UiO : Department of Informatics
University of Oslo

Voxel-Based Level-of-Detail Visualization of Isogeometric Volumes

Sveinung Fossås

Master's Thesis Spring 2016

Voxel-Based Level-of-Detail Visualization of Isogeometric Volumes

Sveinung Fossås

May 27, 2016

Abstract

3D models created in Computer Aided Design (CAD) software are ubiquitous in the manufacturing industry. Preparing for analysis on them has typically been a very time-consuming process, and isogeometric analysis was therefore proposed as an alternative approach. It is desirable to be able to visualize the models during isogeometric analysis, but this is not a straightforward process due to how the CAD models are defined. A recently developed approach allows interactive visualization of these models as volumes by ray-casting. This approach ensures highly accurate renderings, but at the cost of being computationally expensive.

This thesis explores ways of achieving improved performance while rendering these models, by looking at combining the isogeometric models with simplified voxelized representations. A 2D prototype was developed to experiment and test different ray-casting methods, before moving to a 3D implementation. To determine the visual accuracy of the different methods the CIEDE2000 algorithm for color difference was used to compare the results from rendering to a reference solution.

Several methods have been proposed in this thesis that significantly improves the rendering performance with the results also having good visual accuracy. This includes hybrid methods that dynamically switch between isogeometric models and their voxelized representations while ensuring that samples are pixel accurate. The methods have been compared against each other in different examples, and the generated results show that some of the proposed methods are very good candidates for visualizing isogeometric volumes with increased performance.

Acknowledgements

I wish to sincerely thank my supervisors at SINTEF, Jon M. Hjelmervik and Franz G. Fuchs for their help on this project, all the valuable advice, and for pushing me in the right direction throughout the project. I would also like to thank my supervisor at the University of Oslo, Eigil Samset for his helpful feedback.

I would also like to thank my friends for being there through the toughest times during the project. My thanks also go out to my brother Ole Johnny and my mother Astrid for their patience and support throughout these years. A special thanks goes to Hana for all her love and support.

Contents

1 Introduction **1**
- 1.1 Overview . . . 1
- 1.2 Motivation . . . 1
- 1.3 Research goal . . . 3
- 1.4 Outline . . . 3

2 Background **5**
- 2.1 3D models . . . 5
- 2.2 Isogeometric analysis . . . 8
- 2.3 Scientific visualization . . . 10
- 2.4 Visualizing isogeometric models . . . 13
- 2.5 Performance and memory . . . 16

3 Methods for ray-casting **21**
- 3.1 Scope . . . 21
- 3.2 Validating methods . . . 21
- 3.3 Direct method . . . 23
- 3.4 Reduced models . . . 24
- 3.5 Voxelized method . . . 26
- 3.6 Boundary accurate method . . . 29
- 3.7 Hybrid method based on a geometric criterion . . . 31

4 Comparison of methods in 2D **35**
- 4.1 2D prototype . . . 35
- 4.2 Implementation of ray-casting methods . . . 38
- 4.3 Comparing the methods . . . 44
- 4.4 Case A: Trivial geometry, trivial scalar field . . . 46
- 4.5 Case B: Trivial geometry, non-trivial scalar field . . . 49
- 4.6 Case C: Non-trivial geometry, trivial scalar field . . . 52
- 4.7 Case D: Non-trivial geometry, non-trivial scalar field . . . 54

5 Comparison of methods in 3D **57**
5.1 3D framework 57
5.2 Implementation of ray-casting methods 58
5.3 Comparing the methods 64
5.4 Case A: Hybrid switchover 66
5.5 Case B: Far 73
5.6 Case C: Visual artifacts 75
5.7 Case D: Multiple volume blocks 78

6 Discussion **81**
6.1 General discussion 81
6.2 Conclusion 82
6.3 Future work 82

List of Figures

1.1 The different stages for isogeometric models. Visualization and diagram by Fuchs and Hjelmervik [8], model from the TERRIFIC project. 2

2.1 A voxel grid of size 8^3. Each voxel contains information about its own region in the volume. 6
2.2 A spline curve of degree 3 shown in green, and the corresponding control polygon in blue. The blue dots mark the control points. 8
2.3 Rendering isogeometric models. The geometry is denoted G, while P is the parameter domain. Diagram by Fuchs and Hjelmervik [8]. 9
2.4 In the simplified model, only absorption and emission affect the radiation along a view-ray. (Wenke and Vornberger, 2010) 11
2.5 Visualization of a middle ear with pre- and post-classification. Figure from Rezk-Salama and Hastreiter [27]. 13
2.6 Discrete LOD example for a boundary representation model. A lower LOD is used as the model of the car the further away it gets from the camera. Figure from Project CARS [26]. 17
2.7 A small example of a 2D sparse texture. Figures from Barrett [25]. 19

3.1 Visualization of the volume block from dataset 1. 24
3.2 Sampling the model with orthogonal projection as part of the voxelization process. Here an 8x8 texture is created. The black lines illustrate the sampling rays and each sample point is marked with a green circle. . . . 27
3.3 Comparison of the smooth boundary of the spline model (blue line), and the serrated boundary of the generated voxel grid. 30
3.4 Illustration of a pixel frustum in 2D. The view-ray, marked as a blue dotted line, starts at the eye, goes through the middle of a pixel, and continues into the scene. The sample point g is somewhere inside the isogeometric model, and ϵ denotes the projected size of the current pixel at this point. The view-ray separates the frustum into an upper and a lower part, which can be of different sizes. To achieve pixel accuracy, the distance between two diagonally adjacent voxels must be lower or equal to the minimum radius, which in this illustration is r_1. 32

4.1 Visualization of the transfer function from dataset 1. 37

4.2 Ray-casting with the direct method. The reference solution shown in (a) compared to the results from ray-casting with the pixel accurate direct method shown in (b). The CIEDE2000 color difference is visualized in (c). 39
4.3 A generated texture of size 32x32. Each texel is shown as a square with a greyscale color depending on the scalar value. Notice that the texture coordinates ranges from 0 to 1 inside the bounding box, shown with a blue dashed line. 40
4.4 Ray-casting with the voxelized method. The reference solution shown in (a) compared to the results from ray-casting with the voxelized method. A texture of size 192x192 has been used to generate the result in (b), and (c) shows the CIEDE2000 color difference. Figure (d) and (e) similarly shows the results from using a texture of size 256x256. 41
4.5 Ray-casting with the boundary accurate method. The reference solution shown in (a) compared to the results from ray-casting with the boundary accurate method. A texture of size 192x192 has been used to generate the result in (b), and (c) shows the CIEDE2000 color difference. 42
4.6 Comparison of mean color difference when clamping to edge versus clamping to border. A texture of size 32^2 has been used for all sampling distances. The results have been generated from ray-casting an isogeometric model with trivial geometry, meaning that $\phi(u, v) = (u, v)$. The scalar field is the same as was shown in figure 3.1b, and the transfer function is the same as was shown in figure 4.1. 43
4.7 Ray-casting with a hybrid of the direct method and the boundary accurate method using a 192x192 texture. Figure (a) shows the reference solution, the hybrid solution in (b), and the CIEDE2000 color difference in (c). The ratio between sample points from the spline model and sample points from the voxel grid is shown in (d), where the lighter the color is, the more samples have been taken from the voxel grid. 44
4.8 Illustration of the visual accuracy for Case A: Trivial geometry, trivial scalar field. The dataset is ray-casted with the different methods using different sample distances. These distances are dependent on the size of the textures. Along the horizontal axes, the numbers show one dimension of the quadratic texture sizes used. The resulting pixel colors are then compared to a reference solution with the CIEDE2000 algorithm to find the color differences. Subsequently the maximum and the mean of these are found to generate the data points shown in the graphs here. The vertical axes show the ΔE color difference value. 47
4.9 Illustration of the visual accuracy for Case B: Trivial geometry, non-trivial scalar field, similarly to figure 4.8. 49
4.10 Illustration of the visual accuracy for Case C: Non-trivial geometry, trivial scalar field, similarly to figure 4.8. 52
4.11 Illustration of the visual accuracy for Case D: Non-trivial geometry, non-trivial scalar field, similarly to figure 4.8. 54

5.1 The boundary accurate method can introduce visual artifacts for specific view-angles. 60
5.2 A 2D illustration of the problem with the boundary accurate method. The blue line represent the boundary of the spline model, while the greyscale rectangles represent the voxels in the voxel grid. The red lines represent two of the view-rays from the camera below the model, that cast upward through the model. Blue dots represent sample points where the spline model has been sampled, while green dots indicate samples from the voxel grid. Due to the area inside the spline model that has no voxels, the boundary accurate method has a larger sample distance between the first sample points on the rightmost view-ray. This leads to a somewhat different scalar value in this case, and therefore also a noticeable different resulting pixel color. The end result can be sharp transitions of color in the model, causing visual artifacts. 61
5.3 The thick method reduces the visual artifacts introduced by the thin boundary accurate method. 62
5.4 Visualization of the reference solution for Case A: Hybrid switchover. . 66
5.5 Case A: Hybrid switchover. Color differences for different methods compared to a reference solution. The methods based on using a voxel grid have used a texture size of 512^3. All the methods have used the same sampling distance of 0.008, and supersampling with 20 steps. 67
5.6 Hybrid ratio for Case A: Hybrid switchover. The ratio represents the number of samples from the spline model compared to the number of samples from the voxel grid. For pixels with black color, all of the samples have been sampled from the spline model. The lighter the color is; the more samples have been made from the voxel grid. 68
5.7 Case A: Hybrid switchover. Color differences for the boundary accurate methods and their corresponding hybrid methods. The methods based on using a voxel grid have used a texture size of 512^3. All the methods have used the same sampling distance of 0.008, and supersampling with 20 steps. 69
5.8 Graphs of the color differences for different methods in Case A: Hybrid switchover. 72
5.9 Visualization of the reference solution and the hybrid ratio for Case B: Far. 73
5.10 Visualization of Case C: Visual artifacts. The image shows a screen capture of the full 640x480 OpenGL viewport while rendering the reference solution. The geometric criterion is never met, meaning that the whole model is too close to the camera to use the voxel grid for any sample point. 75
5.11 Visualization of the model used in Case D: Multiple volume blocks. . . . 78

List of Tables

3.1 Color codes used for CIEDE2000 color difference. 23

4.1 Statistics for Case A: Trivial geometry, trivial scalar field. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. For the methods that use a voxel grid, the texture sizes are shown under #V. 48

4.2 Statistics for Case B: Trivial geometry, non-trivial scalar field. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. For the methods that use a voxel grid, the texture sizes are shown under #V. 51

4.3 Statistics for Case C: Non-trivial geometry, trivial scalar field. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. For the methods that use a voxel grid, the texture sizes are shown under #V. 53

4.4 Statistics for Case D: Non-trivial geometry, non-trivial scalar field. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. For the methods that use a voxel grid, the texture sizes are shown under #V. 55

5.1 Statistics for Case A: Hybrid switchover. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. The FPS column shows the performance in frames per second while rendering on an NVIDIA GeForce GTX 680 GPU, and is marked with a grey background when both of the threshold requirements are met. 71

5.2 Statistics for Case B: Far. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. The FPS column shows the performance in frames per second while rendering on an NVIDIA GeForce GTX 680 GPU, and is marked with a grey background when both of the threshold requirements are met. 74

5.3 Statistics for Case C: Visual artifacts. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. The FPS column shows the performance in frames per second while rendering on an NVIDIA GeForce GTX 680 GPU, and is marked with a grey background when both of the threshold requirements are met. 77

5.4 Statistics for Case D: Multiple volume blocks. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. The FPS column shows the performance in frames per second while rendering on an NVIDIA GeForce GTX 680 GPU, and is marked with a grey background when both of the threshold requirements are met. 79

Chapter 1

Introduction

1.1 Overview

Today, when a new car is made, the individual parts of the car are most likely designed digitally in computer-aided design (CAD) software. Before a part goes into production, it is analyzed to see how it reacts to stress and strain. The CAD model is typically converted to another representation that is suitable for this type of analysis. Creating this representation usually takes longer than the analysis itself for complex parts. In addition, the representation only approximates the CAD model. A recently proposed approach called *isogeometric analysis* allows for analysis directly on the CAD model, and thus have the same representation during the different stages in the engineering, see figure 1.1. Throughout the whole process, visualization is a helpful tool for both inspection and quality control. Visualizing the isogeometric models is challenging however, because the data used in analysis must be approximated for each sample point in the geometry. Because of the nature of isogeometric models, determining good approximations for the analysis data is computationally expensive. Fuchs and Hjelmervik [8] recently developed an approach for visualizing volumes based on isogeometric analysis in real-time. However, if the models are sufficiently complex, or a scene contains many isogeometric models, it may not be possible to achieve real-time performance using their approach directly. In this thesis I will look at ways of improving performance when rendering isogeometric models by looking at ways of creating reduced models, and level-of-detail methods for such models.

1.2 Motivation

Designs made in CAD software can express a host of different shapes, including models with smooth surfaces. The models are considered to be "exact", which is important in production when machines shape the model from raw materials. The industrial practice has been to convert the CAD models to meshes suitable for analysis. However, these mesh-based models are typically only approximations of the original

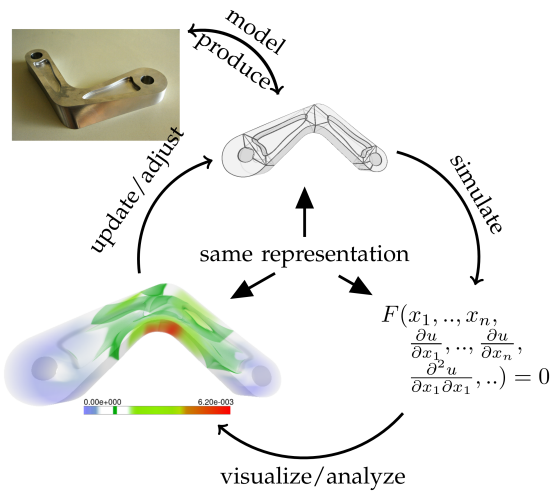


Figure 1.1: The different stages for isogeometric models. Visualization and diagram by Fuchs and Hjelmervik [8], model from the TERRIFIC project.

CAD models. This means that the results from the analysis may not be accurate. To increase the accuracy of the mesh-based models, it is typical to increase the resolution of the mesh at critical areas in the model. This usually takes time as it is difficult to do automatically. If a change is made in the design, the mesh must be made anew before it can be analyzed. Changes to the design are made on a daily basis in the automotive industry, and it has been estimated that as much as 80% of the total analysis time is spent creating these meshes [11]. A better solution would be to use the same model for both design and analysis. This is the motivation behind isogeometric analysis.

The results from analysis can basically be represented as numbers within a given range. With numerous detailed results, getting an overview can be difficult. By visualizing the results, much information can be conveyed in a short amount of time. Interactive visualization also allows the user to inspect effects made from changes in real-time. With isogeometric analysis, changes could be made to a CAD model, and the results from analysis could then be visualized directly. To rely on the visualization however, it is important that the results from rendering are visually accurate.

Visualizing isogeometric models is not a trivial task, and is computationally expensive. New demanding applications such as VR (virtual reality) requires both high resolution images and high frame rates to be optimal. It is therefore desirable to find methods to simplify the rendering process, but while keeping the result visually accurate. A popular technique in traditional 3D rendering is level-of-detail, where the 3D models are simplified, and the simpler versions are used when the user would not notice the difference compared to the original model. The same techniques are however not directly applicable to isogeometric models.

1.3 Research goal

Based on the research by Fuchs and Hjelmervik [8], the main goal for this thesis is to explore if there are ways of improving performance when visualizing isogeometric models. This is based on using simplified models, and using these in combination with the original model. While an alternative method should have better performance, it is also important that the result from rendering is visually accurate.

1.4 Outline

This thesis consists of six chapters, starting with this introduction. The following chapter gives a thorough background on the topics related to the thesis.

Chapter 3 introduces different ray-casting methods and discusses how these can solve the research problem. The following chapter describes how these were implemented in a 2D prototype, and the insight gained from the results after comparing them against each other. Chapter 5 contains details of the challenges when implementing the ray-casting methods in 3D, along with results from rendering different examples.

The last part concludes this thesis, with discussion about the results shown throughout the thesis and suggestions for future work.

Chapter 2

Background

2.1 3D models

In many 3D graphics applications, such as video games, the objects in a scene are represented by their surfaces. These surfaces can be thought of as infinitesimally thin shells that outline the objects, and are also known as boundary representations. The surfaces are typically comprised by a set of adjacent simple polygons such as triangles. Each triangle is defined by three vertices, and each vertex defines a point in the geometry, along with additional information such as normals, colors, etc. The whole model is stored as a mesh of all the vertices which are uploaded to the GPU before rendering.

Some phenomena are difficult to represent with boundary representations. Clouds and smoke for instance do not have a solid barrier and are also typically transparent to some degree. It is therefore difficult to define these based on a thin outer shell, as the information about the interior of the object is needed as well. These are therefore normally defined as volumes instead, which makes it possible to calculate light interactions at the interior of the objects.

