direct mathematical projection of NURBS surface. The results will be compared to the conventional X-ray simulation using voxelized model and the popular ray tracing using triangle mesh. In total, there will be four different approaches to be tested. Experiments using same scene setting and hardware configuration will be performed.

4.1.1 Test platform

The operation system for testing is Window 10 Enterprise 64bit with Microsoft Visual Studio 2017 as the IDE. Intel Xeon W-2125 Processor is used as the CPU and NVIDIA Quadro P5000 is selected as the GPU for experiment due to its large amount of CUDA cores and memory. Table 4.1 shows more specifications of the GPU.

	Quadro P5000		Xeon W2125
Compute Capability	6.1	Cores	4
CUDA Cores	2560	Threads	8
Core Clock	1.73GHz	Base frequency	4.00GHz
Memory	16GB	Cache	8.25MB
Memory Bandwidth	288GB/s	Memory Bandwidth	85.3GB/s
CUDA Driver	9.3		
CUDA Toolkit	9.3		

TABLE 4.1. The technical specifications of NVIDIA Quadro P5000 GPU and Intel Xeon W-2125 Processor.

4.1.2 Test scene

The raw data of NURBS objects is provided by Duke University[SW], in which a heart model (dias_pericardium.obj) and a human phantom (male_pt146.obj) are considered to be the test objects. The test scene is composed of X-ray source, detector plane, and the test object. The rotation and relocation of X-ray source and detector plane creates a different perspective of the output image. Fig.4.1 and Fig.4.2 demonstrates the NURBS heart model and the NURBS human phantom visualized in Rhino (a 3D modeler software[Rhi]) respectively. The NURBS degree of both objects is three here. The triangle meshed model data is transformed from NURBS data via Rhino. The specifications are shown in Table4.2, NURBS has much smaller data size to represent the models compared voxels and triangles.

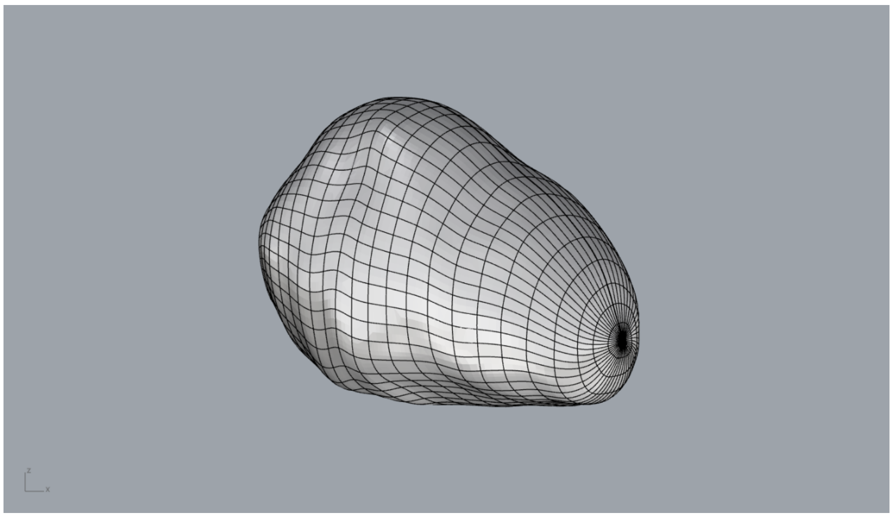


FIGURE 4.1. The heart model (dias_pericardium.obj) rendered with Rhino.

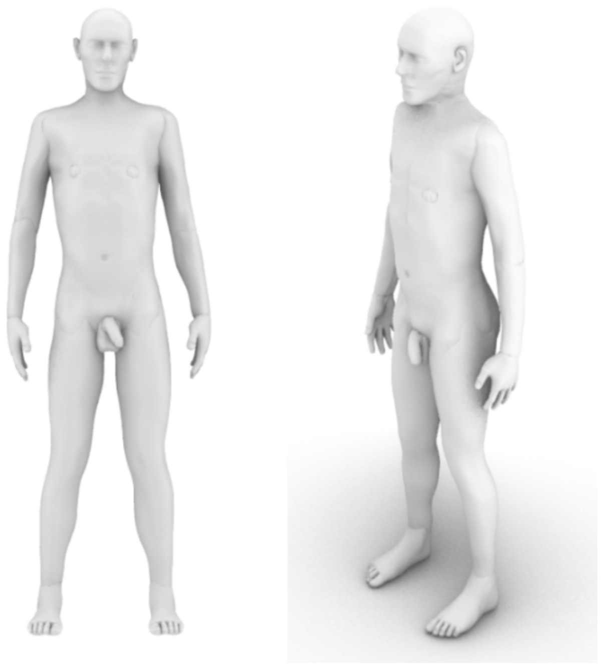


FIGURE 4.2. The human phatom (male_pt146.obj) rendered with Rhino.

	dias_pericardium (heart)	male_pt146 (whole human body)
Number of NURBS patches	1	588
Number of triangles	288/780/2600	3974143
Number of voxels	200 × 200 × 200 (0.5mm)	267 × 1062 × 1767 (1mm)
NURBS data	44KB	61.79MB
Triangle meshed data	13KB/32KB/103KB	337MB
Voxel data	15.26MB	1.8GB

TABLE 4.2. The two test scene objects: dias_pericardium and male_pt146.

4.2 Image Quality

Currently, there are no standard X-ray images from the same NURBS objects as the reference. Therefore, the image quality can not be quantified using methods such as image subtraction and comparison of histogram. The artifact level of the output image remains to be the only option now for analyzing image quality. The artifact level includes the occurrence of unsmoothness, curvatures and blocky edges. Eye vision is applied in order to examine those artifacts, despite the fact that different people might have different opinions regarding artifacts.

4.2.1 X-ray tracking voxelized model

Since there is no voxelized model data of the heart model available, the test of image quality is performed by simulating the region of the heart in a voxelized phantom as shown in Fig.4.3. Each voxel has the volume of 0.5mm*0.5mm*0.5mm. After enlargement of the region of edges and borders, blocky structures and stripe artifacts show up. The acceptable image quality can be achieved only when there are sufficient voxels and their size is at least equal to the pixel pitch (which ultimately is planned to be 50 µm) on the detector. Nevertheless, the enlargement of specific region in the image still can reveal artifacts due to voxel's property. The following tests are performed in the case that detector size is 512*512 pixels.

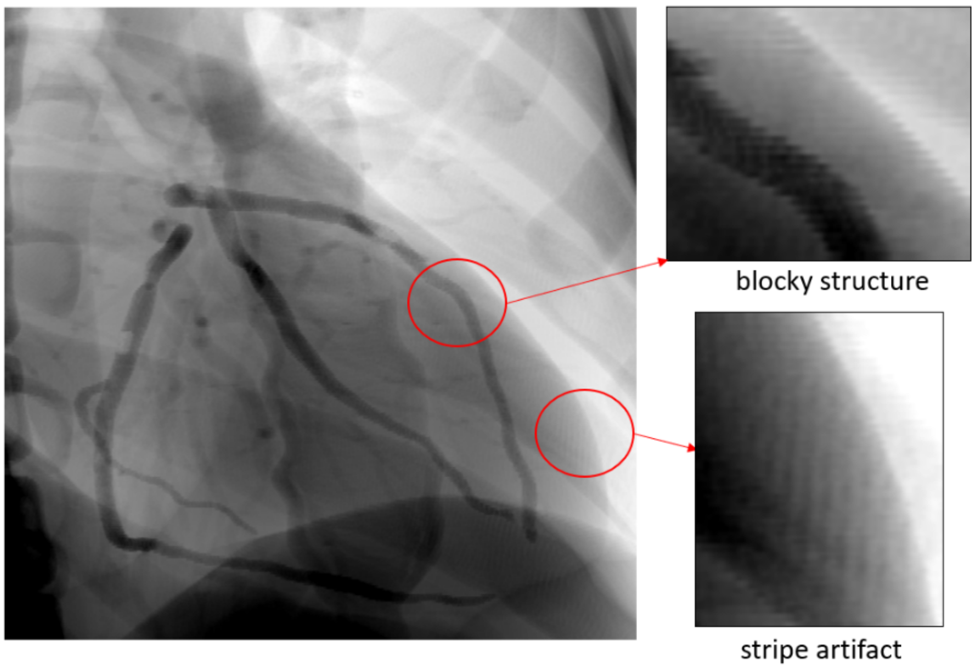


FIGURE 4.3. The X-ray image simulated with voxelized model of 0.5mm. The region of the heart is simulated from different phantom data. Blocky structure and stripe artifact is obvious in the image.

4.2.2 X-ray tracking triangle meshed model

The algorithm that simulates triangle meshed object is written by myself, the resulting image quality is promising while the performance should not be referred as the performance of the standard algorithm in state of the art or in the market.

Fig.4.4 shows three variants of the heart model represented by different number of triangles and Fig.4.5 shows their corresponding output depth images. The left image displays the depth image of the model composed of 288 triangles, showing obvious curvatures as indicated in the figure. Additionally, when viewing the image from a long distance, the unsmoothness of the surface can be noticed. After increasing the number of triangles to 780, the unsmoothness and curvatures are hardly visible unless enlarge some specific region. Compared to 288 triangles and 780 triangles, 2600 triangles present a satisfying image quality, with no unsmoothness and curvatures even after zoom-in. Also to be mentioned, in all three images, there are several white dots randomly appearing on the surface, which I believe is the artifact of my algorithm.

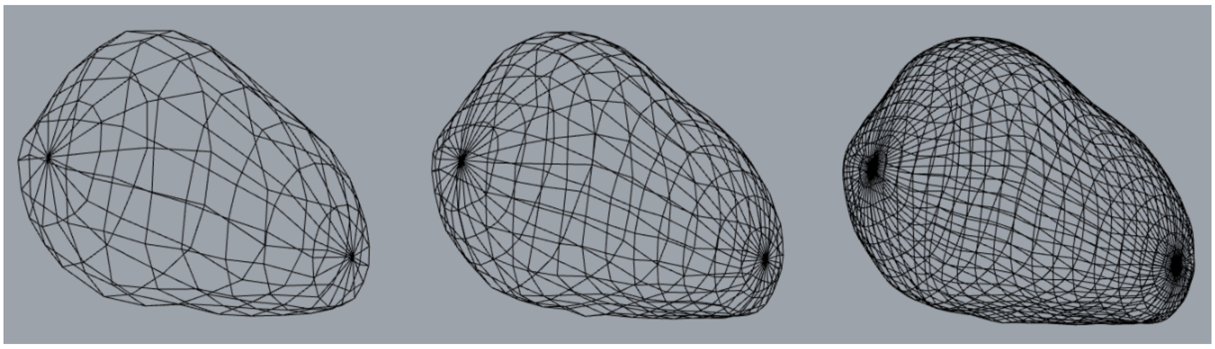


FIGURE 4.4. The heart model transformed from NURBS data, composed of different number of triangles: 288 triangles (left), 780 triangles (middle), 2600 triangles (right).

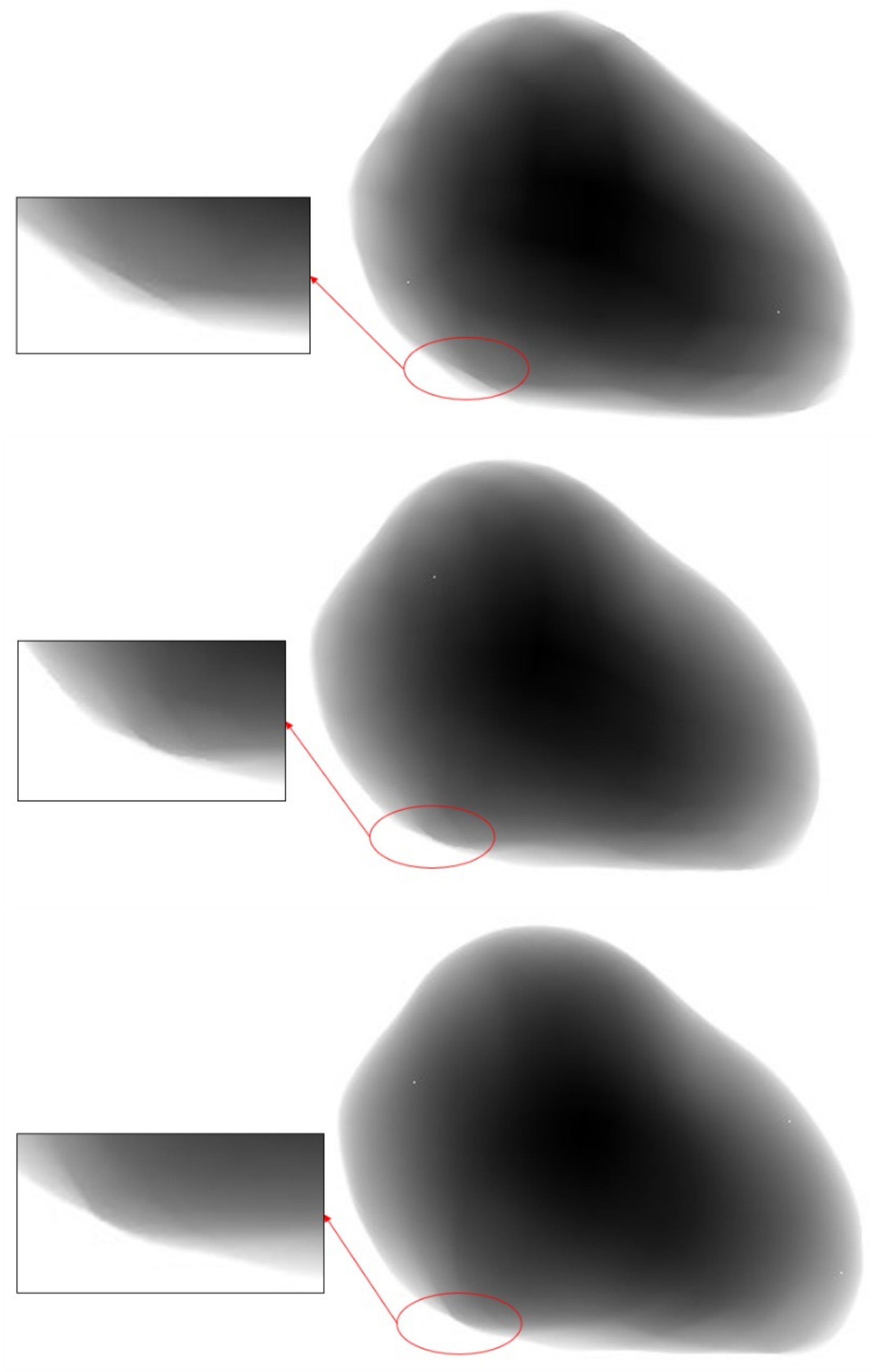


FIGURE 4.5. The depth image simulated with the heart model represented by different number of triangles: 288 triangles (top), 780 triangles (middle), 2600 triangles (bottom).

4.2.3 X-ray tracking NURBS surface with Newton's iteration

The algorithm of X-ray tracking NURBS surface with Newton's iteration cannot produce a depth image due to the problems discussed in Section 3.1.3. Therefore, the algorithm is adapted to generate an image that indicates whether there is intersection found for each pixel. Fig.4.6 shows the experiment result of tracking the heart model represented by one NURBS object with Newton's iteration to calculate the intersection. In the figure, the pixel rendered with red indicates at least one intersection between the corresponding ray and the NURBS object is found and calculated, while the pixel rendered with gray indicates no intersection is found for the corresponding ray. There are severe artifacts (the surface of the object is not completely rendered) due to the intersections that are not found in the process of Newton's iteration. The image quality of this approach is far from satisfactory.