Volume data are typically represented in a discrete grid. A uniform grid of discrete samples is called a *voxel grid*, which consist of many equally sized voxels. Voxel is short for "volume element", analogous to the 2D equivalent pixel, which is short for "picture element". Voxel grids may be represented as a grid of small cubes such as in figure 2.1.

Voxel grids can be stored on the GPU in so-called *textures*, as demonstrated by, e.g., Wilson, Van Gelder, and Wilhelms [22]. Textures are typically used to store images that are mapped onto surfaces, hence the name. They are not limited to this use however, and can more generally be thought of as data structures that are uploaded to the GPU. The individual elements of textures are called texels. Points in the texture are determined by texture coordinates, which normally ranges from 0 to 1. Doing a lookup in a texture at a given coordinate is called doing a *texture fetch*.

Voxel grids are discretized representations, which means that the data are not defined continuously. When doing a texture fetch, the coordinate will most likely not be at the exact same point where a texel has been defined. Normally this is solved by

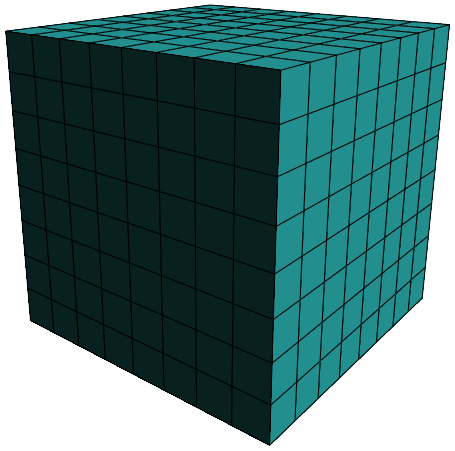


Figure 2.1: A voxel grid of size 8^3. Each voxel contains information about its own region in the volume.

interpolation. The value for a given coordinate is calculated using the nearest texels and their distances to the coordinate as weights. This has been implemented natively in hardware on modern GPUs.

Triangle meshes and voxel grids work well for representing objects with straight surfaces, but is not ideal for objects with curved surfaces. A workaround is to use more data points for curved surfaces, as many that are needed so that the surfaces appear to be curved. However, in some applications, such as engineering and manufacturing, there is a need for accurate models. Such models can be created using computational geometry.

Computational geometry

A designer use Computer Aided Design (CAD) software to create computational geometry. CAD models based on *splines*, such as NURBS (Non-Uniform Rational B-splines), have become the industry standard [18]. Splines are mathematical functions that are piece by piece defined with polynomials of degree p.

The term spline predates computer technology, and is the name given to flexible strips made from wood, plastic or metal. They were held in place with weights, and combined with the elasticity of the spline material, these strips would take a smooth shape that minimized the total bending energy. Draftsmen used these to draw smooth curves in designs by hand, especially in the shipbuilding industry.

Similarly to the draftsman's splines, the mathematical splines are also typically smooth. How the polynomials are pieced together is defined by parameters called *knots*. A nondecreasing sequence of these is called a knot vector. The points where the polynomial pieces connect can have up to $p - 1$ continuous derivatives, depending on the choice of knots. A spline can be represented as a linear combination of certain

basis functions, called *B-splines*. A spline curve can thus be defined as

$$\boldsymbol{f}(u) = \sum_{i=1}^{n} \boldsymbol{c}_i B_{i,p}(u). \tag{2.1}$$

The ith B-spline of degree p is defined recursively as

$$B_{i,p}(u) = \frac{u - \tau_i}{\tau_{i+p} - \tau_i} B_{i,p-1}(u) + \frac{\tau_{i+p+1} - u}{\tau_{i+p+1} - \tau_{i+1}} B_{i+1,p-1}(u), \tag{2.2}$$

where u is a real number and

$$B_{i,0}(u) = \begin{cases} 1, & \tau_i \leq u < \tau_{i+1}; \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

The parameters $(\tau_i)_{i=1}^{n+p+1}$ are the so-called knots. For a more thorough introduction to splines, see e.g., de Boor [5].

Given a set of points, the simplest way of constructing a curve would be to linearly interpolate the points, i.e., forcing the curve to pass through the points. In contrast, a spline can create a curve that is "influenced" by the points, and does not actually pass through them. The points in this context are often called *control points*, and the piecewise linear interpolation is called the *control polygon*, see figure 2.2. The degree of the polynomials dictates the degree of the spline. Higher degree splines have more flexibility, but can introduce more "wiggles". The most often used are cubic splines and are considered to have the best trade-off between stiffness and flexibility, but also compute time.

A property of spline curves is that they are contained in the *convex hull* of their corresponding control points. A shape is convex if for any two points within the shape, the whole line connecting them is also within the shape. Further, a convex hull for a set of points is the smallest convex shape that contains the points. This means that the maximum extent of the curve can be determined from the control points.

A volume or solid can be represented with a trivariate B-spline as

$$\boldsymbol{f}(u,v,w) = \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \sum_{i_3=1}^{n_3} B_{i_1,p_1}(u) B_{i_2,p_2}(v) B_{i_3,p_3}(w) \boldsymbol{c}_{i_1,i_2,i_3}, \tag{2.4}$$

where B_{i_1,p_1}, B_{i_2,p_2} and B_{i_3,p_3} are the B-spline basis functions of degree p_1, p_2 and p_3 respectively. The B-splines are defined with the underlying knot vectors $\boldsymbol{\tau} = \{\tau_1, \tau_2, \ldots, \tau_{n_1+p_1+1}\}$, $\boldsymbol{\sigma} = \{\sigma_1, \sigma_2, \ldots, \sigma_{n_2+p_2+1}\}$ and $\boldsymbol{\omega} = \{\omega_1, \omega_2, \ldots, \omega_{n_3+p_3+1}\}$. The convex hull property of B-spline curves also holds for B-spline solids.

If the parameters are limited to given intervals, such that $u \in [u_a, u_b]$, $v \in [v_a, v_b]$ and $w \in [w_a, w_b]$, then the parameters are contained inside a rectangular cell. This cell is called the parameter domain, here denoted P. By fixing a parameter and let the other two parameters vary, we have an axis-aligned surface. Doing this for all three parameters and for each of the endpoints of the intervals, we get the six boundary

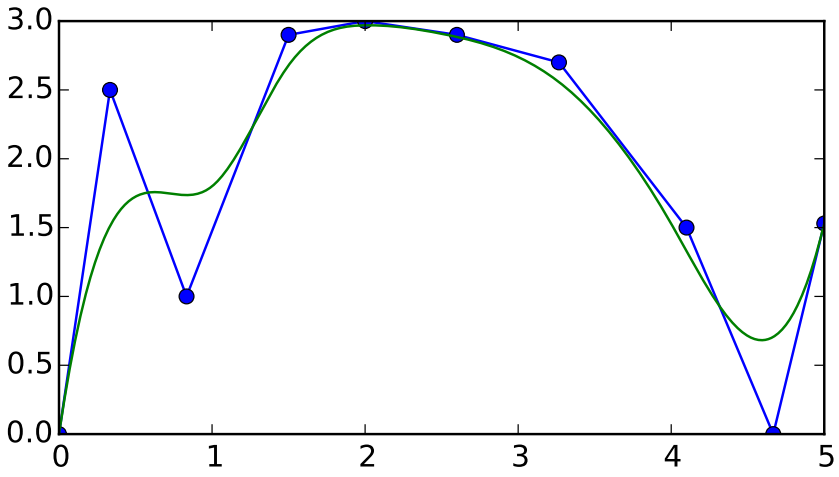


Figure 2.2: A spline curve of degree 3 shown in green, and the corresponding control polygon in blue. The blue dots mark the control points.

faces of the cell. All these faces map to surfaces in the geometry domain, here denoted G. Since the faces share boundaries in P, the surfaces in G also share boundaries and create a closed volume.

2.2 Isogeometric analysis

The purpose of analysis in engineering is to reveal potential issues in products or systems. For instance, if force is applied to an object, it would be desirable to know how this affects the structural properties of the object. It would then be necessary to be able to analyze stresses, strains or displacements at each material point. To solve this analytically, it is necessary to solve multiple partial differential equations, which cannot be done in practice. Approaches that approximate the exact solution are therefore used instead.

Finite element analysis (FEA) is a numerical method that has become widely used in engineering analysis. A complex model is discretized into a finite element model, which is a mesh of so-called elements. The accuracy of the mesh depends on the discretization interval, and may vary throughout the material. Usually, the density of the mesh is higher in areas where it is anticipated more change, such as in stress levels. The elements contain material and structural properties at the areas they represent in the model. This is defined by simple functions that approximates the true function of the original model.

Since the mesh only approximates the model, there may be some loss of accuracy. This means that if the approximation is not good enough, there is a possibility that the result from analysis does not match "the reality". A part of creating these meshes is to

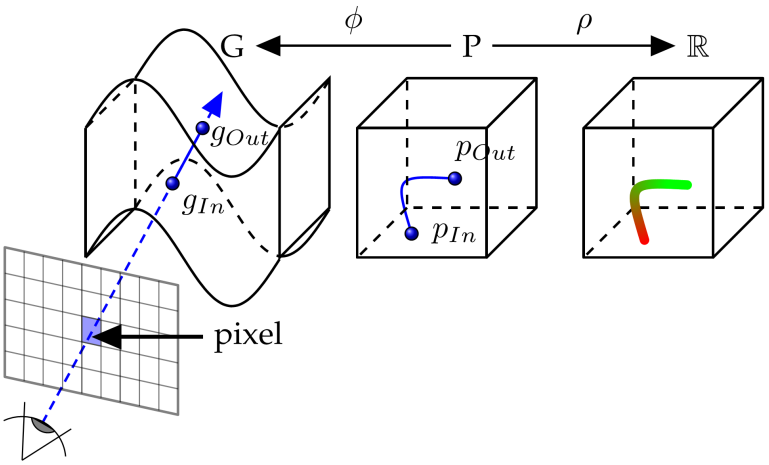


Figure 2.3: Rendering isogeometric models. The geometry is denoted G, while P is the parameter domain. Diagram by Fuchs and Hjelmervik [8].

identify where the areas of interest are, so these can be presented more accurately in the mesh. If an issue is revealed while analyzing, the CAD model can be updated to solve the issue at hand. It would however then be necessary to create a new mesh based on the updated CAD model. Creating these meshes can be quite time consuming, and can take as much as four months for an entire vehicle in the automotive industry [11].

Isogeometric analysis, often abbreviated IGA, is a newly developed approach for doing analysis directly on CAD models. It was pioneered by Hughes, Cottrell, and Bazilevs [11]. Isogeometry basically means "having the same geometry", so IGA could therefore be thought of as "doing analysis on the same geometry". The idea behind is to represent the fields, which describes physical values such as temperature or velocity, by using the same basis functions that defines the geometry.

The geometry is represented through a spline, here denoted ϕ, while the field is represented by a second spline ρ. Both ϕ and ρ are defined on the same parameter domain P, see figure 2.3. In three dimensions, the parameter domain P is a rectangular cell $P_u \times P_v \times P_w \subset \mathbb{R}^3$. The corresponding geometry $G \subset \mathbb{R}^3$ is a closed volume of arbitrary shape. The faces of P maps to surfaces in G.

A set of ϕ and ρ defines a so-called *volume block*. In some cases, it is not possible to create the whole model from one coherent volume block. A typical case is if the physical object that is to be modeled has holes in it. An isogeometric volume may therefore consist of multiple adjacent volume blocks. This may also be done to improve the parametrization of the model [8]. Industrially relevant isogeometric models consist of many such blocks.

To be able to analyze the interior of a block it can be visualized as a semi-transparent volume. Typically, some of the volume blocks are completely transparent. Before looking at how to visualize isogeometric models, some background is needed on scientific visualization.

2.3 Scientific visualization

Scientific visualization is the practice of graphically illustrating scientific data. The purpose is for scientists to get an improved understanding and insight. The data can come from a variety of different fields. One of the early fields that saw the use for scientific visualization was medical imaging. CT (computerized tomography) and MRI (magnetic resonance imaging) are examples that generates data by scanning the body of a patient.

Typically, the data is defined as a discretely sampled 3D grid of scalars. With discretized data we want to be able to reconstruct the underlying continuous function. How accurately this can be done is dependent on the number of samples available. Sampling theory tells us that to be able to correctly reconstruct a continuous signal from a discrete signal, the sampling frequency must be greater than twice the highest frequency of the input. This is called the Nyquist sampling rate, see e.g., Higgins [9] for more details. Reconstruction between discrete samples is typically done by interpolation.

The objective is to project 3D data onto a 2D screen. A simple approach for scientific visualization is to only render the surfaces of objects. This approach is quite similar to rendering triangle meshes, and is therefore also typically computationally inexpensive. The main drawback is that no information about the interior of objects are shown, although this approach can be ideal when the interior is mostly uniform.

Another approach to visualizing 3D data is called *isosurface extraction*. An isosurface is a 3D surface where the underlying data have a particular scalar value, called the isovalue. The isosurface separates regions with higher and lower scalar values. The volume inside the surface contains scalar values that are greater (or less) than the chosen isovalue, while the volume outside has the opposite. The marching cubes algorithm [15] is an often used technique for isosurface creation. The algorithm creates an intermediate polygonal representation based on discretely sampled 3D data. The polygonal representation can then be efficiently rendered on a GPU. Since it is the intermediate representation that is visualized, and not the 3D data itself, isosurface extraction is also called indirect volume visualization.

The goal of direct volume visualization is to use the volume data directly to render each image without any intermediary geometrical representations. To achieve realistic imagery, volume rendering is based on the physical model for light transport. An often used simplified model is the so-called emission-absorption model. In this model the participating medium can both emit and absorb light, but scattering and indirect illumination are omitted. It can be expressed as the *volume-rendering integral*

$$I(d) = I(0)T(0,d) + \int_0^d q(t)T(t,d)\,\mathrm{d}t, \tag{2.5}$$

where I is the accumulated radiance. $T(t,d)$ specifies absorption of light energy from t to d, and $q(t)$ defines the emission at t. An illustration of absorption and emission is

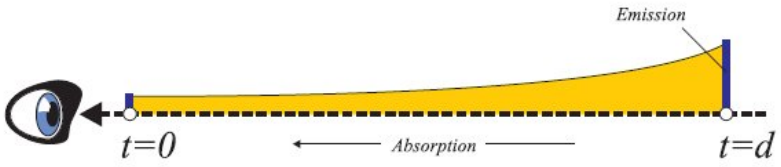


Figure 2.4: In the simplified model, only absorption and emission affect the radiation along a view-ray. (Wenke and Vornberger, 2010)

shown in figure 2.4. In practice the integral is approximated as there are no closed-form solutions known. This is commonly done by splitting the integral into intervals, and finding the approximate for each interval. The intervals may be of different lengths. The approximation can be written as

$$I(d) = \sum_{i=0}^{n} C_i \prod_{j=i+1}^{n} T_j, \text{ where } C_0 = I(0). \tag{2.6}$$

Here C_i denotes the color and T_i denotes the transparency for the ith segment. For a more in-depth description, see e.g., Engel et al. [7]. 3D graphics APIs most commonly represent the optical properties as RGBA values. The emission is represented by the colors red, green and blue (RGB), and the absorption is represented by an alpha value (A).

Ray-casting is a commonly used approach for direct volume visualization. It is an image order technique where the image is generated pixel by pixel. The basic idea is to cast a view-ray from an imaginary eye, through each pixel of the screen, and into the scene. More formally, a view-ray γ is defined by

$$\gamma(t) = \boldsymbol{O} + t\boldsymbol{v}, \tag{2.7}$$

where $\boldsymbol{O}$ is the origin (eye) and v is a unit vector pointing in the direction of the ray.

Color values are determined by sampling inside the volumes along each ray, and these are composed to the pixel's final color value. The approximated intervals of the volume rendering integral are composited to generate the final color. The most common schemes for compositing are front-to-back and back-to-front compositing. These compose the interval colors in sequential operations, and can be efficiently implemented on modern graphics hardware.

Front-to-back compositing has a distinct advantage compared to back-to-front compositing when ray-casting. It is used when the view rays are traversed from the camera into the volume. As the pixel color is composited along a ray, eventually the colors at sample points have little to no influence on the final color. This means that sampling along a ray can in some cases stop before the whole volume has been traversed, without the end result being any different. This is called *early ray termination*. Readers interested in more details about compositing may find more information in, e.g., Engel et al. [7].

The equations for the compositing schemes depends on whether or not the colors are *associated*. This term was introduced by Blinn [2], and the color components of these colors have already been premultiplied with the opacity. Here, only nonassociated colors will be covered, and the front-to-back compositing equation can then be expressed as

$$\begin{aligned} T &\leftarrow (1-\alpha_{src})^{\frac{\Delta s_i}{\ell}}, \\ C_{dst} &\leftarrow C_{dst} + (1-\alpha_{dst})(1-T)C_{src}, \\ \alpha_{dst} &\leftarrow \alpha_{dst} + (1-\alpha_{dst})(1-T). \end{aligned} \tag{2.8}$$

Here, C represents the color components, typically RGB, α is the opacity, and ℓ is the standard length. The standard length determines how quickly the color is satiated based on a sampling distance.