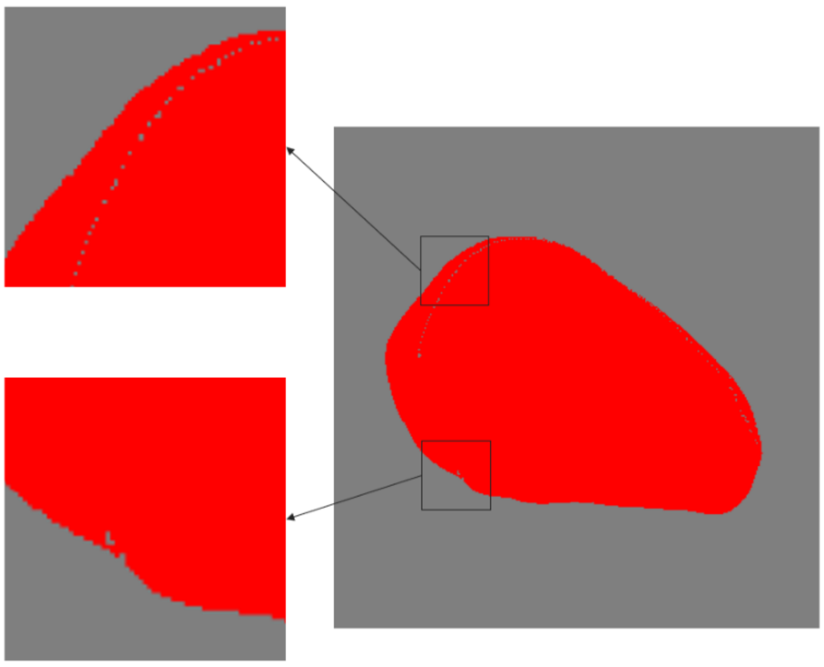


FIGURE 4.6. The resulting image of X-ray tracking NURBS surface with Newton's iteration. Pixels that are rendered gray indicates no intersection is found between rays and NURBS object, pixels that are rendered red indicates intersection is found. The holes in the clippings are non-found intersections.

4.2.4 Direct mathematical projection of NURBS surface

Fig.4.7 demonstrates the three result depth images of the heart model represented by one NURBS object with direct mathematical projection, each one of those is simulated with different sampling density in the sampling program. Sampling density refers to the number of surface points that are sampled and projected in the parametric region UV. The left image displays the depth image with sampling density of 20*20, image artifacts are barely visible when viewed from a long distance. However, some curvatures of the surface can be noticed after enlargement. When increasing the sampling density to 40*40 and 60*60, no image artifact is found even when enlargement and the image quality is satisfying. Additionally, the image quality remains as such if the sampling density is further increased. In the three depth images, a white dot appears at the left part of the heart, which is identified as the pole ($u = v = 1.0$) of the NURBS patch. The definition of [PT12] does not cover the pole, therefore resulting this non-existing pole in the image. This artifact can be fixed by defining a special case for $u = v = 1.0$ in the algorithm.

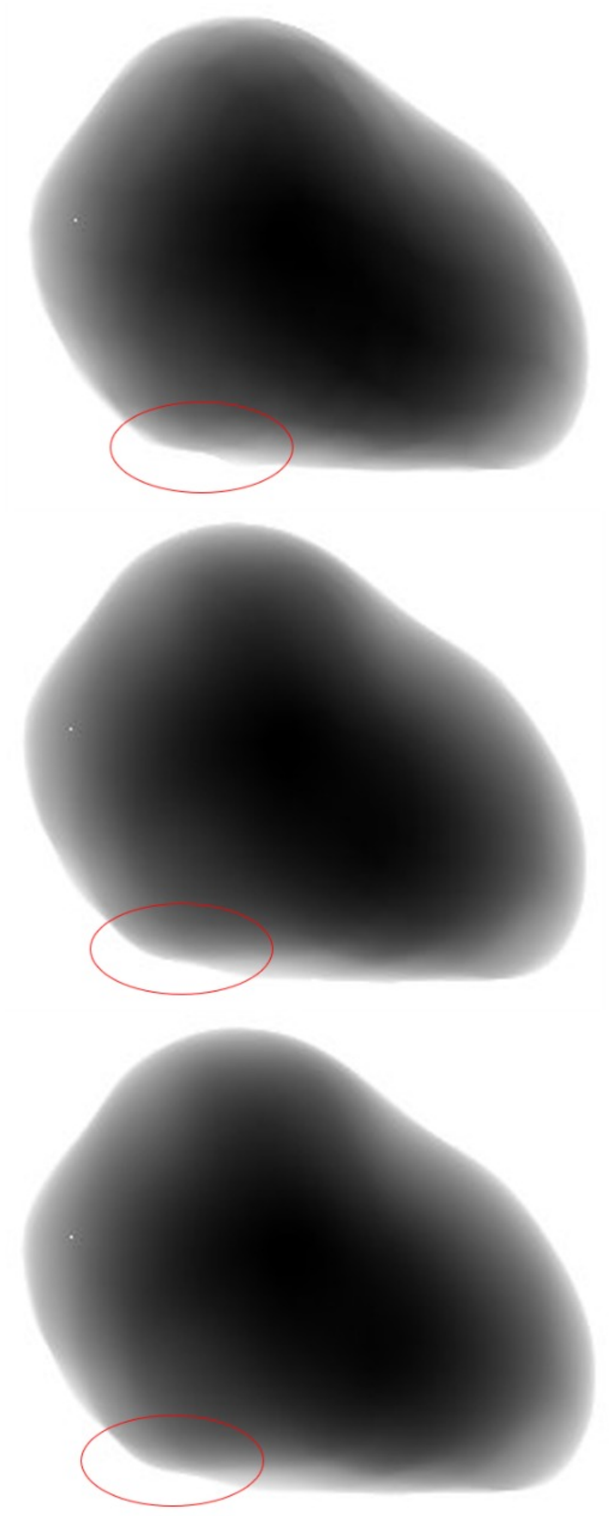


FIGURE 4.7. The depth image simulated with the heart model by setting different sampling density: 20*20 (top), 40*40 (middle), 60*60 (bottom).

4.2.5 Evaluation

According to the test results on image quality, X-ray tracking NURBS surface with Newton's iteration has the most significant image artifacts because of its difficulty to efficiently find intersections. Simulation with voxelized model produces blocky structures and stripe artifacts especially at the edge of objects. Simulation with triangle-meshed model and direct mathematical projection both give some unsmoothness on the surface when the number of triangles/sampling density is low. However, simulation with triangle-meshed model requires a great number of triangles to achieve the comparable image quality as direct mathematical projection with a medium sampling density. In general, direct mathematical projection presents the best image quality when the triangle-meshed model has an average number of triangles.

4.3 Performance

The performances of different approaches are compared based on the usage of CPU and GPU. To be noted, X-ray tracking NURBS surface with Newton's iteration has only GPU version since it used Nvidia OptiX framework at the beginning. And the algorithm that simulates triangle-meshed object is written by myself, the resulting image quality is reliable while the performance should not be referred as the performance of the standard algorithm in state of the art or in the market. The following tests are performed with the heart model in the case that the detector size of 512*512 pixels with 0.2mm pixel pitch. Only direct mathematical projection of NURBS surface would be tested with the human phantom.

4.3.1 X-ray tracking voxelized model

The simulation of voxelized model is already developed and optimized on CPU. The performance of simulating a full human voxelized phantom of 1mm is measured with different pixel numbers as shown in Fig.4.8. The execution to render the X-ray image on a 1024*1024 pixel detector takes around 0.65 second. However, with the linear scalability, the simulation time would reach 6.5 seconds when rendering a image on a 3000*3000 pixel detector by estimation. The calculation time for Fig.4.3 takes 2.6 seconds considering the shrinking size of voxels. As compared to the 0.21 seconds calculation time for 512^2 pixels in Fig.4.8, this is roughly a factor of 12 slower, however also the voxel size is a factor of 2 smaller, 0.5mm to be specific. And voxel size of factor 2 means voxels number of factor 8. What's more, the performance scales dependently on the viewing angle (due to memory access effects), and the phantom inner structure complexity.