So far it has been assumed that the optical properties, the emission and the absorption coefficients, have been available directly in the 3D data of the volume when compositing. However, in scientific visualization the 3D volume data are usually scalar. Determining the optical properties based on the scalar values is a process called *classification*.

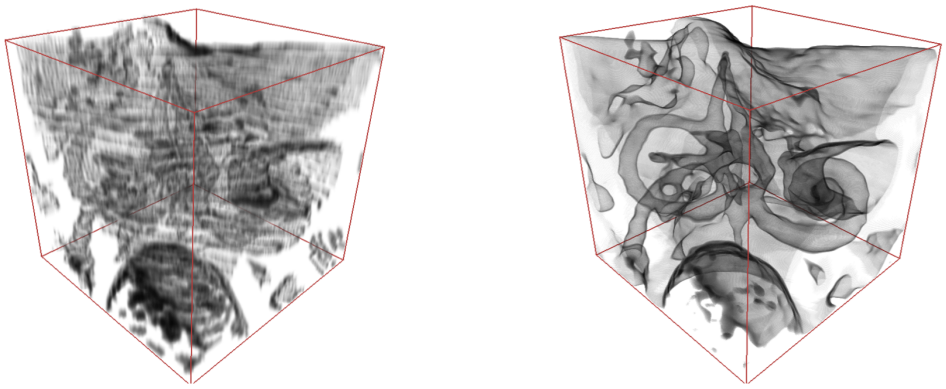
Classification

Classification is to determine what the different scalar values in 3D data actually represent. For example, in data from a CT scan of a human, the skin, the bones and the veins all have different scalar values, and the user may want to classify these so that they are visually distinguishable. This means that different scalar values are assigned different colors, a process called color mapping. This is realized by a *transfer function*. A 1D transfer function may be defined as $(C_i, \alpha_i) = TF(s_i)$, where s_i is a scalar value and TF is the transfer function. Some applications use two separate transfer functions, one for the color components and one for the opacity value. Higher dimensional transfer functions are also possible to achieve more flexibility.

Although there exist approaches to automatically generate transfer functions, it is generally done manually due to the requirement of detailed knowledge of a given data set [7]. It is therefore desirable to be able to modify the transfer function in real-time while visualizing the volume.

At which stage the classification is done gives visually different results. Under pre-classification, the transfer function is applied before any interpolation. This means that the scalar values are passed to a transfer function directly, and the resulting colors are stored in a volume. To find the color of a given point in the volume, the nearby colors are interpolated.

Alternatively, the transfer function can be applied after interpolation. This is called post-classification. The volume representation would then consist of the scalar values. To find the color of a given point in the volume, an interpolated scalar value is first retrieved, and then the transfer function is applied.



(a) Pre-classification (b) Post-classification

Figure 2.5: Visualization of a middle ear with pre- and post-classification. Figure from Rezk-Salama and Hastreiter [27].

The transfer function may introduce additional high frequencies. This means that even if the discretized volume data are a result of sampling at or above the Nyquist rate of the scalar field, high frequencies in the transfer function can lead to inaccurate reconstruction. Supersampling can be used to deal with these high frequencies. Between two scalar values for two sample points, an additional n linearly spaced samples are added.

To overcome the problem of high Nyquist frequencies, a third option is by using pre-integrated transfer functions. The basic idea is to create a lookup table that takes two scalar values as arguments. The sampling distance must however be constant and known before creating the lookup table. A color is then stored in the lookup table based on calculating the volume rendering integral. This table is then used to determine the color for segments along the view-ray. The lookup table is pre-calculated, which means that during rendering the color of a ray segment is determined by a $O(1)$ lookup. Even if the transfer function is nonlinear, pre-integrated classification allows sampling the underlying scalar field without needing to increase the sample rate. This can lead to increased performance, but can also lead to a more accurate result. A disadvantage of pre-integrated classification is that the lookup table must be regenerated when the transfer function changes.

2.4 Visualizing isogeometric models

Having a tool for visualizing isogeometric models is useful while analyzing, such as for quality inspection. In particular, it is scientific visualization of these models that is of interest. The 3D data is defined by continuous functions, which differs from traditional volume rendering where the volume data contain discrete samples. This means that the traditional methods for volume visualization cannot be used on isogeometric models

directly.

Determining the optical properties for a point is dependent on the field data at the point. One of the main challenges in visualizing isogeometric models is finding the correct field data value for a point in the geometry. Since both the geometry function ϕ and the field function ρ are defined on the parameter domain P, the field data values are not directly available from the geometry. Mapping points in the geometry to points in the parameter domain is equivalent to finding the inverse of ϕ. The problem is that there is no closed-form solution known for the inverse of a spline. It can however be approximated numerically, for instance with the Newton-Raphson method. The method is used to find the parameters of a function that approximates the roots of the same function. It has quadratic convergence given that the initial guess x_0 is good. For a univariate function $f(x)$, the equation

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2.9}$$

is solved iteratively until the solution is sufficiently accurate. For a trivariate function $\boldsymbol{F}(\boldsymbol{v})$ where $\boldsymbol{v} = (x, y, z)^T$ and $\boldsymbol{F}(\boldsymbol{v}) \in \mathbb{R}^3$, the derivative of the function is expressed with a Jacobian matrix

$$\boldsymbol{J_F} = \begin{bmatrix} \frac{\partial \boldsymbol{F}}{\partial x} & \frac{\partial \boldsymbol{F}}{\partial y} & \frac{\partial \boldsymbol{F}}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{\partial F_1}{\partial x} & \frac{\partial F_1}{\partial y} & \frac{\partial F_1}{\partial z} \\ \frac{\partial F_2}{\partial x} & \frac{\partial F_2}{\partial y} & \frac{\partial F_2}{\partial z} \\ \frac{\partial F_3}{\partial x} & \frac{\partial F_3}{\partial y} & \frac{\partial F_3}{\partial z} \end{bmatrix}, \tag{2.10}$$

where $\frac{\partial F_j}{\partial x}$ denotes the partial derivative of $\boldsymbol{F}$ with respect to x, and the result is the j-th element of the output vector. To solve equation 2.9 for a trivariate function it is necessary to find the inverse of the Jacobian matrix, which can be a numerically unstable operation. Alternatively, an iteration of the Newton-Raphson method can be expressed as a system of linear equations

$$\boldsymbol{J_F}(\boldsymbol{x}_n)(\boldsymbol{x}_{n+1} - \boldsymbol{x}_n) = -\boldsymbol{F}(\boldsymbol{x}_n), \tag{2.11}$$

which is solved for $(\boldsymbol{x}_{n+1} - \boldsymbol{x}_n)$. Here, $\boldsymbol{x}_n$ is an approximation to the root of $\boldsymbol{F}$.

Another challenge is that evaluating splines is computationally expensive. This makes sampling computationally expensive as well, which affects real-time performance.

Iso-surface extraction

In section 2.3, a brief description of iso-surface extraction was given related to traditional scientific visualization. The typical approach is to use the marching cubes algorithm. However, this algorithm expects discretely sampled 3D data. Since isogeometric models are defined with continuous functions, this algorithm cannot be used directly.

Another algorithm for iso-surface extraction was presented by Martin and Cohen [17]. Their approach is tailored to trivariate spline volumes, along with so-called attribute descriptions. The paper predates Hughes, Cottrell, and Bazilevs [11] where the term isogeometry was coined, but Martin and Cohen's "attribute description" is similar to the field in isogeometric analysis. Finding an iso-surface is formulated as finding the zeros to a function. The volume is then subdivided into smaller volumes until the control points of each volume partition meets a certain criterion. At the same time a list is constructed containing the subvolumes that contain the roots of the function. Finally, the roots are approximated by using the Newton-Raphson method, and a polygonal approximation is created to visualize the iso-surface.

Volume rendering

Martin and Cohen [17] also proposed an approach for ray-casting isogeometric volumes. In their approach, the volume is ray-casted starting from the camera and then by traversing through the volume. The first step is to find the first point where the view-ray intersects with the volume. The problem is formulated as implicit equations, and by using the Newton-Raphson method approximated solutions are found. In a pre-process step, multiple hierarchies of bounding volumes are created for the volume. When ray-casting, the bounding volumes that can not contain any intersection are culled. Parameters inside the remaining bounding volumes are used as initial guesses for the Newton-Raphson method. The result is a list of $2n$ intersection points, where n is a non-negative integer. The attribute data at the first intersection point are evaluated to obtain the optical properties. The volume is then traversed, starting at the first intersection point. Successive points are determined by traversing a small distance along the view-ray. At each point the Newton-Raphson method is used to approximate the parameters, and these are used to find the optical properties. The colors and opacities are accumulated by front-to-back compositing.

Fuchs and Hjelmervik [8] recently developed an approach for visualizing isogeometric volumes in real-time. Similarly to [17], the first step is finding the ray-surface intersections. Instead of solving by finding zeros of a function, Fuchs and Hjelmervik restates the problem by that of finding approximate surfaces of the volume. This approach can be efficiently implemented on a GPU, as demonstrated in [10]. Since the ray-surface intersections are found in parallel, they must subsequently be sorted according to the distance from the camera. Internal points of the volume are then sampled along the view-ray. In addition to approximating ϕ^{-1} by the Newton-Raphson method, Fuchs and Hjelmervik proposed an alternative based on ordinary differential equations (ODEs).

Each pixel's color is determined by approximating the volume rendering integral. Radiance at sample points are determined by scalar values from the volume.

In the spirit of isogeometry, it is desirable that the visualization is accurate as well. In the approach to volume visualization by Martin and Cohen, due to the approximation of ϕ^{-1} the corresponding parameter can have any arbitrary value. The

approach by Fuchs and Hjelmervik guarantees that the information used for the colors of a pixel comes from the frustum of the given pixel, i.e., the visualization is *pixel-accurate*.

Pixel-accuracy

For a given point g in the geometry, the corresponding color must be found for that point. The color is dependent on the transfer function used, and the scalar value s for the point. Since the scalar function ρ is defined on the parameter domain P, the parameter p must first be found, which maps to g. Since there is no closed-form solution known for ϕ^{-1}, it must be approximated. Depending on the approach for approximating and how accurately a solution is approximated, there can be multiple distinct approximations $\tilde{p} = \phi^{-1}(g)$. These parameters can potentially map to any point in the geometry by $\tilde{g} = \phi(\tilde{p})$, which is not desirable.

To render isogeometric objects correctly, certain requirements must be fulfilled when sampling from the geometry. Yeo, Bin, and Peters [23] proposed a definition for pixel-accuracy based on *parametric accuracy* and *covering accuracy*. Fuchs and Hjelmervik [8] further defined pixel-accurate rendering of isogeometric objects. In their definition, a point $\tilde{p}$ fulfills the requirement of parametric accuracy if the mapped point in the geometry $\tilde{g}$ projects into the pixel of the current view-ray. This basically means that the geometric point must be guaranteed to be within the current pixel's frustum. Covering accuracy is achieved by assuring that the sample points have the correct depth order along the view-ray. In other words, after orthogonally projecting all the sample points onto the view-ray, each subsequent sample point would be further away from the camera than the previous sample point.

Volume visualization can be used to represent light interactions at the interior of objects. However, it is generally more computationally expensive than surface-based rendering methods such as visualization of isosurfaces. Speed-up techniques such as *level-of-detail* may be essential to achieve acceptable real-time performance.

2.5 Performance and memory

2.5.1 Level-of-detail

Level-of-detail, or LOD for short, is the idea of reducing the complexity of objects before rendering. The intention is to increase performance by reducing the overall workload of the graphics pipeline. Objects that are far away from the viewer for instance, can be replaced with simpler objects without the viewer noticing.

Luebke et al. [16] categorizes level-of-detail into *discrete*, *continuous* and *view-dependent* LOD. Discrete LOD uses multiple versions of each object, where each version has a different level of detail. These are generated either manually or automatically as a separate process before the actual rendering. During rendering an object is visualized by one of these versions, and changed to another on-the-fly when appropriate.

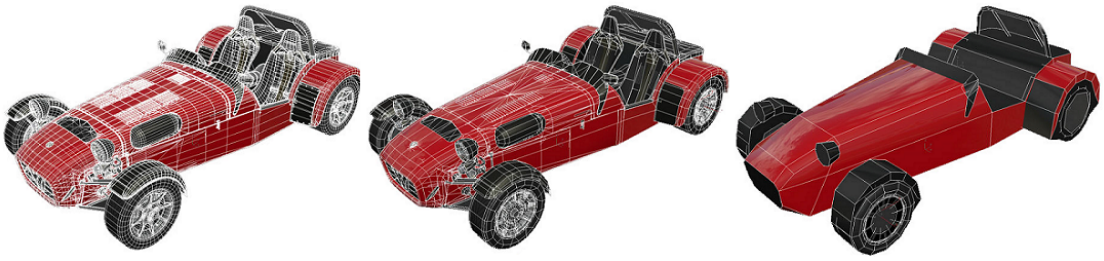


Figure 2.6: Discrete LOD example for a boundary representation model. A lower LOD is used as the model of the car the further away it gets from the camera. Figure from Project CARS [26].

However, the transition from one discrete representation to another cannot be done smoothly, and it can in some cases be visible by the viewer. An example of discrete LOD is shown in figure 2.6.

A more gradual change between levels of detail can be achieved by using continuous LOD. Under simplification, a hierarchical data structure is created with the original object at the top. Each lower level in the structure has a decomposition of the object at the level above. While rendering, the desired level of detail is generated from this structure. View-dependent LOD is also a form for continuous LOD, but the criteria for simplification is view-dependent. A single object can thus be represented by multiple levels of detail simultaneously. This is more ideal for large objects, where some parts of the object may be close to the camera, while others may be far away at the same time. View-dependent LOD also allows for keeping a high degree of detail close to the outline of an object, even if the object is far away. This can prevent noticeable inaccuracies in the outline when reducing the level of detail.

Discrete simplifications of textures can be created with hardware acceleration. This type of LOD is called *mipmapping*. As described by Williams [21], the basic idea is to have the original texture accompanied with pre-generated smaller versions of the same texture. The norm is for each smaller texture to be half the size of the previous texture in each dimension. This continues until the last texture is a single texel. For instance, mipmapping a 16x16 2D texture will generate 8x8, 4x4, 2x2 and 1x1 sized subtextures.

La Mar, Hamann, and Joy [13] demonstrated level of detail for volume visualization. In their approach, a model is partitioned into so-called *bricks*. The bricks may be of different sizes, and a hierarchy of them is stored in an octree. Each brick is represented by a texture. Regions of interest are represented by more bricks than other regions. The original data is defined at the leaves of the tree, and internal nodes defines version with lower resolutions. During rendering, the bricks are chosen according to the distance from the camera and the view frustum. To represent volumes accurately at the finest level, it would typically be necessary to use large textures. The textures reside in the GPU memory while rendering, which is a limited resource.

2.5.2 GPU memory

So far the problem of scarce GPU memory has briefly been discussed. Whether a voxel grid is stored in one large or several smaller sized 3D textures, even modern GPUs will eventually run out of memory when the models get sufficiently large. In this section, some existing proposed solutions will be discussed.

GigaVoxels

Crassin et al. [4] proposed an approach that allows real-time rendering of voxel-based models larger than the size of the video memory. In their approach, voxels are stored in an octree structure which subdivides the volume hierarchically. Each node in the octree indicates either homogeneous or empty space, or points to a so-called *brick*. The bricks are small voxel grids of a predetermined size. At the leaf nodes in the octree, the bricks represent a portion of the volume at the highest resolution. The brick at an internal node represents the volume of all of its children, but at a lower level of detail. While ray-casting, the distance to the eye decides what level from the octree to sample from.

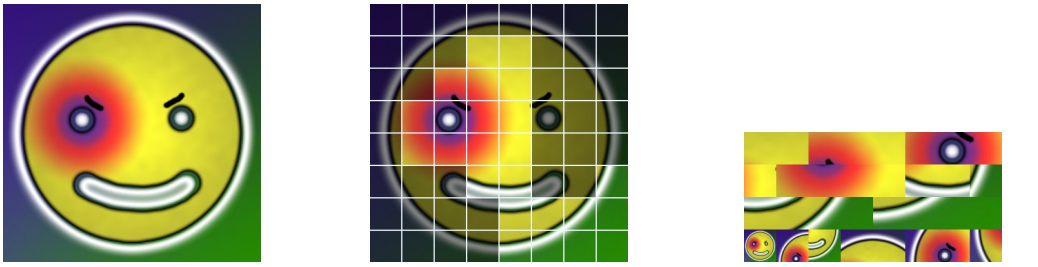
A brick may also be absent from GPU memory. If such a brick is requested while rendering, an update is triggered, and the CPU uploads the requested data to the GPU. The new data replaces the brick(s) that were used least recently in the GPU memory.

There are some disadvantages to this approach, however. The voxels are not accessed directly as in a mipmapped 3D texture, but through the octree. This means that the octree must be traversed for each sample point along a ray. In addition, the different LODs are stored in the same memory pool. Hardware accelerated interpolation between mipmaps can therefore not be utilized. These problems can be solved by the use of *sparse textures*.

Sparse textures

Sparse textures, also known as Partially Resident Textures, were recently implemented as an extension to OpenGL. The concept behind is that only a portion of a texture needs to be resident in the GPU video memory. When using sufficiently large textures, not all of the texture content will be visible at any given time. By virtual addressing, an application can manage textures larger than the actual texture allocated in video memory. When a portion of the texture that is not currently addressable by the GPU is needed, it signals the application which then can upload an updated texture. Mipmaps of the original texture can also be stored in the same virtual texture. The application can then specify the lowest level of detail allowed, so that the GPU may use a mipmapped texture while waiting for a higher level of detail texture to be uploaded from the application.

Similarly to the approach proposed by Crassin et al. [4], regions of the texture may be marked as non-resident. This means that for such regions there are no higher level of detail available. Empty regions or regions with the same values in a voxel grid can thus be marked as non-resident.



(a) The original texture

(b) Parts of the texture that has visible texels

(c) Texture in GPU memory

Figure 2.7: A small example of a 2D sparse texture. Figures from Barrett [25].

An example on a small 2D texture is shown in figure 2.7. The texture is partitioned into fixed-size *pages*, shown as a grid in figure 2.7b. If any of the texels from a page is needed, the entire page must be present in GPU memory. Notice that the mipmapped textures are also stored in the same texture in figure 2.7c. Sparse textures are also extensible to 3D textures.

Chapter 3

Methods for ray-casting

This chapter explores ways of improving performance when visualizing isogeometric volumes. Different methods for ray-casting are then described. These have different assurances for visual accuracy, which also affects the performance.

3.1 Scope

In section 2.4, different approaches for visualizing isogeometric models were described. Visualizing these as volumes is perhaps the most useful under analysis since not only data about the surfaces of the object are shown, but also for the internals. In this thesis the focus will therefore be on volume visualization. Fuchs and Hjelmervik developed an approach for ray-casting isogeometric volumes interactively, but the performance will deteriorate if a scene contains many isogeometric models. However, to the best of my knowledge, no other implementation exists that is able to visualize such volumes in real-time. The research in this thesis will therefore be based on the same approach for visualization as the one introduced in their paper [8]. Similarly to [8], the isogeometric models will also be restricted to B-splines, and the field is assumed to be scalar. In addition, only one dimensional transfer functions are used in the examples and results presented here.

3.2 Validating methods

In this thesis, different ray-casting methods are proposed. To evaluate them, there must be a way of validating the results from ray-casting. Measuring time is a natural way of seeing how each method performs compared to each other. However, it is important that the visual results are also validated. It is easy to create a method which performs good but where the visual results are inaccurate. In the spirit of isogeometry it is desirable to maintain the accuracy in the visualization. This depends on the approximations that are done and the margins of error related to these.

3.2.1 Factors affecting accuracy

To visualize isogeometric volumes, several approximations must be made. Section 2.4 described how it is necessary to approximate the inverse of a spline to sample at a given point in the geometry. In section 2.3, it was discussed how to approximate the volume rendering integral. Finally, it was explained that a discretization such as a voxel grid would need to approximately reconstruct the field by a filter. These factors affecting accuracy can be summarized as follows:

- Inverse of geometry
- Volume rendering integral
- Reconstruction of scalar field (interpolation)

3.2.2 Evaluating visual results

Ideally, the isogeometric objects should be rendered with a small room for error as cheap as possible. Since the inverse of the geometric function must be approximated, there will be a margin for error regardless of the approach chosen. Different methods lead to different algorithms for ray-casting, which can lead to different colored pixels from rendering. To evaluate the visual results of each method, the results need to be compared to a reference solution. The reference solution should be the most accurate solution possible, which means that it should have very good approximations for the potential inaccuracies listed above. Comparing the approximation and quality of different approaches is done by comparing each pixel to one or more corresponding pixels in the reference solution. To compare the colors objectively, a quantifiable measurement is needed for the color difference.

Color difference

The Commission Internationale de l'Eclaraige (International Commission on Illumination, CIE) is an organization that standardizes color metrics. They have defined a distance metric between colors which they call ΔE. For a more in-depth description about color difference, see e.g., Sharma and Bala [19]. CIE recommends the CIEDE2000 algorithm to calculate ΔE [14]. CIEDE2000 is currently the latest standard proposed by CIE, and is recommended as a replacement for previous standards such as the CIE 1976 formula and the CIE 1994 formula. The algorithm calculates the color difference between colors in the L*a*b* color space. In computer graphics the RGB color space is mostly used, but conversion to L*a*b* is possible.

ΔE is a scalar where 0 essentially means that the two compared colors are the same color, and the greater the color difference is, the greater ΔE is. When testing different approaches, all the pixel color differences should be lower than a certain limit. This limit is ideally where it is no longer possible to see any difference between the compared colors. It has been considered that if the ΔE values calculated between colors are below

ΔE	Color	Description
$\in [0, 1]$		No noticeable color difference
$\in (1, 5]$		Acceptable color difference
$\in (5, 10]$		Noticeable color difference between most color pairs
$\in (10, +\infty]$		Clear color difference between color pairs

Table 3.1: Color codes used for CIEDE2000 color difference.

1.0, the color differences are not visible to the human eye. Different ΔE values have been proposed that cause a "just noticeable difference" between colors, but a common threshold is 1.0. Similarly to Fuchs and Hjelmervik [8], a threshold of 1.0 will be used here for the mean color difference, and a threshold of 5.0 for the maximum. The argument for the higher maximum threshold is that when the mean is below 1.0, and there are sufficiently many pixels on the screen, then only a small number of pixels may have a slightly different color. The visual perception of the result will not be changed in any significant way. To visualize the ΔE values in this thesis, they have been color coded. In the illustrations shown here, pixels are colored according to table 3.1.

3.3 Direct method

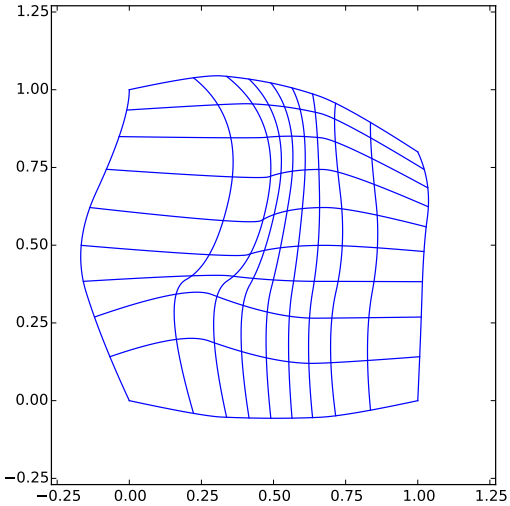
Fuchs and Hjelmervik [8] presented an approach for ray-casting isogeometric models. Since the isogeometric models are used directly, their method will here be referred to as the *direct* method. The visual accuracy depends on how good the approximations are, and to achieve real-time performance the approximations are considered good enough when there is pixel accuracy. Since using the isogeometric model directly generates the most accurate visual results, the reference solution will also be based upon this method. The approximations for the reference solution can however be made more accurate than what is feasible when rendering interactively. This involves better approximations of the volume rendering integral and the parameters of each sample point.

A 2D example of an isogeometric model is shown in figure 3.1. The geometry defined by the spline ϕ is visualized in figure 3.1a, while figure 3.1b is a visualization of the scalar field defined by the spline ρ. Both ϕ and ρ have been defined on a parameter domain $P_u \times P_v$ where $u \in [0, 1]$ and $v \in [0, 1]$.

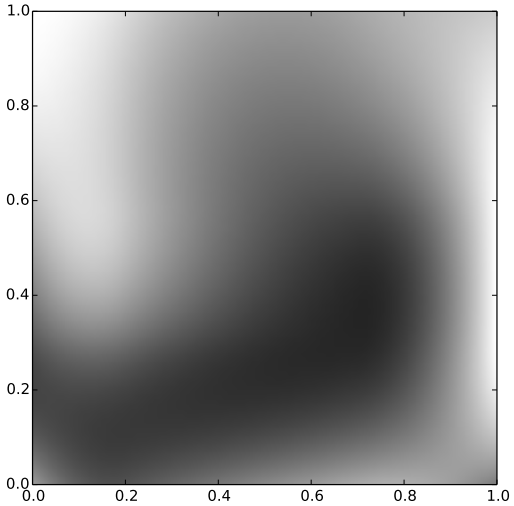
While ray-casting, the isogeometric volume is sampled directly. For each view-ray that has two intersections, the next step is to traverse the model starting at $g_0 = g_{In}$, where g_{In} is the intersection point closest to the camera. Point i in the geometry is determined by

$$g_i = g_{i-1} + \Delta s_i \frac{g_{Out} - g_{In}}{\|g_{Out} - g_{In}\|}, \tag{3.1}$$

where s_i is the sampling distance for the ith segment, and $\|v\|$ denotes the Euclidean distance of the vector v. Since ϕ^{-1} must be approximated, g_i is only an ideal sample



(a) The geometry defined by the bivariate spline ϕ. The grid consists of isoparametric curves, a result from fixing one parameter to a value, and let the other vary. This has been done for both parameters and for ten uniformly spaced different fixed values.



(b) The scalar field defined by ρ. The greyscale color is determined by using the scalar value for all three color components (RGB).

Figure 3.1: Visualization of the volume block from dataset 1.

point. The actual sample point is $\tilde{g}_i = \phi(\tilde{p}_i) = \phi(\phi^{-1}(g_i))$, as explained in section 2.4.

So far the direct method for ray-casting an isogeometric model have been described. The problem is that rendering isogeometric volumes with this method is computationally expensive. Level-of-detail methods have already been proposed for traditional volume rendering [4, 20]. Since both the geometry and field of isogeometric models are given through splines, the earlier proposed approaches cannot be used directly. Similarly to LOD for traditional volume rendering, it is necessary to create reduced approximate models to realize LOD for isogeometric models.

3.4 Reduced models

In this section different approaches for creating reduced models that approximates isogeometric models will be discussed. One of the reasons why visualizing isogeometric volumes is computationally expensive is because of the spline evaluations needed at each sample point. The first approach that will be discussed is based on generating simpler spline-based models.

3.4.1 Models based on splines

B-splines are computed recursively, as shown in equation (2.2). The lower the degree of the spline, the fewer steps will be necessary to evaluate it. An algorithm for degree reduction of B-spline curves has been presented by, e.g., Yong et al. [24]. Reducing the degree will however alter the shape to some extent. How many times we can perform a degree reduction will be dependent on the original degree and can be severely limited. Degree reductions affect the whole spline, so it can only create discrete simplifications.

Another option is to reduce the amount of control points. Fewer control points make the spline simpler and lead to faster evaluation as there are fewer terms in equation (2.1). The number of control points is affected by how many knots there are. Eck and Hadenfeld [6] have proposed an algorithm for knot removal. But as they mentioned in their article, this will also alter the shape of the spline curve. Knot removal can be done locally, which means that a spline can have multiple levels of simplification.

Both of these approaches reduce the complexity of the model. However, they create models that are no longer a true representation of the original model, they will only be approximations. They may be appropriate in a LOD system, since some deviations from the original model are acceptable at lower levels of detail. Working with splines is still computationally burdensome however, so it is also worth taking a look at discretized volume representations.

3.4.2 Simplices

In traditional volume rendering, volumes consist of discrete samples that are structured in a grid. An example is a grid of so-called simplices. Simplex grids can span arbitrary dimensions. In 2 dimensions, each simplex would be represented by a triangle, and in 3D the simplices are represented by tetrahedrons. An example use case for simplices are the elements in finite element analysis (FEA). The simplex grids are non-uniform, which means they are flexible and can match arbitrary volumes closely. This can be ideal for approximating isogeometric models, which typically have smooth surfaces. However, the flexibility also means that a lot of computation is needed to solve seemingly simple problems. Finding a neighboring simplex is one example. This problem becomes trivial for uniform voxel grids.

3.4.3 Voxel grid

Voxel grids are the most commonly used approach to represent discretely sampled volume data [7]. There are different interpretations of what voxels precisely are, one is that each voxel is a small 3D cube where the value given to the voxel is homogeneous for the whole cube. In this thesis I will use the interpretation that they are points in 3D space. This means that when casting a ray through a voxel grid, the sampling points will usually not be at the same position as the voxels. In those cases, the surrounding voxels are interpolated. The simplicity of voxel grids make them ideal for representing

volumes with high performance. Throughout the rest of this text, I will restrict my attention to using voxel grids as the reduced models.

3.5 Voxelized method

3.5.1 Voxelization

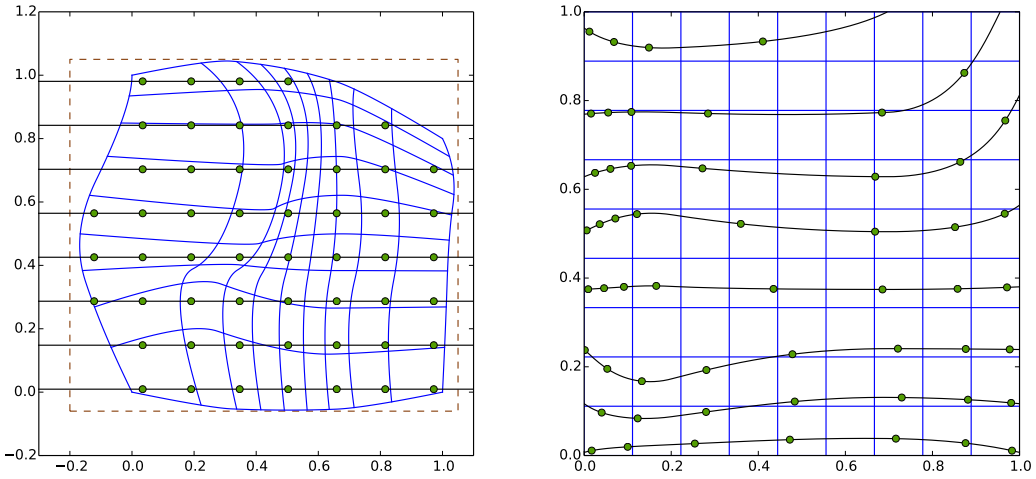
To represent isogeometric models as voxels, there must be a voxelization process. If the models are static, i.e., the geometry of the models does not change while rendering, the voxelization only needs to be precomputed once before the actual rendering. Dynamic scenes, such as simulations, must either be voxelized for every frame, or all simulation states must be voxelized before any rendering. Voxelization can be done by viewing the models in an orthogonal projection, and use equidistant sample points along each view-ray. In an orthogonal projection all the view-rays are parallel. The resulting voxel grid will ultimately be a rectilinear box for simplicity and convenience under storage. Since each volume block in an isogeometric model can have any arbitrary shape, a corresponding box-shaped voxel grid can contain many non-resident voxels, meaning voxels without a defined value. It is important that the non-resident voxels are not included when interpolating.

The distance between each sample point must also be determined, which affects how many voxels are needed to represent a scene. The larger the sampling point distance, the fewer voxels there will be, decreasing the amount of memory needed. Typically, isogeometric models have smooth geometry, which can be problematic to represent with a coarse voxel grid. As Fuchs and Hjelmervik argued in [8], to represent the scalar field accurately, a very high number of voxels may potentially be needed. This is especially true at the outline of an object, where deviations from the original model are more easily visible.

For every internal sample point g_i using the direct method, it is necessary to evaluate $\rho(\phi^{-1}(g_i))$ to determine the scalar, followed by a lookup in the transfer function. By having a simpler discretized model representation, the numerous computationally expensive spline evaluations can be replaced with hardware accelerated texture fetches. As a consequence, it is expected to see an increase in performance, while the visual accuracy is expected to be reduced.

In the voxelized model approach, the isogeometric model is represented as a discretized voxel grid. The first step when generating this is to define a bounding box B that completely encloses the geometry. Recall from section 2.1 the convex hull property of B-splines. The volume block is enclosed within the convex hull from the coefficients $\boldsymbol{c}$ of the spline ϕ. The bounding box can thus be determined by using the extrema of $\boldsymbol{c}$ along each axis.

The isogeometric model is then sampled inside the bounding box B using orthogonal projection, see figure 3.2. Since voxel grids are uniform, both the sampling rays and the sampling distances must be equidistant. The resolution of the resulting voxel grid depends on the chosen number of rays and the number of samples along



(a) The geometry domain. The dashed line shows the bounding box.

(b) The parameter domain.

Figure 3.2: Sampling the model with orthogonal projection as part of the voxelization process. Here an 8x8 texture is created. The black lines illustrate the sampling rays and each sample point is marked with a green circle.

each ray. For each sample point g_i in the volume, the parameter $p = \phi^{-1}(g_i)$ is approximated. Generating these voxel grids can be done in a pre-process step, as it has no dependency on the observer. This means that the approximation can be made highly accurate, since this has no effect on the performance while rendering. The parameters are approximated using the Newton-Raphson method similarly to the reference solution, and a very small tolerance can be used as the stopping criterion.