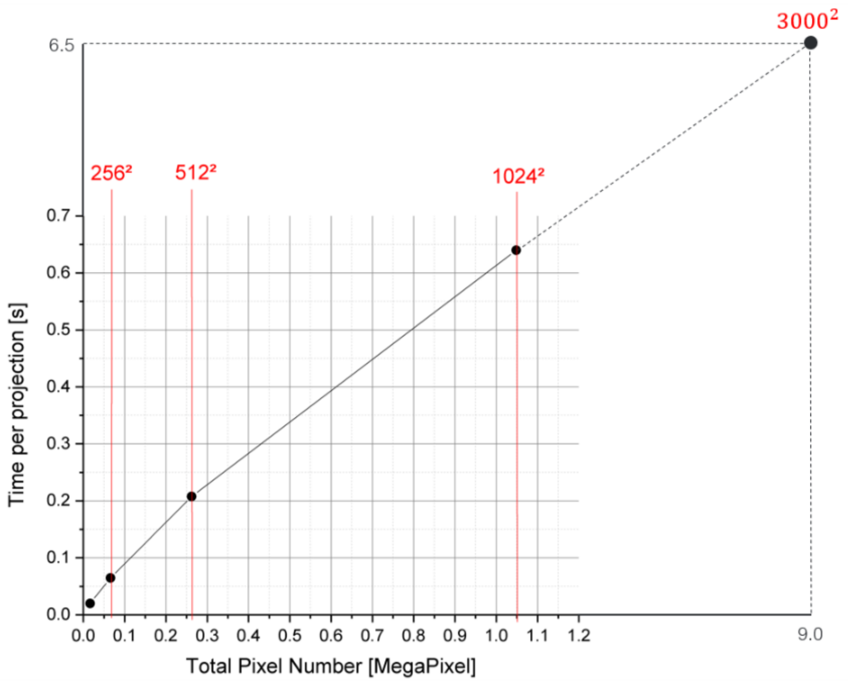


FIGURE 4.8. The performance measurement of simulation with voxelized phantom of 1mm, which scales withe the number of pixels on the detector.

4.3.2 X-ray tracking triangle meshed model

The algorithm to simulate triangle meshes model is adapted from a standard ray tracing algorithm by calculating all intersections on a ray instead of the nearest intersection. No further optimization technique is applied to improve the performance except porting onto GPU. The GPU simply makes each thread to track a ray corresponding to each pixel. Figure.4.9 demonstrates the performance of X-ray tracking triangle meshed heart model (dias_pericardium.obj) represented by different number of triangles. The detector size is set as 512*512 with 0.2mm pixel pitch. From the figure, we can tell the performance of X-ray tracking voxelized model has linear scalability with the number of triangles representing the object.

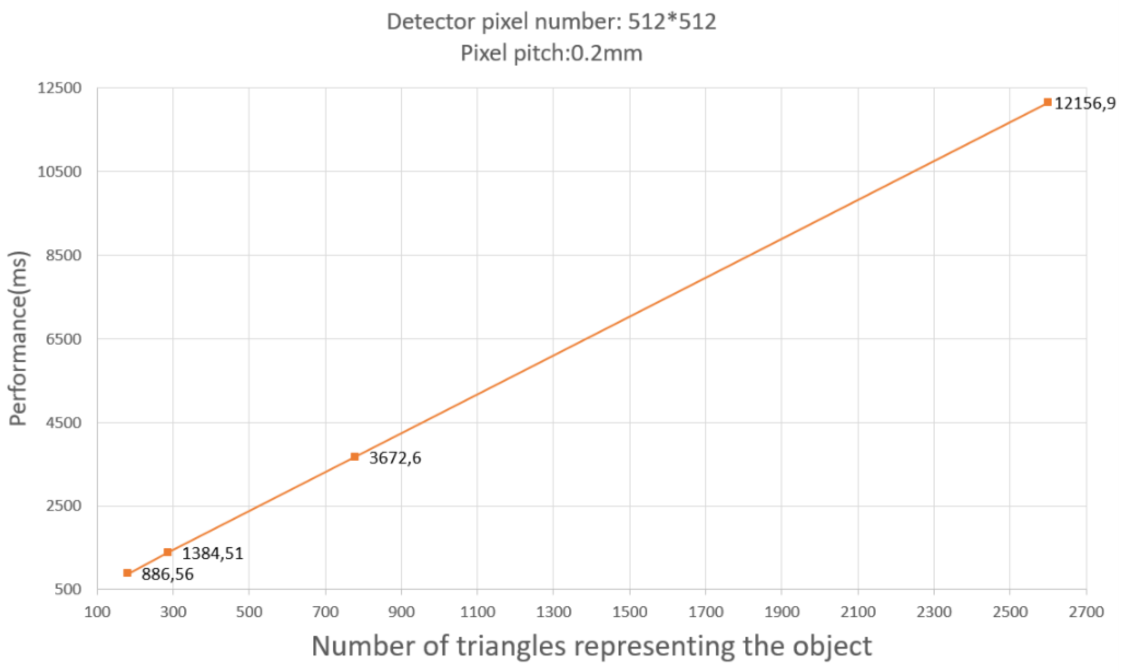


FIGURE 4.9. The performance simulated with the heart model represented by different trianlge numbers: 180, 288, 780, 2600 respectively.

4.3.3 X-ray tracking NURBS surface with Newton's iteration

Since Nvidia OptiX does not provide any library or tool for low-level profiling yet, the time measurement of specific function can only be realized by the following code (List.3):

```
clock_t start_time = clock ();
//function to be measured
clock_t stop_time = clock ();
int time = (int) (stop_time - start_time);
rtPrintf ("time in func: \%f\n", time / clockRate);
```

Listing 3: The code that shows the time measurement of function inside of an OptiX kernel. The clock rate refers to the GPU core clock in Table.4.1.

The performance of X-ray tracking NURBS surface with Newton's iteration simulating the NURBS heart model is **5217.97ms** on average. Furthermore, profiling over single GPU kernel would provide extra information on how the initial guess finding performs in the approach. Fig.4.10 shows the average time consumption of kernel with low and high workload. Both indicates that the initial guess finding occupies a huge portion of the entire kernel execution time, approximately 99%. The figure verifies that the initial guess finding is the biggest difficulty when implementing Newton's iteration approach. It also shows that the converge of Newton's iteration takes only 1.5ms approximately to finish or terminate. This method provides at least a minimum required processing performance, so a full generation of (multiple) intersections list should be a multiple of the measured calculation time.

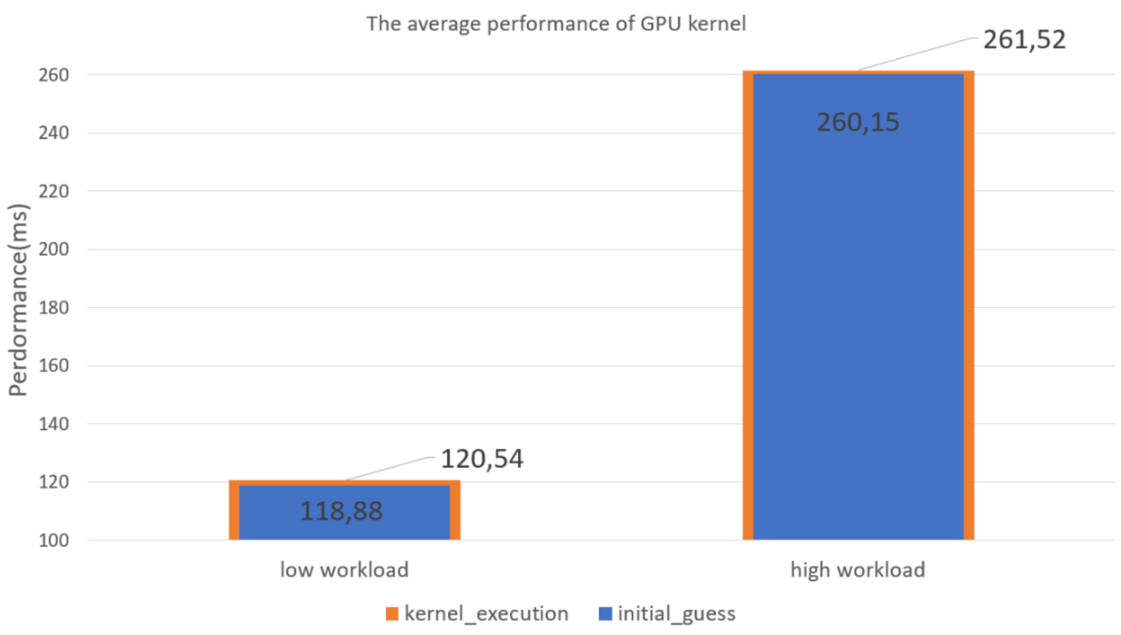


FIGURE 4.10. The profiling of the initial guess finding function in each GPU kernel.

4.3.4 Direct mathematical projection of NURBS surface CPU version

Direct mathematical projection of NURBS surface as the newly proposed method in this thesis are tested from different aspects to show its advantages and disadvantages. Profiling is done as well in order to show the bottleneck of this approach. The testing focuses on the CPU version.

Performance

The performance of direct mathematical projection of NURBS surface is tested with both test objects, dias_pericardium.obj (heart) and male_pt146.obj (whole human body) respectively. The test setting with human body model is different because it is huge in the simulation 3D space compared to the heart model. Normally, the length of a detector in the real life would not exceed 40cm. Nevertheless, to make sure most components of the human body model are contained in the generated depth image, both the detector size and pixel patch are increased for testing. Table.4.3 shows the performance regarding total execution time for the whole object and calculation time for each NURBS patch. Fig.4.11 demonstrates the result depth image of male_pt146.obj from the test.

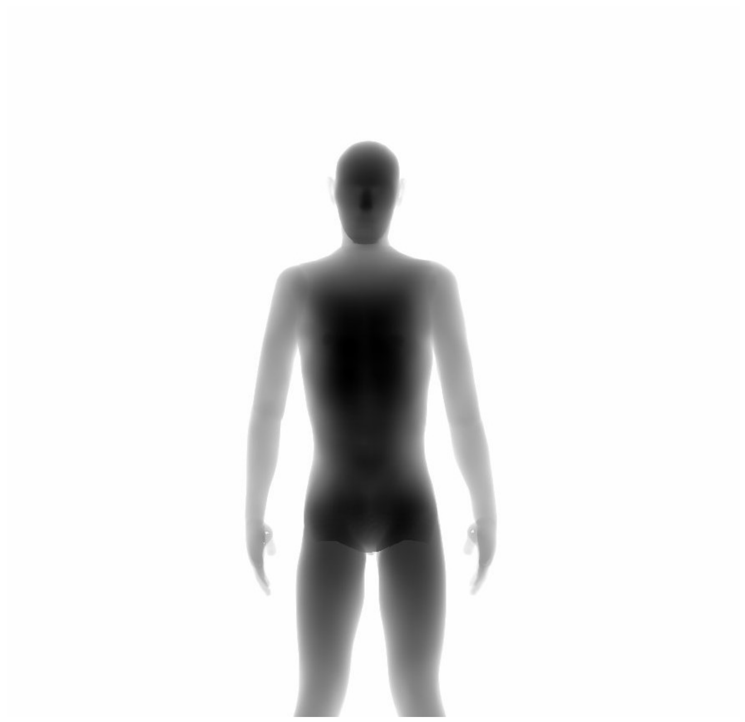


FIGURE 4.11. The depth image rendered by direct mathematical projection of NURBS surface approach with male_pt146.obj. The test setting refers to Table.4.3.

sampling	projection	rendering	sorting
22.5%	2.5%	66%	9%

TABLE 4.4. The profiling of time consumption of programs in direct mathematical projection of NURBS surface approach.

	dias_pericardium (heart)	male_pt146 (whole human body)
Detector size	512*512	1024*1024
Pixel pitch	0.2mm	2mm
Sampling density	60*60	60*60
Number of NURBS patch	1	588
Average performance	104.23ms	78039.40ms
Average performance per NURBS patch	104.23ms	132.70ms

TABLE 4.3. The test setting and result of direct mathematical projection of NURBS surface.

Profiling

This section briefly discusses on the distribution of the time consumption of each function, and the bottleneck of the program based on the profiling result. Table.4.4 displays the average percentage of time spent on different stages of the program.

As shown Table.4.4, more than half of the time is consumed in the rendering program. Sampling the surface points costs more than 20% of the time (depending on sampling density). Such that, the rendering program has the top priority to be ported and optimized on GPU. However, the algorithm still samples the surface points of all NURBS objects even when they are not in the scope of the detector plane after projection. In that case, the rendering program would be terminated and the sampling program could spend a huge portion of time on calculating equations regarding NURBS. Therefore, it is essential to port the NURBS calculation part onto GPU as well.

Sampling density

In this part, the influence of different sampling density on the performance of simulating single NURBS patch is investigated regardless of image quality. The detector size and its pixel pitch are fixed as 512*512 and 0.2mm respectively. By varying the sampling density from 20*20 to 120*120, the change of the performance can be observed as shown in Fig.4.12. Despite the fact that the graph presents a quadratic growth of performance when increasing sampling density, the image quality remains unimproved once reach 60*60 in the case of this test.

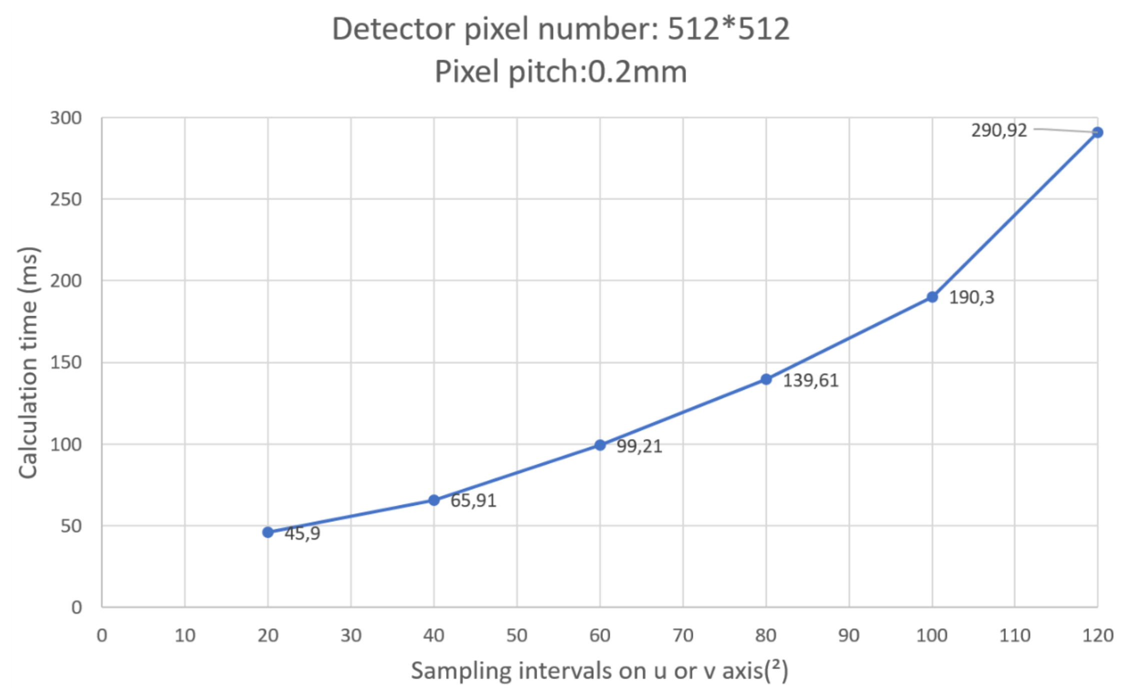


FIGURE 4.12. The performance of direct mathematical projection of NURBS surface when varying the sampling density in the algorithm.