There are some challenges when reducing isogeometric models to voxel grids. The first challenge that is discussed is related to what to store in each voxel and subsequently how to generate the LODs. This depends on when the classification is applied.

3.5.2 Classification

Since the classification is difficult to do automatically, it is desirable to be able to change the transfer function on-the-fly while visualizing in real-time. For isogeometric volumes, the 3D data is represented as a continuous field defined by a spline ρ. A discretized representation of volume data contains discrete samples. What to store for each sample depends on the type of classification.

Pre-classification

In pre-classification, the optical properties are stored directly in the volume representation. For the case with voxelizing isogeometric models, this would mean that we would have to apply the transfer function for every sample point, and then store the resulting optical property. In other words, each texel in a 3D texture stores an RGBA value. The color for a given point in the volume is determined by interpolating the surrounding texels. If the transfer function were changed however, the original field data would not be available in the texture. That means that a new voxelized model must be created as soon as the transfer function changes. Depending on the discretization interval of the voxelization process, this could severely affect performance in real-time rendering. For every sample point ϕ^{-1} has to be approximated and an evaluation of ρ is necessary to get the field data. It is also necessary to evaluate the transfer function, but this would typically only be a 1D texture fetch and thus insignificant in comparison. In the end it is unlikely that this approach would allow the user to change the transfer function while visualizing in real-time.

In addition, this approach stores four elements (RGBA) in each texel of the texture. The precision of these values affects the total size of the whole texture. As was explained in section 2.5.2, even for modern GPUs memory may become a scarce resource.

Creating mipmaps for textures with RGBA values is a pretty straightforward process, and is done by down-sampling the original texture. A simple way of down-sampling is by using a box filter, where each new texel is computed as the average of the four nearest texels in the original texture. Other more advanced filters, such as Gaussian, Kaiser or Lanczos, will in most cases give a better result [1]. However, if the transfer function changes, the mipmap textures must also be recreated.

Post-classification

If the volume representation stores field data instead, we have post-classification. In the isogeometric model setting, we would still have to approximate ϕ^{-1} and evaluate ρ for every sample point when voxelizing, but the transfer function is not evaluated at this time. The texels in the 3D texture now contain discretized field data. The color for a given point is determined by interpolating the field data, and then apply the transfer function on the result. This means that the transfer function can be changed in real-time without having to recreate the volume representation. There is however an additional texture fetch, i.e. a lookup in the transfer function, for every sample point while rendering.

How to actually generate mipmap textures for this approach is not obvious. By uncritically interpolating the values when down-sampling, the end result may be that all the values are more or less similar. Distinctive features from the original data may be lost which can make classification difficult. How to achieve the same image quality when down-sampling scalar values is unclear [12].

3.5.3 Ray-casting

After having created the voxel grid, the ray-casting can begin. Similarly to the direct model, the ray-casting starts where the view-ray intersects with the model. This time however, the model is the voxel grid, so it is where the view-ray intersects with the boundaries of the voxel grid that is of interest. The boundaries are defined with same bounding box that was used when creating the voxel grid. Note that in the direct model there were $2n$ intersection points with the volume block per view-ray, while in the voxelized model there are always either 0 or 2. Traversing the volume is done by equation 3.1.

The actual sampling process is identical for the intersection points and the internal sample points. Based on the point g_i, the corresponding point t_i in the voxel grid texture is found by

$$t_i = (g_i - B_{min}) \oslash (B_{max} - B_{min}), \tag{3.2}$$

where B is the bounding box defined by its corners, and $\oslash$ denotes element-wise division. B_{max} is the corner of the bounding box furthest towards positive infinity along each axis, while B_{min} is the opposite. If any of the elements of t_i is outside the interval $[0, 1]$, then the sample point g_i is outside the voxel grid. Otherwise, a texture lookup is performed. Scalar values in the texture are then interpolated to generate the scalar value at t_i. The optical properties are determined by the transfer function, and these are composited from front to back. Since the sample points are not approximated and are used directly, the same sample distances from equation 3.1 are also used while compositing.

3.6 Boundary accurate method

One of the problems with a voxelized model is the representation of the boundaries of the volume block. While the isogeometric model may have smooth boundaries, the voxelized model will typically have more serrated representations, see e.g., figure 3.3. When ray-casting with the direct method, the first sampling point will be the intersection point between the view-ray and one of the boundaries of the spline geometry. However, when ray-casting with the voxelized method, the intersection between the view-ray and the bounding box will be the first sampling point. This means that for a non-trivial geometry the same sample points are not chosen for the two different approaches. In fact, it is likely that the first sample point for the voxelized method results in a lookup at non-resident texels in the texture. The location of the next sample point is determined by the sampling distance Δs_i, and the boundary of the voxelized model will therefore typically be skipped. An attempt to combat the issue with the voxel grid boundaries is the *boundary accurate* method.

The boundary accurate method does not introduce a new model, but uses a combination of the isogeometric and the voxelized models. The idea is to use the intersection points from the isogeometric model, and the internal sample points from the voxelized model. Similarly to the direct model, the ray-casting starts at the

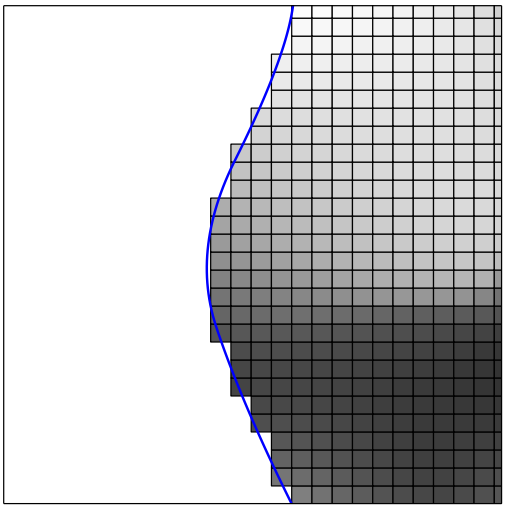


Figure 3.3: Comparison of the smooth boundary of the spline model (blue line), and the serrated boundary of the generated voxel grid.

first intersection point between the view-ray and one of the boundaries of the spline geometry. The model is then traversed between pairs of intersection points using equation 3.1.

For each intersection point, an approximated point $\tilde{g}$ in the geometry is found, along with the corresponding parameter $\tilde{p}$. Determining the scalar at the point may be done by evaluating the spline function $\rho(\tilde{p})$, or by doing a lookup in the voxel grid texture based on $\tilde{g}$. Even though the spline geometry has a boundary at $\tilde{g}$, the voxelized model may not have, as can be seen in figure 3.3. A texture lookup based on $\tilde{g}$ would then return a non-resident sample, and the previously mentioned issues with the voxelized model remain. It is therefore more appropriate to evaluate $\rho(\tilde{p})$, even though it is more computationally expensive.

The internal points in the model is sampled similarly to the voxelized method. Note that even though these sample points are guaranteed to be inside the isogeometric model, they may be outside of the voxelized model. Lookups in the voxel grid texture will then return non-resident samples. Using the direct method for these sample points will most likely generate a more visually accurate result, but at the cost of computationally expensive spline evaluations. These sample points will typically not be numerous however, since they only appear near the boundaries of the model. Another option is therefore to just ignore sampling at these points.

Since the intersections between the view-ray and the spline geometry must be found for each ray, it is expected that the voxelized method will perform better than the boundary accurate method. In the implementation presented by Fuchs and Hjelmervik [8], the hardware tesselator in modern GPUs are used to find the intersections. In an example shown in their paper, a total of 0.86 ms is spent finding intersections for each frame on a 640x480 viewport using an NVIDIA Titan GPU. Comparatively, between

5-20 µs is spent for each sample point in their implementation of the voxelized method.

3.7 Hybrid method based on a geometric criterion

By avoiding spline evaluations while rendering, the voxelized method was shown to be performing better than the direct method in [8]. The performance comes at the cost of loss of visual accuracy however. In the case where a volume block is far into the scene, it would only project onto a small amount of pixels on the screen. The result from using the voxelized method may be sufficient in this scenario to achieve the desired level of visual accuracy. At the same time there may be volume blocks close to the observer where the results from the voxelized method are too inaccurate, and it would be desirable to use the direct method.

This introduces the notion of a *hybrid* model, where the direct and the voxelized methods are combined. The basic idea is to use the voxelized method whenever it generates results that are accurate enough, and the direct method otherwise. Which method to use is specified by some sort of criteria. While ray-casting, the criteria is checked to see if the current sample point can use the voxelized method. This means however that checking the criteria cannot be too computationally expensive, as it would mean losing the main advantage of using the voxelized method.

One of the problems with the voxelized method is that if the texture size is not sufficiently large enough, then there is no guarantee that the visualization is pixel accurate. The data points in the voxel grid are highly accurate, given that a small ϵ_{Sample} was used under voxelization. Typically, a sample point g_i will not be exactly at the point where a voxel is defined. The underlying functions are then reconstructed with linear interpolation. If the geometry or the field of the isogeometric model is non-linear, this interpolated value may not be the same as the result from evaluating $\rho(\phi^{-1}(g_i))$ using the direct method. However, as long as the splines ϕ and ρ are continuous functions, the interpolated value can be replicated by evaluating $\rho(\phi^{-1}(\iota))$, where ι is an unknown point in the spline geometry. To achieve pixel accuracy with the voxelized method, there must therefore be a guarantee that ι is within the frustum of the current pixel.

Each voxel in the voxel grid is defined for a given point in the geometry. This point will be called the voxel point here. For a given sample point g_i, the four closest voxel points are denoted $(\nu_{i,k})_{k=0}^{7}$.

The interpolated scalar at g_i will always be bound by the lowest and the highest of the scalars defined at the eight voxel points closest to g_i in the geometry. Assuming that the spline functions are continuous, the point ι must be within the bounding box defined by $(\nu_{i,k})_{k=0}^{7}$. If the bounding box is ensured to be within the pixel frustum, then pixel accuracy has been guaranteed.

It is important to note that the horizontal distance between voxels can be different to the vertical distance. In the worst case, the direction of a line connecting two diagonally adjacent voxels runs perpendicular to one of the extents of the pixel frustum. The sample point may be at one of the corners of its surrounding voxel point bounding box, and ι can be anywhere inside the bounding box. The frustum must be sufficiently

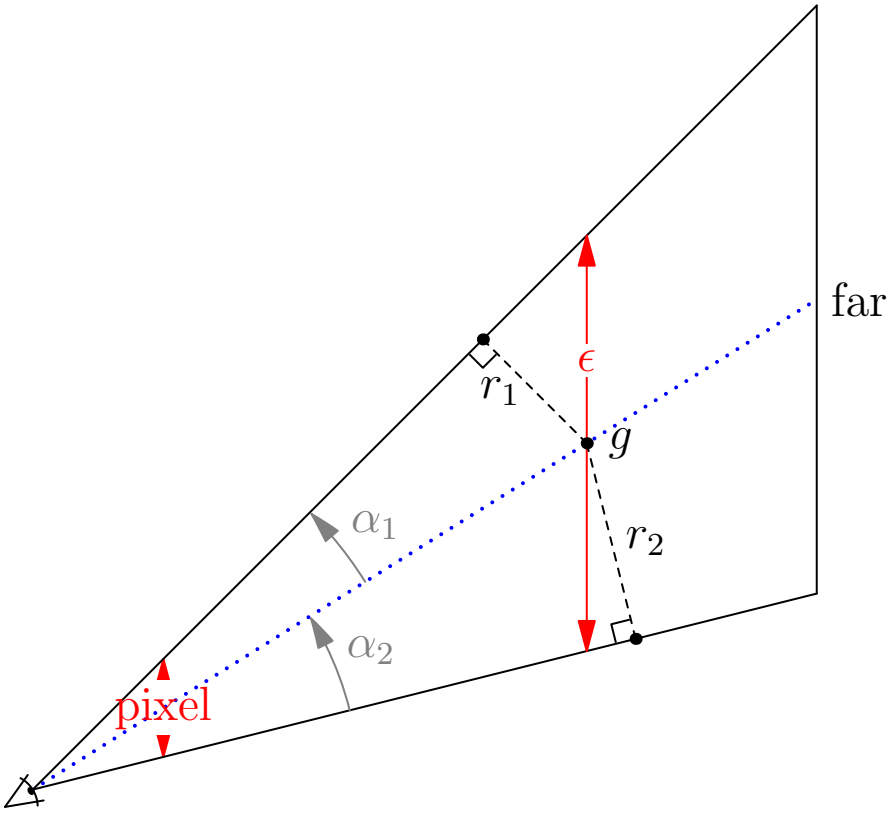


Figure 3.4: Illustration of a pixel frustum in 2D. The view-ray, marked as a blue dotted line, starts at the eye, goes through the middle of a pixel, and continues into the scene. The sample point g is somewhere inside the isogeometric model, and ϵ denotes the projected size of the current pixel at this point. The view-ray separates the frustum into an upper and a lower part, which can be of different sizes. To achieve pixel accuracy, the distance between two diagonally adjacent voxels must be lower or equal to the minimum radius, which in this illustration is r_1.

wide so that the whole bounding box the sample point is a part of is enclosed within the frustum. This means that to achieve pixel accuracy, the shortest distance between the sample point to one of the extents of the pixel frustum must be at least the distance V_d between diagonally adjacent voxels.

The pixel frustum is not always symmetrical along the view-ray, so the pixel frustum is here split into an upper and a lower frustum, see figure 3.4. Both of the two pixel frustum radiuses, r_1 and r_2, must be larger or equal to V_d to ensure pixel accuracy. The figure shows a frustum with a wide angle for illustration purposes, while in practice the angles are very narrow, meaning that $r_1 \approx r_2$. In addition, we know that ϵ is slightly larger than $r_1 + r_2$. To simplify, there is pixel accuracy when $0.5\epsilon > V_d$, which will be the geometric criterion.

Determining if the voxelized method can be used depends on the radius of the

pixel frustum at the sample point. A hybrid method based on a geometric criterion is basically a form of view-dependent level-of-detail based on the distance from the observer. As soon as one of the sample points along a ray can use the voxelized method, then all subsequent sample points can use the voxelized method as well.

The hybrid model can also be further extended to use a level-of-detail hierarchy. With the direct model being the most accurate, it defines level 0 in the hierarchy. Level 1 is defined by a high resolution voxel grid. The subsequent levels are then defined by mipmaps of the original voxel grid with progressively lower resolution.

Chapter 4

Comparison of methods in 2D

4.1 2D prototype

Fuchs and Hjelmervik [8] developed a framework for visualizing isogeometric volumes in OpenGL. This framework was available during the project, but early on it was decided to start developing a 2D prototype. The main purpose for doing this was to have a platform for testing new methods and different approaches during the project. Although the 3D implementation could be used for this directly, implementing different approaches would be simpler while being limited to only two dimensions, and it would also be easier to visualize different aspects of the process. OpenGL code is also difficult to debug compared to programs that run exclusively on the CPU. This meant that a volume visualization framework had to be implemented from scratch. It was estimated that the time implementing the 2D prototype would be less than the extra time spent developing directly on the 3D implementation. Developing the framework also turned out to be quite helpful to get a better understanding of the problem domain.

4.1.1 Comparison to 3D

In 2D, both the parameter domain P and the geometry domain G are in $\mathbb{R}^2$, while the spline ϕ is a mapping $\mathbb{R}^2 \to \mathbb{R}^2$, and ρ is a mapping $\mathbb{R}^2 \to \mathbb{R}$. The parameter domain is limited to a rectangle, where each of the sides maps to lines that bounds the geometry. An isogeometric model in this setting is therefore not a volume, but can be thought of as an area in a plane that result from slicing the volume where one of the parameters is constant. The view-rays are also limited to the same plane.

Similar to equation 2.4, a surface can be represented by B-splines as

$$\boldsymbol{f}(u,v) = \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} B_{i_1,p_1}(u) B_{i_2,p_2}(v) \boldsymbol{c}_{i_1,i_2}, \tag{4.1}$$

where B_{i_1,p_1} and B_{i_2,p_2} are the B-spline basis functions of degree p_1 and p_2 respectively. The B-splines are defined with the underlying knot vectors $\boldsymbol{\tau} = \{\tau_1, \tau_2, \ldots, \tau_{n_1+p_1+1}\}$ and $\boldsymbol{\sigma} = \{\sigma_1, \sigma_2, \ldots, \sigma_{n_2+p_2+1}\}$.

Similar to the approach by Fuchs and Hjelmervik, the 2D prototype also implement visualization with ray-casting. In 2D, a surface has four boundary lines which the view-rays can intersect.