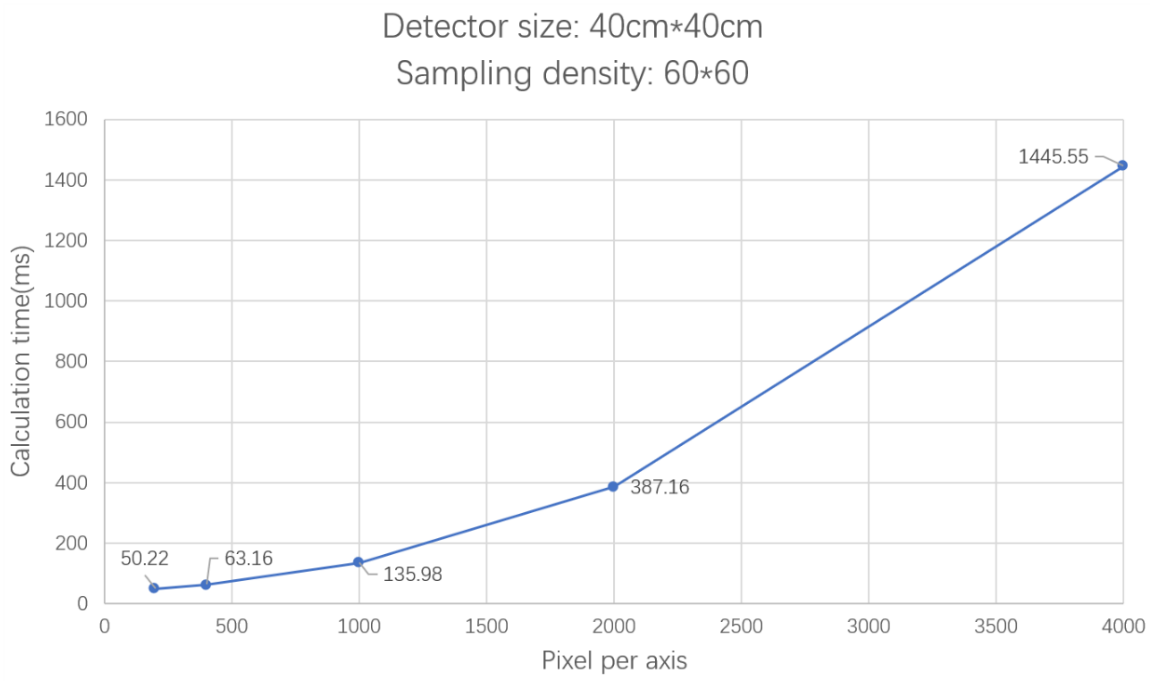


FIGURE 4.13. The performance of direct mathematical projection of NURBS surface when varying the pixel pitch on the detector.

Detector pixel pitch

In the real life, the size of a detector plane is normally not bigger than 40cm*40cm. Therefore, in this test, the influence of varying pixel pitch on a detector of fixed size (40cm*40cm) on the performance is investigated. The sampling density is set as 60*60 in this test. The graph shows a quadratic growth of performance when decreasing the pixel pitch.

4.3.5 Direct mathematical projection of NURBS surface GPU version

Only the rendering program of direct mathematical projection of NURBS surface is implemented due to limited time. GPU porting will fully take advantage of the parallelism inside the algorithm, and further accelerate the program to meet the requirement of real-time (30fps for 9megapixel detector). Two points should be the highest priority when hand-optimizing a GPU kernel: the occupancy should be maximized by exploiting GPU parallelism and the memory latency should be avoided by reducing local memory usage. The algorithm is adapted such that each block is responsible of constructing a triangle and set up the region to be searched (currently a 10*10 pixel square region), and each thread is responsible of searching one pixel in the region to check whether it is located in the triangle or not. Additionally, shared memory is used to store variables that are frequently used by threads in each block. Table.4.5 shows an performance gain of ×18.7 for the heart model and ×8.67 for the full human phantom respectively compared to the CPU version.

	dias_pericardium (heart)	male_pt146 (whole human body)
Detector size	512*512	1024*1024
Pixel pitch	0.2mm	2mm
Sampling density	60*60	60*60
Number of NURBS patch	1	588
Average performance	5.57ms	9002.28ms
Average performance per NURBS patch	5.57ms	15.31ms
Speedup compared to CPU version	×18.7	×8.67

TABLE 4.5. The performance after rendering program is ported onto GPU with the same configuration as Table.4.3.

GPU fine tuning

The CUDA programming model is introduced in Section 2.5.1 and the aim of tuning is to maximize the occupancy of each streaming multiprocessor by configuring the grid and block size. Nvidia Visual Profiler is applied here to profile the usage of GPU. Table.4.6 shows profiling result with different grid and block sizes with the rendering program. The result shows that the GPU kernel favors the grid size close to its sampling density and block size close to its search area. Thus, the configuration of the grid and block size would be dynamically done by assigning themselves the multiple of 32 that is close to the sampling density and search area. 32 is the warp size of the GPU and the performance would be improved when the block size it the multiple of it.

grid size	block size	occupancy (sampling density = 60)	occupancy (sampling density = 100)
128*128	16*16	45.0%	43.4%
128*128	8*8	48.4%	51.0%
64*64	8*8	48.7%	50.6%
64*64	4*4	42.0%	39.9%

TABLE 4.6. Different configuration of grid and block size. Occupancy donates the occupancy per streaming multiprocessor, shared memory donates the shared memory usage of variables in the same block.

Table.4.7 demonstrates the percentages of calculation time spent on sampling program, projection program and rendering program. The next step will be porting the sorting program onto GPU as well and determining the bottleneck.

sampling	projection	rendering
0.05%	≤ 0.01%	≥ 99%

TABLE 4.7. The profiling of time consumption of programs in direct mathematical projection of NURBS surface approach on GPU. The sorting program is not measured.

Figure.4.14 shows the performance with varying pixel pitch size with GPU usage. Only the sampling, projection and rendering program are measured here, because the porting of the sorting algorithm is not done yet and the sorting on CPU occupies approximately 85% of the entire execution time.