4.1.2 Limitations

While the 3D implementation utilizes the GPU as a co-processor, the 2D prototype runs entirely on the CPU. How a 3D GPU implementation performs is not only related to what method is used, but also how efficiently the hardware is used, such as the rate of occupancy on the GPU. The 2D prototype can therefore not easily reflect how the methods will perform on the GPU. However, most likely the spline evaluations and approximating the inverse of the geometric spline ϕ are the bottlenecks in the current 3D implementation. Methods that reduce the amount of spline evaluations would therefore be good candidates for improved performance.

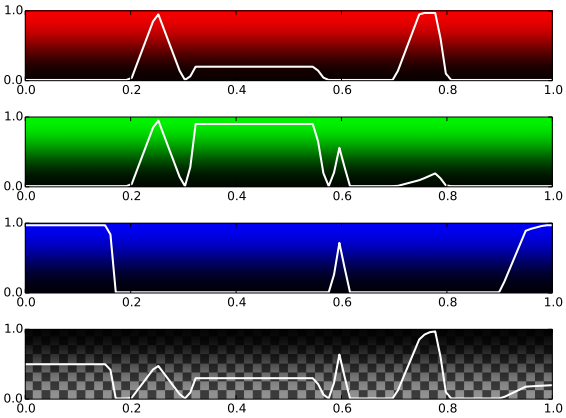
For simplicity, the prototype has not been designed to be able to ray-cast any type of model. The purpose of the prototype is to develop and test new methods for ray-casting isogeometric models. It is assumed that a scene only contains one isogeometric model, consisting of only one volume block. Secondly, it is assumed that the geometry of the models used are such that every ray will either intersect with the geometry exactly twice, or not at all. Of course, in practice most models have a geometry that is more complex than this. However, solving the ray-cast problem for models with more intersections is equivalent to solving the ray-cast problem between two intersections multiple times. New methods can therefore be tested for the general case by solving the ray-cast problem between only two intersections for these new methods.

In section 2.4 different approaches for approximating ϕ^{-1} were described. For simplicity, only the Newton-Raphson method has been used in the prototype. The Newton-Raphson method has also been used to find the intersection points, as previously demonstrated by Martin and Cohen [17].

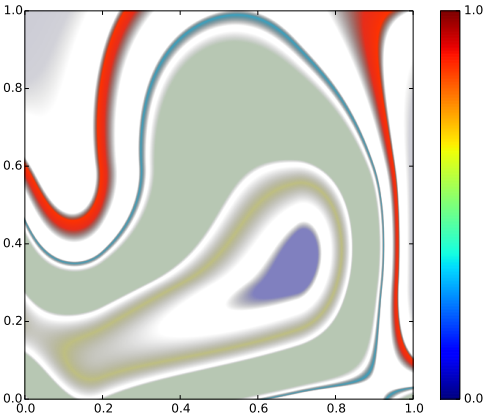
4.1.3 Setup

In the examples shown here, a static 1D transfer function has been used. The transfer function has been designed to have many different properties, including being linear in some intervals while having peaks in others. Most input to the transfer function also generates output with high transparency, while a small interval generates opaque colors. Figure 4.1a is a visualization of the transfer function where it is shown the value of the RGBA components based on the scalar value. The result after applying the transfer function to the scalar field from figure 3.1b is shown in figure 4.1b.

All the rays start at the location of an imaginary observer. The point $(-1.2, 0.65)$ in the geometry has been used in all the results shown here. Each ray then passes through the middle of a pixel on the screen, before it goes into the scene. The screen has been defined as the line segment from $(-0.5, 0.2)$ to $(-0.5, 0.9)$ in the geometry, containing 100 pixels. This setup gives insight about the different methods, while more varied setups are later used in the 3D implementation.



(a) The 1D transfer function. From top to bottom is the Red, Green, Blue and Alpha channels of the output. The vertical axes show the intensity of each channel, while the input scalar varies along the horizontal axes.



(b) Applying the transfer function to the scalar field shown in figure 3.1b.

Figure 4.1: Visualization of the transfer function from dataset 1.

4.1.4 Software and tools

In this section a brief explanation will be made for the choice of software and tools used to make the prototype.

Python

The prototype has been implemented using the Python[1] programming language. Python is a high-level language, which makes it ideal for prototyping, as one does not have to account for small details when programming, and can focus more on solving problems in the prototype domain. Even though it may not produce programs with the same level of performance as programs written in C/C++, at this stage it was more important to develop methods and test the visual accuracy of these than testing the actual performance. One of the main reasons for choosing Python, was the range of useful libraries available to it.

Python libraries

The plots shown in this thesis have been plotted by using the Matplotlib[2] library. Matplotlib is an easy to use plotting library which offers both an object-oriented approach, as well as an interface that resembles MATLAB.

For scientific computing, mathematical modules have been used from the NumPy[3]

[1] https://www.python.org/
[2] http://matplotlib.org/
[3] http://www.numpy.org/

and the SciPy[4] libraries. SciPy offers ready to use spline evaluation, (bi-)linear interpolation among others.

To compare the different results, the CIEDE2000 implementation in the color module from scikit-image[5] have also been used.

4.2 Implementation of ray-casting methods

4.2.1 Direct method

As explained in section 2.4, the first step while ray-casting is to locate where the view-ray γ intersects with the boundaries of the geometry defined by ϕ. In 2D, this process means dealing with the intersections on each of the four bounding lines of the model. An intersection between the view-ray γ and a boundary S is given by

$$S(x) - \gamma(t) = 0, \tag{4.2}$$

where $S(x) \in \{\phi(0,x), \phi(1,x), \phi(x,0), \phi(x,1)\}$. Equation 4.2 is approximated for x and t using the Newton-Raphson method. Each iteration gives the approximations $\tilde{x}$ and $\tilde{t}$, and the Boolean test

$$S(\tilde{x}) - \gamma(\tilde{t}) < \epsilon_{Intersect} \tag{4.3}$$

is used as a stopping criterion where $\epsilon_{Intersect}$ is a predetermined tolerance. The Newton-Raphson method has quadratic convergence, and if the test fails for 20 consecutive iterations it is assumed that the view-ray does not intersect with the geometry.

When ray-casting the direct model in the prototype, an ellipse is used for each point to ensure pixel accuracy. The size of the ellipse is determined by the radius of the pixel frustum at a given sample point and the sampling distance. This means that an ellipse for a given sample point will not overlap with any other similar ellipses. When a candidate parameter $\tilde{p}_i$ is found and its corresponding point $\tilde{g}_i$ in the geometry, a test is performed if $\tilde{g}_i$ is enclosed by the ellipse. If the test fails, then $\tilde{p}_i$ is used as the guess for the next iteration. Ellipses were chosen because it is easy to test if a point is inside or not. By having all sample points inside their corresponding ellipses along a ray both parametric accuracy as well as covering accuracy have been guaranteed. This approach does however set a stricter requirement for the sample points than just pixel accuracy.

When generating the reference solution, better approximations can be made. Instead of using pixel accuracy as the stopping criterion for the Newton-Raphson method, a small maximum tolerance is used. In this case the equation

$$\phi(p_i) - g_i = \mathbf{0} \tag{4.4}$$

is approximated for p_i, where g_i is the ideal sample point. The Newton-Raphson method is repeated until the Boolean test

$$\|\phi(\tilde{p}_i) - g_i\| < \epsilon_{Sample} \tag{4.5}$$

[4] http://www.scipy.org/scipylib/index.html
[5] http://scikit-image.org/

(a)

(b)

(c)

Color	Description
	No noticeable color difference
	Acceptable color difference
	Noticeable color difference between most color pairs
	Clear color difference between color pairs

(d) Color codes used for CIEDE2000 color difference.

Figure 4.2: Ray-casting with the direct method. The reference solution shown in (a) compared to the results from ray-casting with the pixel accurate direct method shown in (b). The CIEDE2000 color difference is visualized in (c).

holds. In addition, the sampling distance in equation 3.1 is made smaller for the reference solution to achieve a better approximation of the volume rendering integral. The reference solution used throughout this chapter have been generated with $\epsilon_{Intersect} = \epsilon_{Sample} = 10^{-5}$ and a constant sampling distance $\Delta s = 10^{-5}$.

For each approximated sample point $\tilde{g}_i$, the scalar s is determined by $s = \rho(\tilde{p}_i)$ and the optical properties by $TF(s)$, where TF is the transfer function. The color at each sample point contributes to the final color of the pixel by front-to-back compositing using equation 2.8. Since typically the approximated sample point $\tilde{g}_i \neq g_i$, then Δs_i from equation 3.1 does not reflect the actual sample distance. While compositing the sample distance $\Delta\tilde{s}_i = \|\tilde{g}_i - \tilde{g}_{i-1}\|$ is used instead. The ray-casting ends when the resulting color has been saturated (alpha value at 1.0), or at the second intersection point g_{Out}, whichever comes first.

The result from ray-casting the model from dataset 1 using the direct method can be seen in figure 4.2. In the top the colors of the 100 pixels from the reference solution are shown horizontally. Below this is the result from ray-casting the pixel accurate direct method, where $\epsilon_{Intersect} = 10^{-3}$ and $\Delta s = 10^{-2}$. In the bottom a visualization of the color difference is shown. To generate this the CIEDE2000 algorithm has been utilized, which was described in more detail in section 3.2.2. The algorithm calculates scalars representing the color differences between the pixels from the reference solution, to the pixels from the pixel accurate direct method. Each scalar has then been color coded according to table 4.2d. As expected, the pixel accurate result deviates slightly from the reference solution due to having less accurate sample points and a longer sample distance.

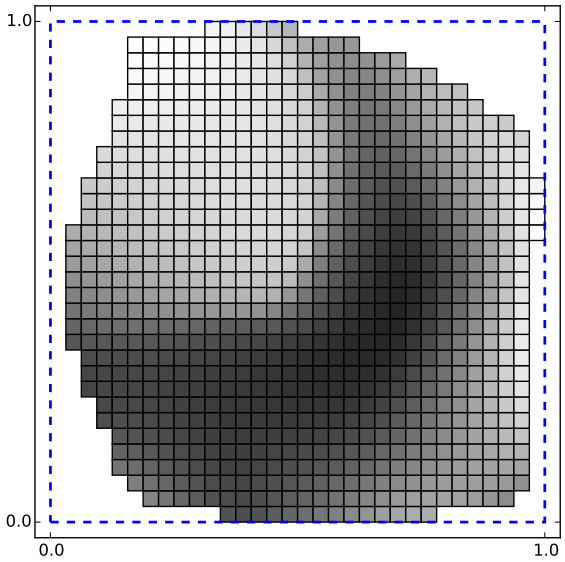


Figure 4.3: A generated texture of size 32x32. Each texel is shown as a square with a greyscale color depending on the scalar value. Notice that the texture coordinates ranges from 0 to 1 inside the bounding box, shown with a blue dashed line.

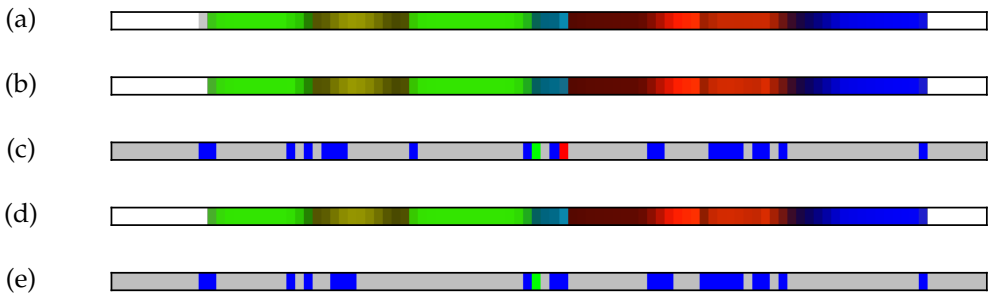
4.2.2 Voxelized method

In 3D, the discretization process is also called voxelization. Even though volume and voxel is not the correct terms for a 2D setting, they will be used here to more easily compare the 2D prototype to a 3D implementation.

Under the voxelization process, the isogeometric model is sampled inside the bounding box, as was explained in section 3.5.1. For each sample point, equation 4.5 is used to approximate the parameter p, with a tolerance $\epsilon_{Sample} = 10^{-5}$. The scalar $s = \rho(p)$ is then stored in the voxel grid. Samples outside the volume are given the invalid scalar value -1. In addition to the scalar values, the texture also stores indicators. For texels with a valid scalar value the indicator is set to 0, while for non-resident texels the indicator is set to 1.

The result after sampling is a matrix of scalars. These discrete samples form a basis for approximate reconstruction of the smooth isogeometric model. Bilinear interpolation is used as the reconstruction filter of the scalar field. Ideally the methods created in the prototype should be easy transferable to a 3D implementation. A texture is a common way of storing a voxel grid in OpenGL applications, so a simple texture is mimicked in the prototype. Doing a texture lookup in OpenGL usually involves hardware accelerated interpolation as well. SciPy's `interp2d` class in the `interpolate` sub-package have been used in the prototype for this purpose. The texture coordinates are limited to the interval $[0, 1]$, similarly to normal use in OpenGL.

The voxel grid is rectilinear, but the isogeometric model can have any arbitrary shape. This means that there will typically be many non-resident texels in the generated



Color	Description
	No noticeable color difference
	Acceptable color difference
	Noticeable color difference between most color pairs
	Clear color difference between color pairs

(f) Color codes used for CIEDE2000 color difference.

Figure 4.4: Ray-casting with the voxelized method. The reference solution shown in (a) compared to the results from ray-casting with the voxelized method. A texture of size 192x192 has been used to generate the result in (b), and (c) shows the CIEDE2000 color difference. Figure (d) and (e) similarly shows the results from using a texture of size 256x256.

voxel grid. When doing a lookup near the edges of the discretized model, there may be fewer than four resident texels available to do bilinear interpolation. It is important that none of the non-resident texels are involved when interpolating, as this will introduce artifacts along the edges. When the texture is sampled the indicator is also bilinearly interpolated, and if the interpolated indicator differs from 0, then a non-resident texel is involved in interpolation and the sample is discarded.

A problem emerges with this simple texture imitation when doing a lookup at a coordinate near the edges of the texture. To do bilinear interpolation, the four nearest texels surrounding a given point are used, but on the edges there are only one or two such texels. In OpenGL this is solved by a texture wrap parameter. This allows interpolation at the edge of a texture by defining how the lookup behaves outside the interval $[0, 1]$. A common approach is clamping to edge, where the outermost texels extend virtually infinitely out from the texture. In this case the values at the outermost texels are used at the texture edges, which means that the scalar samples at the edges can be worse approximations than at other points in the texture. However, the voxel grid can approximate the geometry closer than if the points at the edges were ignored. To get the clamping to edge behavior while using `interp2d`, the prototype use *ghost cells*. These are basically texels that create a border around the original texels. The value of each texel is the same as the texel next to it towards the center of the texture.

Figure 4.4 shows results from ray-casting with the voxelized method. As expected,

(a)

(b)

(c)

Color	Description
	No noticeable color difference
	Acceptable color difference
	Noticeable color difference between most color pairs
	Clear color difference between color pairs

(d) Color codes used for CIEDE2000 color difference.

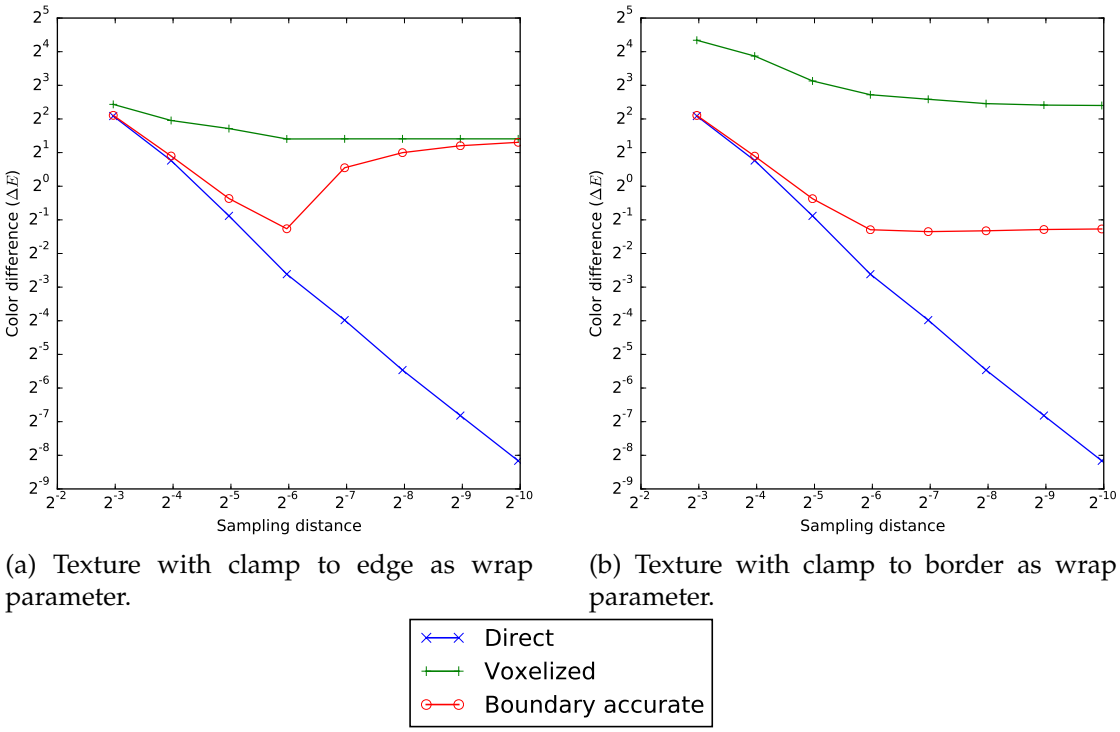
Figure 4.5: Ray-casting with the boundary accurate method. The reference solution shown in (a) compared to the results from ray-casting with the boundary accurate method. A texture of size 192x192 has been used to generate the result in (b), and (c) shows the CIEDE2000 color difference.

the color differences are greater in average than compared to the direct method. However, since there are no spline evaluations while ray-casting, it is also expected that this method will perform better. By increasing the size of the texture, the voxelized method generates a result that more closely resembles the reference solution. This suggests that a desired level of accuracy can be achieved by using appropriately sized textures. However, depending on the isogeometric model, the size of the texture might have to be very high to be able to accurately reconstruct the scalar field [8]. Hardware restrictions will typically limit the maximum texture size available.

4.2.3 Boundary accurate method

The boundary accurate method uses both the spline model and the voxelized model while ray-casting. Sampling at the intersection points will always use the spline model. Sampling at subsequent points normally use the voxel grid, although it is possible that sampling at such a point would result in a non-resident sample. In the prototype these points are ignored while sampling. When compositing it is therefore important to notice that the sampling distances may differ from Δs_i used in equation 3.1. An example is if there is sampling at the points g_i and g_{i+2}, but sampling at g_{i+1} resulted in non-resident texels and is therefore ignored. The sampling distance for g_i will then be $\Delta s_i + \Delta s_{i+1}$.

In the voxelized method, a clamp to edge approach was used for solving the problem of sampling at the edges of a texture. The argument was that even though the approximation of the scalar field could be inaccurate at these points, that the approximation of the geometry is typically better. For the boundary accurate method, the geometry is approximated as good as in the direct method. The disadvantage of



(a) Texture with clamp to edge as wrap parameter.

(b) Texture with clamp to border as wrap parameter.

Figure 4.6: Comparison of mean color difference when clamping to edge versus clamping to border. A texture of size 32^2 has been used for all sampling distances. The results have been generated from ray-casting an isogeometric model with trivial geometry, meaning that $\phi(u, v) = (u, v)$. The scalar field is the same as was shown in figure 3.1b, and the transfer function is the same as was shown in figure 4.1.

using the clamp to edge approach can therefore outweigh the advantage when using the boundary accurate method, see e.g., figure 4.6a. When the sampling distances get small enough, more sample points lay at the edges of the texture. While the overall results for the voxelized method is better with a clamp to edge approach, the results get worse for small sampling distances with the boundary accurate method. An alternative approach is so-called clamp to border, where edge samples are interpolated with a fixed border value. By marking the border as non-resident, the less accurate approximated samples at the edges of the texture are ignored. The mean color differences for this approach can be seen in figure 4.6b, and for this example the results are clearly better for small sampling distances. The graph shows that the mean for the boundary accurate method does not really change after hitting the sample distance of 2^{-6}. This is because points inside the texture are linearly interpolated, and having a sample distance lower than the distance between voxels does not make the result more accurate.

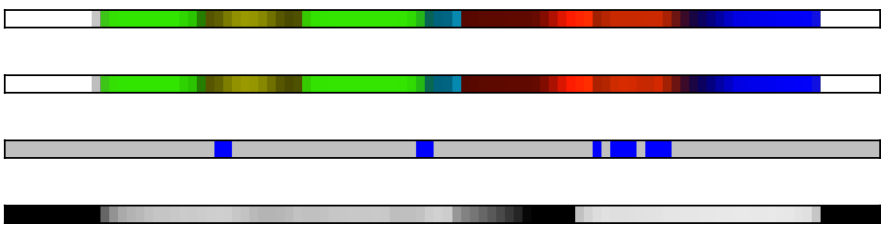
In figure 4.5 the results from visualizing the model with the boundary accurate method is shown. The average color difference is noticeably better than for the voxelized method, even though the performance is expected to be almost similar. Since

(a)

(b)

(c)

(d)



Color	Description
	No noticeable color difference
	Acceptable color difference
	Noticeable color difference between most color pairs
	Clear color difference between color pairs

(e) Color codes used for CIEDE2000 color difference.

Figure 4.7: Ray-casting with a hybrid of the direct method and the boundary accurate method using a 192x192 texture. Figure (a) shows the reference solution, the hybrid solution in (b), and the CIEDE2000 color difference in (c). The ratio between sample points from the spline model and sample points from the voxel grid is shown in (d), where the lighter the color is, the more samples have been taken from the voxel grid.

the internal sample points are considered less accurate than the intersection points, the visual accuracy is likely to be affected the more internal sample points there are.

4.2.4 Hybrid

The hybrid method switches between the spline model and the voxelized model based on a geometric criterion. In section 3.7, it was explained that the geometric criterion depends on the size of the smallest of the radiuses of the pixel frustum at a given sample point. Although the exact size of the radiuses can be found in the 2D prototype, we want to compare against the projected pixel size ϵ because it will later be useful for the 3D implementation. Finding ϵ in the 2D prototype is trivial with trigonometric functions and will not be covered here.

The hybrid method can also combine the direct method with the boundary accurate method. The results from visualizing with this type of hybrid can be seen in figure 4.7. The ratio between the use of the spline model and the voxelized model has also been visualized. As expected the color differences reaches a middle ground between the voxelized method and the direct method.

4.3 Comparing the methods

The ray-casting methods generates results with varying degrees of visual accuracy, depending on how many samples are taken from the spline model compared to the

voxel grid, which is highly likely to affect the performance while rendering. Since the geometry, defined with the function ϕ, and the scalar field, defined with the function ρ, are independent of each other, they can be combined from different datasets to create interesting cases. Dataset 0 includes the trivial geometry defined as $\phi_0(u, v) = (u, v)$, and the trivial scalar field is defined as $\rho_0(u, v) = v$. Recall that to determine (u, v) based on a point in the geometry, it is necessary to evaluate the inverse of ϕ. When ϕ is a spline function, the inverse must be approximated. Note that in the cases with a trivial geometry, ϕ^{-1} can be evaluated exactly. For dataset 1 both ϕ_1 and ρ_1 are spline functions. Visualization of the geometry defined by ϕ_1 can be seen in figure 3.1a, and of the scalar field defined by ρ_1 in figure 3.1b. In all the cases the static transfer function visualized in figure 4.1a has been used.

The setup for the cases presented here is similar as before, with the view-rays starting from an imaginary observer located at $(-1.2, 0.65)$ in the geometry. Ray-casting has been performed on a vertical screen, defined as a line segment from $(-0.5, 0.2)$ to $(-0.5, 0.9)$ in the geometry containing 100 pixels. The result from visualizing is a 1D array of RGB colors, and the CIEDE2000 algorithm has been used to calculate the color difference against a reference solution. The reference solution in all the different cases have been generated by visualizing the model with the direct method, with a sampling distance of 10^{-5} along the view-rays, and a tolerance of 10^{-5} for approximating ϕ^{-1}.

In the methods that use a voxel grid, bilinear interpolation is used to determine the value at a given point in the texture. Since the reconstruction is linear, having multiple sample points within the size of a voxel does not yield a more accurate result. The sampling distance can therefore be made greater for coarser textures. In the results shown here, the sampling distance have been set to a constant half of the distance between two adjacent voxels. To allow for a fair comparison, the same constant sampling distances are used for all different methods, including the direct method.

The visual accuracies of the different methods are then compared. The CIEDE2000 algorithm returns a color difference as a scalar between pairs of colors. For each method there is therefore generated 100 scalars representing the color difference. To compare the different methods, the maximum, the mean and the variance of all pixel's scalar values are calculated. The results are shown in tables here, rounded to two decimal places. The mean and maximum of the color differences are also illustrated in graphs. Note however that the axes are logarithmic. The graphs have been color coded depending on the method used to generate the data. The data points are shown with markers, with separate markers for each method. Note also that the data points are linearly interpolated in the graphs, so the points between the data points are therefore not accurately represented.

When the results for a specific method are good enough was specified in section 3.2.2. To summarize, the maximum color difference should not exceed 5.0, and the mean color difference should not exceed 1.0. Below is a brief overview of the cases presented in this chapter which have been used to generate results.

- *Case A: Trivial geometry, trivial scalar field*. In this case the isogeometric model is defined by trivial functions. The voxel grid can therefore represent the

isogeometric model accurately. This serves as a control case where it is expected that the different methods generate similar results.

- *Case B: Trivial geometry, non-trivial scalar field*. While using the same geometry as in case A, the scalar field is for this case defined by a spline function. A voxelized version can represent the geometry exactly, but not the scalar field. This case therefore shows how the different methods compare when the scalar field is reconstructed approximately.

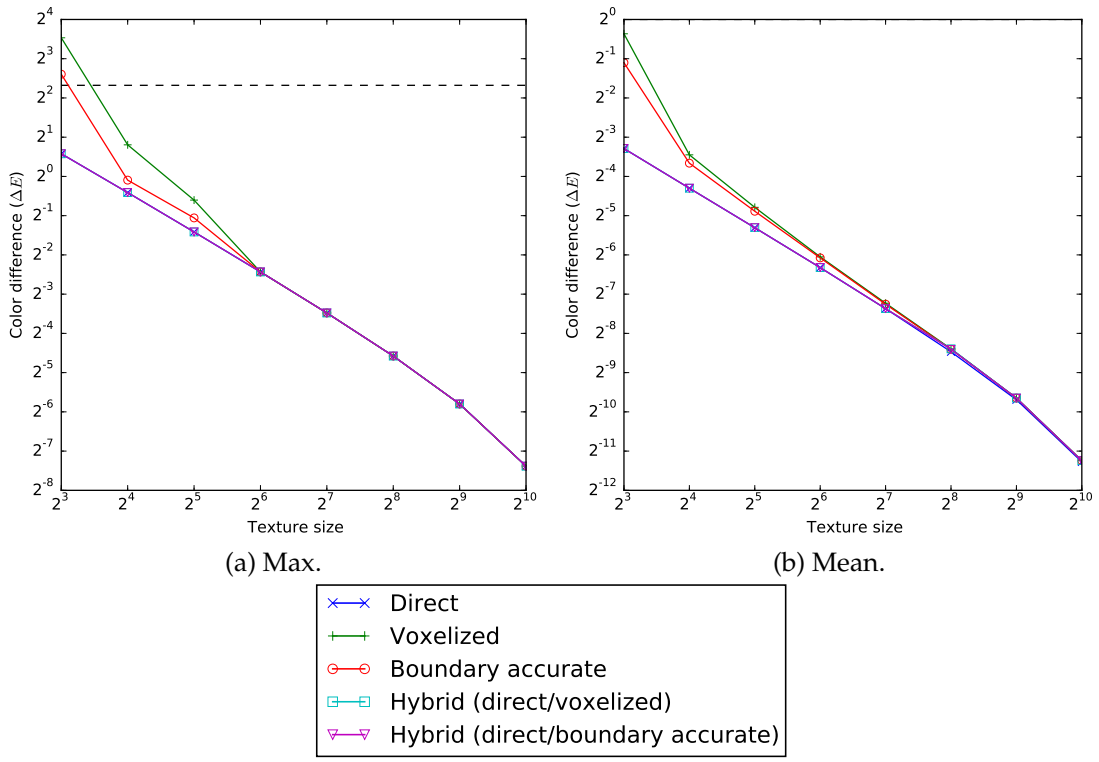
- *Case C: Non-trivial geometry, trivial scalar field*. The geometry for the isogeometric model in this case is defined by a spline, while the scalar field is kept trivial. While an isogeometric model typically has smooth boundaries, a voxelized representation is restricted to a grid. By having a trivial scalar field which can be reconstructed, the problem of approximating the geometry is isolated.

- *Case D: Non-trivial geometry, non-trivial scalar field*. This is the most complete example, as it has an isogeometric model defined by spline functions. The results from this case is the most interesting as it generated from a more practical example.

4.4 Case A: Trivial geometry, trivial scalar field

In this case the volume block is defined with the trivial functions ϕ_0 and ρ_0. The results can be seen in table 4.1, and an illustration of these in figure 4.8. Since the geometry is rectilinear and axis-aligned, the voxel grid represents the boundaries of the geometry exactly. The trivial scalar function is also linear, which means that it can be reconstructed exactly using linear interpolation. As expected, the visual accuracy results show that in this case there are no significant difference between using the different methods. The solutions do differ from the reference solution however because of the transfer function and the way the volume rendering integral is approximated. Recall from section 2.3 the algorithm for front-to-back compositing. In the prototype it is assumed that the color for a segment between the sample points g_i and g_{i+1} only depends on the color at sample point g_i and the sampling distance between the points. This leads to a step function approximation of the volume rendering integral. The lower the sampling distance gets, the closer the results get to the reference solution.

The visual accuracy of all methods converge towards the reference solution approximately by linear decay. The maximum and mean thresholds are shown in their respective graphs as dashed horizontal lines. In the mean graph, the threshold is at the very top of the graph window. For all but the lowest texture size, the requirements for visual accuracy are met for all methods.



(a) Max. (b) Mean.

Figure 4.8: Illustration of the visual accuracy for Case A: Trivial geometry, trivial scalar field. The dataset is ray-casted with the different methods using different sample distances. These distances are dependent on the size of the textures. Along the horizontal axes, the numbers show one dimension of the quadratic texture sizes used. The resulting pixel colors are then compared to a reference solution with the CIEDE2000 algorithm to find the color differences. Subsequently the maximum and the mean of these are found to generate the data points shown in the graphs here. The vertical axes show the ΔE color difference value.

(a) Direct method.

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
18	1.49	0.10	0.06
35	0.75	0.05	0.02
68	0.38	0.03	0.00
135	0.19	0.01	0.00
268	0.09	0.01	0.00
534	0.04	0.00	0.00
1067	0.02	0.00	0.00
2133	0.01	0.00	0.00

(b) Voxelized method.

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	18	11.58	0.78	4.88
16^2	35	1.75	0.09	0.06
32^2	68	0.66	0.04	0.01
64^2	135	0.19	0.02	0.00
128^2	268	0.09	0.01	0.00
256^2	534	0.04	0.00	0.00
512^2	1067	0.02	0.00	0.00
1024^2	2133	0.01	0.00	0.00

(c) Boundary accurate method.

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	18	6.08	0.47	1.19
16^2	35	0.94	0.08	0.03
32^2	68	0.48	0.03	0.01
64^2	135	0.19	0.01	0.00
128^2	268	0.09	0.01	0.00
256^2	534	0.04	0.00	0.00
512^2	1067	0.02	0.00	0.00
1024^2	2133	0.01	0.00	0.00

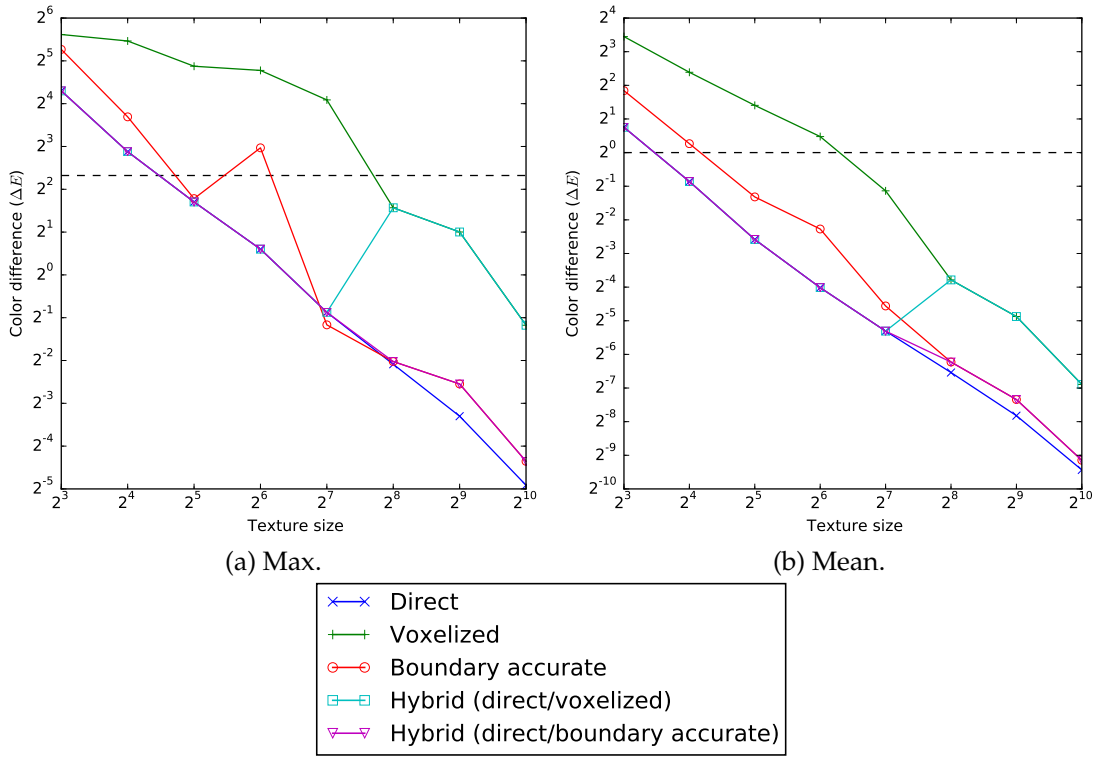
(d) Hybrid method (direct/voxelized).