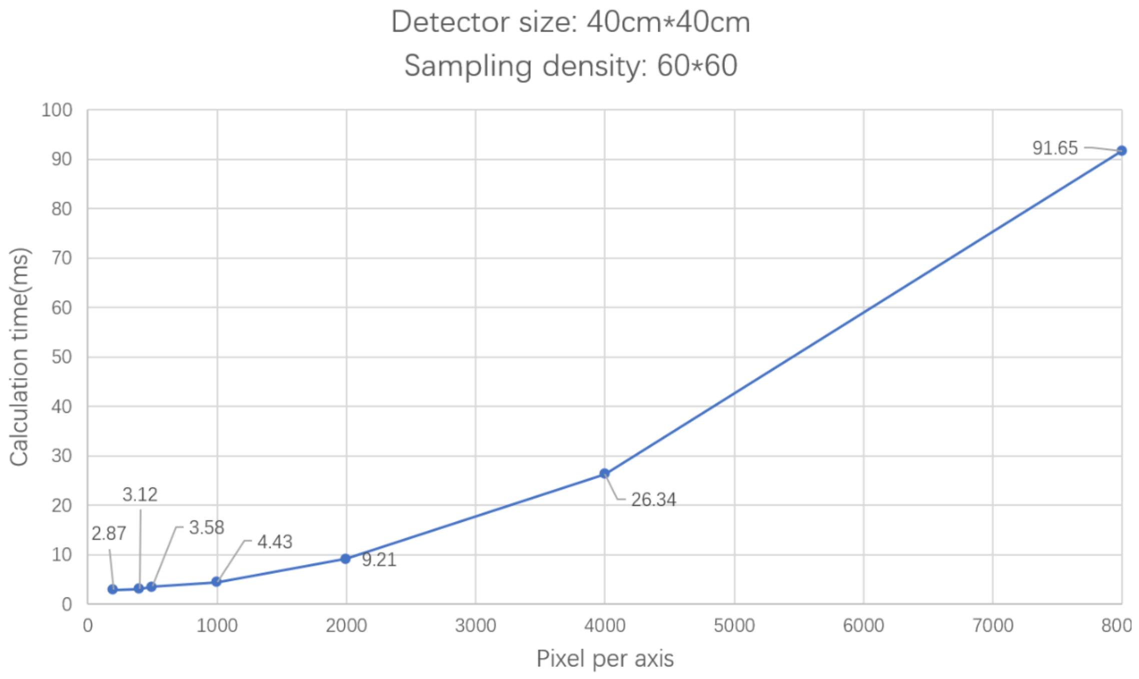


FIGURE 4.14. The performance of direct mathematical projection of NURBS surface with GPU usage when varying the pixel pitch on the detector. Tested with the heart model.

4.3.6 Evaluation

X-ray tracking NURBS surface with Newton's iteration gives the worst performance even executed on GPU, with almost all of the execution time spent on initial guess finding. For the other approaches on CPU version, simulation with voxelized model has the best performance which can simulate a whole human phantom in 0.65s on a 1024*1024 detector. There is no remarks regarding performance of triangle-meshed model since the algorithm is not state-of-the-art yet. Direct mathematical projection on GPU achieves a ×8.67 speedup compared to its CPU version, however still does not outperform simulation with voxelized model on a 1024*1024 detector. The bottleneck of direct mathematical projection indeed lies in the rendering program and sorting algorithm as discussed in Section 3.2.3. Contrast to the approach of tracking voxelized model, the code still can be well optimized (which was not possible given the limited time for the thesis work) to gain speedup, this is valid for the rendering as well as for the sorting code.

4.4 Discussion

4.4.1 Image quality

From the perspective of image quality of resulting depth image, among the four approaches that has been tested, X-ray tracking NURBS surface with Newton's iteration can hardly produce a satisfying result due to the fundamental mathematical difficulty in finding suitable initial guess for Newton's iteration. The drawback makes it complicated to generate a depth image since it adds even more complexity when calculating multiple intersections along a single ray. X-ray tracking voxelized model is able to produce satisfying image quality when the voxels are even much smaller than a pixel on the detector, however resulting in an incredibly large data size (over 10TB) and therefore low performance. Normally, the human phantom is represented by 0.5mm*0.5mm*0.5mm voxels and the result X-ray image has visible artifacts due to voxel's property. The conventional approach, namely X-ray tracking triangle meshed model, is able to produce a decent image quality at the price of constructing the object with sufficient number of triangles at the pre-processing stage. Additionally, the performance of this approach scales with the increase of the number of triangles. Direct mathematical projection of NURBS surface can generate a depth image without artifacts when a enough sampling density is reached (depending on the pixel pitch, detector size and its distance to X-ray source). Normally, the sampling density is dependent on object size, it would make sense to adapt the sampling density to intervals matching the pixel pitch. During my tests, under the condition of comparable image quality, direct mathematical projection has a better performance than triangle meshed approach. Since the code of triangle meshed approach is not state-of-the-art or from the market, it remains questionable which approach would outperform the other.

4.4.2 Performance

From the perspective of performance, X-ray tracking NURBS surface with Newton's iteration still performs poor again because its initial guess finding is the biggest bottleneck, costing over 95% of the time consumption. On the other hand, the GPU version direction mathematical projection of NURBS surface has achieved at most $\times18.7$ speed up compared to the CPU version. Meanwhile, the GPU version is able to simulating the heart model at 30fps on a 9megapixels detector (40cm*40cm) as well. Nevertheless, its performance scales quadratically both with sampling density and number of pixels on the detector (also known as increasing detector resolution). Currently, only the rendering program is ported onto GPU. Fig.4.15 reveals the performance comparison between simulation of voxelized model and GPU version direct mathematical projection. Only sampling program, projection program and rendering program are measured in order to avoid the time spent on sorting algorithm which is not ported onto GPU yet. As can be seen in the figure, the human phantom is simulated at least 3 times faster with the new approach on a 9megapixel detector. Additionally, the simulation with voxelized model scales much faster with the number of pixels as well. The reason why direction mathematical projection approach scales slowly with the pixel number and have an offset around 1 second might be there are parts in the algorithm that compute regardless of pixel numbers and the calculation time of the actual rendering procedure already has a good scalability with pixel number.

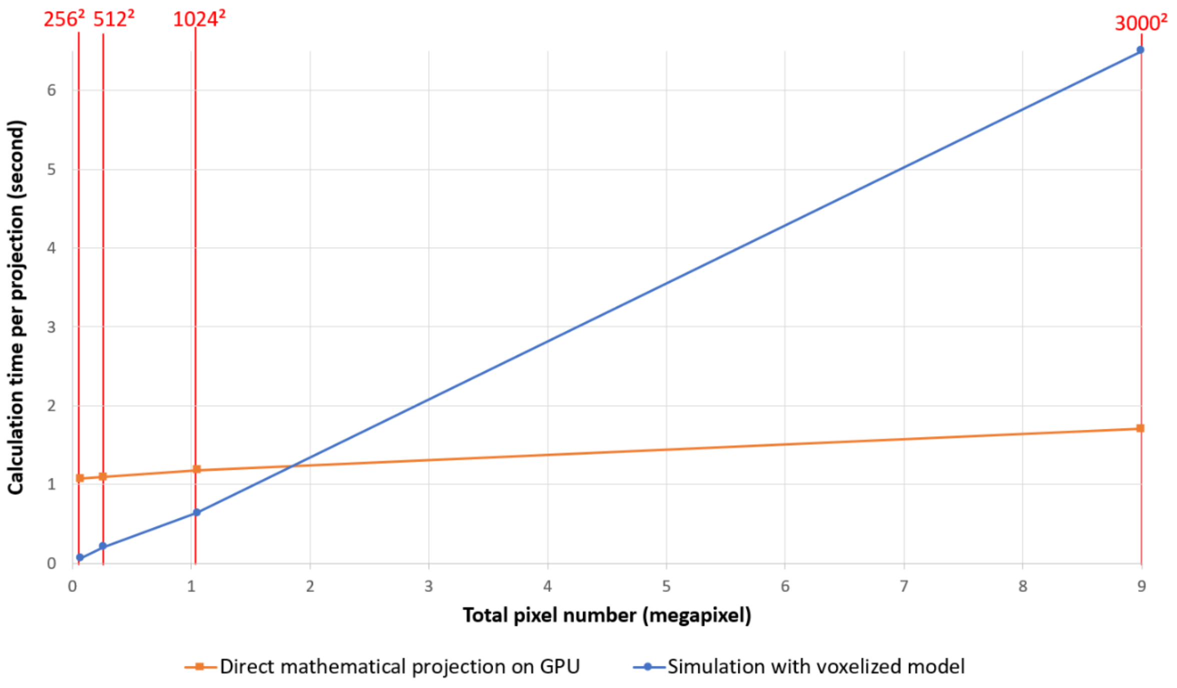


FIGURE 4.15. The performance comparison of simulating a human phantom between simulation of voxelized model and GPU version direct mathematical projection (only sampling program, projection program and rendering program). The simulation of voxelized model scales much faster than direct mathematical projection with the increasing size of the detector. The latter one has approximately $\times 3$ speed up compared to the former one on a 9megapixel detector.

CHAPTER 5

FUTURE WORK

Further algorithm and GPU optimization
The rendering program in direct mathematical projection of NURBS surface is not fully optimized yet. There are still some repeated calculation and unnecessary search in the algorithm to be optimized. In addition, more parallelism in the sampling program and projection program can be fully adapted onto GPU. The sorting algorithm which is essential for generation of the depth image can also be ported onto GPU. Despite the fact that plenty of C++ Standard Template Library (STL) code is used, Thrust as a C++ template library for CUDA can provide some help.

Dynamic adjustment of sampling density
Actually, there is no necessity to manually set the sampling density since the size of the detector is fixed (normally 40cm*40cm) in the real world. Once we know the position of the X-ray source and the NURBS patch to be projected, the size of the NURBS patch projected on the detector can be easily calculated considering the magnification factor. The sampling density can be decreased when the patch size is small and vice versa. Since lower sampling density leads to higher performance, the overall performance will increase when small NURBS patches are contained.

Acceleration data structure
Currently, the proposed methods have not applied any acceleration data structure yet. Components of human phantom such as bones, muscles and vessels are represented by single NURBS patches. Acceleration data structures (ADSs) like grid, KD-tree and bounding volume hierarchy (as [Vin13] introduced) can effectively skip the NURBS patches that are not in the scope of the detector, therefore save a large portion of computation effort. Another approach regarding ADS is portioning the NURBS patches into smaller pieces and bounding them with volumes[Val10], the center of the volumes can be taken as reliable initial guess for the Newton's iteration. Therefore, the problem of X-ray tracking NURBS surface with Newton's iteration can be solved.

Packet-based ray tracing

In [WSBW01], a performance increase by a factor of ten had been achieved by tracing packets of rays in parallel instead of tracing each ray sequentially. By taking advantage of the coherence of rays (the same memory address is accessed multiple times for rays intersecting the same primitive), multiple rays are grouped to form a packet of rays and the memory is accessed only once for the whole packet. The main bottle-neck of this approach is the memory bandwidth[WSBW01]. This issue would be solved using the advanced GPU from this thesis.

Chapter 6

Conclusion

The current X-ray image simulation realized by tracking voxelized model presents an unsatisfactory image quality where blocky structures and stripes artifacts appear, with limited real-time performance. The data size of voxelized model will increase cubically with higher resolution as well. Triangle-meshed model might be able to improve the computation performance, however with limited usage of modification for human phantom in the future.

The wishes for the X-ray image simulation are: First, enabling the real-time simulation speed. Second, a highly realistic image quality at $50\mu m$ resolution.

In this thesis, an existing algorithm that directly calculates intersections with NURBS using numerical approach for X-ray physics has been applied. Then, a novel method called direct mathematical projection of NURBS surface is proposed, which promises higher computation efficiency and better image quality. Besides, an algorithm that tracks triangle-meshed models for X-ray physics is adapted from ray tracing algorithm. This algorithm is mainly used for image quality testing. Finally, comparison among voxelized model, triangles-meshed model and the novel method (direct mathematical projection of NURBS surface) on image quality and performance has been conducted.

Simulation with voxelized model
This approach still has the best performance on a average size detector, nevertheless produces image artifacts. In addition, the data size of voxelized model scales cubically with image resolution.

Simulation with triangle-meshed model
This approach requires sufficient number of triangles to give decent image quality. The performance scales linearly with the number of triangles in the scene.

X-ray tracking NURBS surface with Newton's iteration
This approach produces the worst performance as well as image quality due to its mathematical

complexity. The bottleneck of this approach is the initial guess finding.

Direct mathematical projection of NURBS surface

The GPU version of this approach achieves the best image quality and best scalability with the number of pixels. It has the flexibility to adjust the image quality. However, The computation algorithm is still the bottleneck that drags down the performance.

In conclusion, direct mathematical projection of NURBS surface with the usage of GPU is the best approach in this thesis. This method not only has a promising computation efficiency, but also scales slower with the number of pixels on performance. What's more, it is flexible to provide realistic image quality.

In the future development, the method of direct mathematical projection of NURBS surface can be improved by further algorithm and GPU optimization. Acceleration data structure is strongly recommended as well such that NURBS data can be accessed more efficiently and unnecessary computation can be avoided.

Bibliography

[Abe05] Oliver P Abert.
Interactive ray tracing of nurbs surfaces by using simd instructions and the gpu in parallel.
Diss. Nanyang Technological University, 2005.

[AGM06] Oliver Abert, Markus Geimer, and Stefan Muller.
Direct and fast ray tracing of nurbs surfaces.
In *Interactive Ray Tracing 2006, IEEE Symposium on*, pages 161–168. IEEE, 2006.

[CAPD14] Raquel Concheiro, Margarita Amor, Emilio J Padrón, and Michael Doggett.
Interactive rendering of nurbs surfaces.
Computer-Aided Design, 56:34–44, 2014.

[Efr05] Alexander Efremov.
Efficient ray tracing of trimmed nurbs surfaces.
Master's thesis, Computer Graphics Group, Max-Planck-Institut für Informatik, Saarbrücken, Germany.–2005.–162 p, 2005.

[EHS05] Alexander Efremov, Vlastimil Havran, and Hans-Peter Seidel.
Robust and numerically stable bézier clipping method for ray tracing nurbs surfaces.
In *Proceedings of the 21st spring conference on Computer graphics*, pages 127–135. ACM, 2005.

[MCFS00] William Martin, Elaine Cohen, Russell Fish, and Peter Shirley.
Practical ray tracing of trimmed nurbs surfaces.
Journal of Graphics Tools, 5(1):27–52, 2000.

[NSK90] Tomoyuki Nishita, Thomas W Sederberg, and Masanori Kakimoto.
Ray tracing trimmed rational surface patches.
ACM SIGGRAPH Computer Graphics, 24(4):337–345, 1990.

[Nvi] Nvidia.
Nvidia titan v gpu.
`https://www.nvidia.com/en-us/titan/titan-v/` (Oct. 2018).

[Opt] OptiX.
Nvidia optix 5.1 — programming guide.
`http://raytracing-docs.nvidia.com/optix/guide/index.html#guide#`.
Accessed: 2018-09-10.

[PT12] Les Piegl and Wayne Tiller.
The NURBS book.
Springer Science & Business Media, 2012.

[PTVF96] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery.
Numerical recipes in C, volume 2.
Cambridge university press Cambridge, 1996.

[Rhi] Rhino.
Rhino, the 3d modeller software.
`https://www.rhino3d.com//` (Oct. 2018).

[Rup] Karl Rupp.
Cpu, gpu and mic hardware characteristics over time.
`https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-t`
(Oct. 2018).

[SW] Mendonca S Grimes J Tsui BM Segars WP, Sturgeon G.
4d xcat phantom for multimodality imaging research.
`https://www.ncbi.nlm.nih.gov/pubmed/20964209` (Oct. 2018).

[Tat07] Natalya Tatarchuk.
Real-time tessellation on gpu.
SIGGRAPH Advanced Real-Time Rendering in 3D Graphics and Games Course, 37, 2007.

[Val10] EEJ Valkering.
Ray tracing nurbs surfaces using cuda.
Master's thesis, Delft University of Technology, 2010.

[Vin13] Marek Vinkler.
Acceleration Data Structure Construction for Ray tracing.
PhD thesis, Masarykova univerzita, Fakulta informatiky, 2013.

[WMG$^+$09] Ingo Wald, William R Mark, Johannes Günther, Solomon Boulos, Thiago Ize, Warren Hunt, Steven G Parker, and Peter Shirley.
State of the art in ray tracing animated scenes.

In *Computer graphics forum*, volume 28, pages 1691–1722. Wiley Online Library, 2009.

[WSBW01] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner.
Interactive rendering with coherent ray tracing.
In *Computer graphics forum*, volume 20, pages 153–165. Wiley Online Library, 2001.

[PT12] [Tat07] [WMG$^+$09] [Efr05] [Abe05] [Val10] [Vin13] [CAPD14] [AGM06] [MCFS00] [EHS05]