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	18	1.49	0.10	0.06
16^2	35	0.75	0.05	0.02
32^2	68	0.38	0.03	0.00
64^2	135	0.19	0.01	0.00
128^2	268	0.09	0.01	0.00
256^2	534	0.04	0.00	0.00
512^2	1067	0.02	0.00	0.00
1024^2	2133	0.01	0.00	0.00

(e) Hybrid method (direct/boundary accurate).

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	18	1.49	0.10	0.06
16^2	35	0.75	0.05	0.02
32^2	68	0.38	0.03	0.00
64^2	135	0.19	0.01	0.00
128^2	268	0.09	0.01	0.00
256^2	534	0.04	0.00	0.00
512^2	1067	0.02	0.00	0.00
1024^2	2133	0.01	0.00	0.00

Table 4.1: Statistics for Case A: Trivial geometry, trivial scalar field. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. For the methods that use a voxel grid, the texture sizes are shown under #V.



(a) Max. (b) Mean.

Figure 4.9: Illustration of the visual accuracy for Case B: Trivial geometry, non-trivial scalar field, similarly to figure 4.8.

4.5 Case B: Trivial geometry, non-trivial scalar field

Similarly to case A, the geometry is also in this case defined with ϕ_0 and the boundaries can be represented exactly with a voxel grid. In this case however, the scalar field is defined with the non-linear function ρ_1. This means that since bilinear interpolation between discrete sample points is used to reconstruct the scalar field, the reconstruction will only be an approximation. The more sample points there are, the better the approximation will be. It is therefore expected that the results from methods based on a voxel grid gets closer to the reference solution the higher the resolution of the voxel grid used is. The results for this case can be seen in table 4.2, and an illustration in figure 4.9.

The graphs show that the results from the boundary accurate method fluctuates compared to the direct method. It is worth noting that approximations must be made for all methods. The inverse of the geometry function is approximated differently for the direct method compared to the reference solution, and the other methods use a different approximation for the geometry altogether. The methods based on a voxelized representation are also dependent on an approximate reconstruction of the scalar field. Since the boundary accurate method combines these models, the approximations can

in some cases be better by chance, which can explain the fluctuations in the results.

An interesting detail in the graphs is where the switchover happens for the hybrid methods. For the initial texture sizes, the geometric criterion is never satisfied because the distances between the voxels are too big. As soon as the distances get small enough, the geometric criterion is satisfied for the samples furthest from the observer. The distance between two diagonally adjacent voxels seems to be small enough to meet the geometric criterion when the texture size is somewhere between 128^2 and 256^2. The graphs show that the visual accuracy gets worse for both the hybrid methods compared to the direct method after the switchover. This is as expected since the direct method is the most accurate. However, the question is if the visual accuracy is good enough after the switchover. For the hybrids the graphs show that both methods satisfy having a mean below 1.0, and max below 5.0 for all the chosen texture sizes. However, while the boundary accurate hybrid method consequently gives better results when the texture size is doubled, this is not the case for the voxelized hybrid method.

(a) Direct method.

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
18	19.68	1.69	6.93
35	7.35	0.55	1.01
68	3.26	0.17	0.14
135	1.52	0.06	0.03
268	0.55	0.03	0.00
534	0.24	0.01	0.00
1067	0.10	0.00	0.00
2133	0.03	0.00	0.00

(b) Voxelized method.

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	18	49.04	10.93	132.68
16^2	35	44.15	5.23	81.68
32^2	68	29.33	2.65	35.36
64^2	135	27.41	1.39	22.00
128^2	268	17.00	0.45	3.94
256^2	534	2.97	0.07	0.10
512^2	1067	2.00	0.03	0.04
1024^2	2133	0.44	0.01	0.00

(c) Boundary accurate method.

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	18	38.51	3.58	36.80
16^2	35	12.93	1.20	3.87
32^2	68	3.44	0.40	0.33
64^2	135	7.83	0.21	0.63
128^2	268	0.45	0.04	0.01
256^2	534	0.25	0.01	0.00
512^2	1067	0.17	0.01	0.00
1024^2	2133	0.05	0.00	0.00

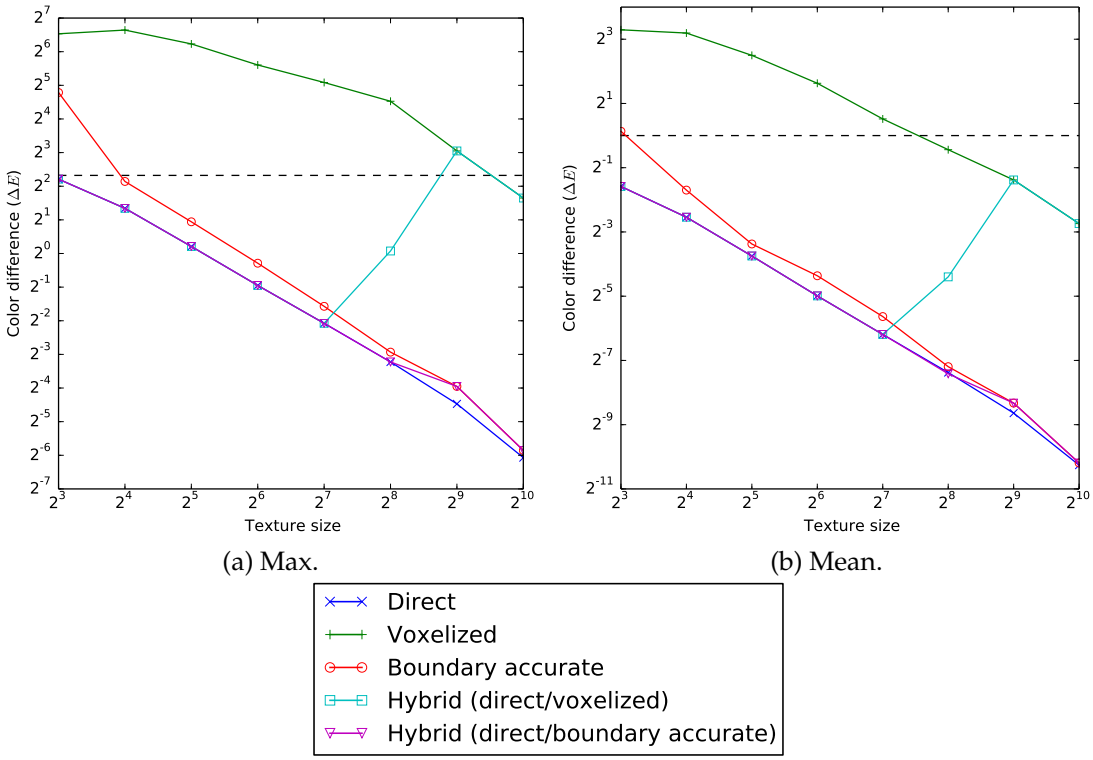
(d) Hybrid method (direct/voxelized).

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	18	19.68	1.69	6.93
16^2	35	7.35	0.55	1.01
32^2	68	3.26	0.17	0.14
64^2	135	1.52	0.06	0.03
128^2	268	0.55	0.03	0.00
256^2	534	2.97	0.07	0.10
512^2	1067	2.00	0.03	0.04
1024^2	2133	0.44	0.01	0.00

(e) Hybrid method (direct/boundary accurate).

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	18	19.68	1.69	6.93
16^2	35	7.35	0.55	1.01
32^2	68	3.26	0.17	0.14
64^2	135	1.52	0.06	0.03
128^2	268	0.55	0.03	0.00
256^2	534	0.25	0.01	0.00
512^2	1067	0.17	0.01	0.00
1024^2	2133	0.05	0.00	0.00

Table 4.2: Statistics for Case B: Trivial geometry, non-trivial scalar field. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. For the methods that use a voxel grid, the texture sizes are shown under #V.



(a) Max. (b) Mean.

Figure 4.10: Illustration of the visual accuracy for Case C: Non-trivial geometry, trivial scalar field, similarly to figure 4.8.

4.6 Case C: Non-trivial geometry, trivial scalar field

In cases with a non-trivial geometry, the simplified model only approximates the boundaries of the isogeometric model. It is therefore expected that at least the pure voxelized method will lose some of the visual accuracy compared to the direct method. Graphs for this case are shown in figure 4.10, and shows that the voxelized method consistently has the worst color difference among all the methods. The boundary accurate method is not affected as much by this, since it uses the boundaries of the spline geometry instead. It therefore significantly improves the visual accuracy compared to the voxelized method. Increasing the resolution of the voxel grid will lead to a better approximation of the geometry, so it is again expected that the results get closer to the reference solution with higher voxel grid resolutions.

Once again it can be seen that the boundary accurate method always generates better results when doubling the texture size, while the voxelized hybrid method does not. When using a texture size of 512^2, the voxelized hybrid method does not even meet the requirement for maximum color difference. The visual accuracy is largely affected by the boundary issues with the voxelized method, which also affects the direct/voxelized hybrid method. The geometric criterion is sensible for

the direct/boundary accurate hybrid method, which universally has a maximum color difference below 5.0.

(a) Direct method.

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
17	4.63	0.33	0.55
32	2.54	0.17	0.16
63	1.16	0.07	0.03
125	0.52	0.03	0.01
248	0.24	0.01	0.00
494	0.11	0.01	0.00
986	0.05	0.00	0.00
1971	0.01	0.00	0.00

(b) Voxelized method.

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	18	92.56	9.80	272.28
16^2	34	99.95	9.14	363.63
32^2	60	75.12	5.65	165.88
64^2	122	48.67	3.09	57.81
128^2	245	33.93	1.43	17.92
256^2	492	23.04	0.74	6.52
512^2	983	8.28	0.38	1.28
1024^2	1968	3.14	0.15	0.18

(c) Boundary accurate method.

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	17	27.59	1.10	9.29
16^2	32	4.41	0.31	0.45
32^2	62	1.92	0.10	0.06
64^2	124	0.82	0.05	0.02
128^2	247	0.34	0.02	0.00
256^2	493	0.13	0.01	0.00
512^2	985	0.06	0.00	0.00
1024^2	1970	0.02	0.00	0.00

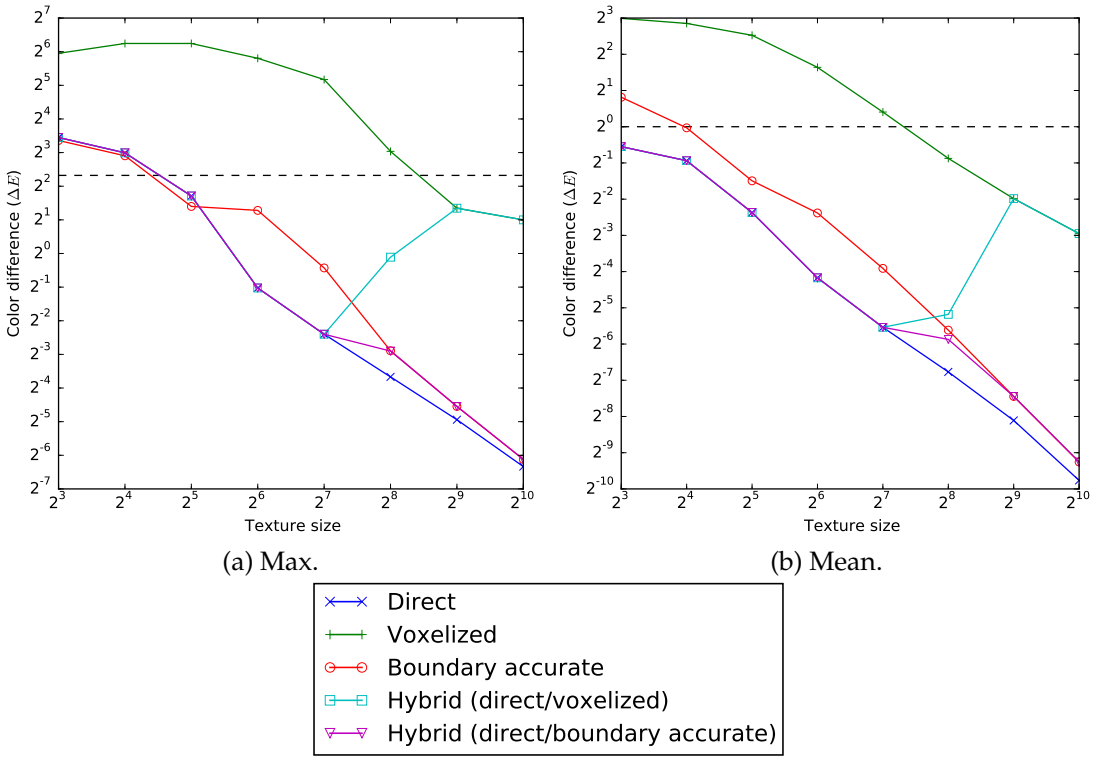
(d) Hybrid method (direct/voxelized).

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	17	4.63	0.33	0.55
16^2	32	2.54	0.17	0.16
32^2	63	1.16	0.07	0.03
64^2	125	0.52	0.03	0.01
128^2	248	0.24	0.01	0.00
256^2	492	1.05	0.05	0.03
512^2	983	8.28	0.38	1.28
1024^2	1968	3.14	0.15	0.18

(e) Hybrid method (direct/boundary accurate).

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	17	4.63	0.33	0.55
16^2	32	2.54	0.17	0.16
32^2	63	1.16	0.07	0.03
64^2	125	0.52	0.03	0.01
128^2	248	0.24	0.01	0.00
256^2	493	0.11	0.01	0.00
512^2	985	0.06	0.00	0.00
1024^2	1970	0.02	0.00	0.00

Table 4.3: Statistics for Case C: Non-trivial geometry, trivial scalar field. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. For the methods that use a voxel grid, the texture sizes are shown under #V.



(a) Max. (b) Mean.

Figure 4.11: Illustration of the visual accuracy for Case D: Non-trivial geometry, non-trivial scalar field, similarly to figure 4.8.

4.7 Case D: Non-trivial geometry, non-trivial scalar field

This is the most interesting case, as the dataset reflects a more practical scenario. As has been seen when the geometry or the field was approximated, the visual accuracy gets better as the resolution of the voxel grid is increased. When these approximations are combined, it is therefore also expected that the same trend will be visible here. Figure 4.11 illustrates the visual accuracy for this case.

The boundary accurate method meets the requirements as soon as the direct method does. It is therefore a good alternative in this case, and is also likely to achieve higher performance since it has fewer spline evaluations. Both of the hybrid methods generate results with color differences lower than the thresholds for all the chosen texture sizes, which indicates that a hybrid method based on a geometric criterion works as expected.

(a) Direct method.

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
17	10.92	0.68	2.24
32	7.96	0.52	1.69
63	3.27	0.19	0.28
125	0.49	0.06	0.01
248	0.19	0.02	0.00
494	0.08	0.01	0.00
986	0.03	0.00	0.00
1971	0.01	0.00	0.00

(b) Voxelized method.

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	18	61.90	7.95	148.85
16^2	34	75.82	7.22	162.51
32^2	60	75.82	5.75	172.59
64^2	122	55.86	3.11	58.76
128^2	245	36.04	1.32	15.85
256^2	492	8.20	0.55	1.38
512^2	983	2.54	0.25	0.25
1024^2	1968	2.00	0.13	0.09

(c) Boundary accurate method.

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	17	10.27	1.76	5.89
16^2	32	7.49	0.98	2.28
32^2	62	2.64	0.36	0.32
64^2	124	2.43	0.19	0.16
128^2	247	0.74	0.07	0.02
256^2	493	0.13	0.02	0.00
512^2	985	0.04	0.01	0.00
1024^2	1970	0.01	0.00	0.00

(d) Hybrid method (direct/voxelized).

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	17	10.92	0.68	2.24
16^2	32	7.96	0.52	1.69
32^2	63	3.27	0.19	0.28
64^2	125	0.49	0.06	0.01
128^2	248	0.19	0.02	0.00
256^2	492	0.93	0.03	0.01
512^2	983	2.54	0.25	0.25
1024^2	1968	2.00	0.13	0.09

(e) Hybrid method (direct/boundary accurate).

#V	max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)
8^2	17	10.92	0.68	2.24
16^2	32	7.96	0.52	1.69
32^2	63	3.27	0.19	0.28
64^2	125	0.49	0.06	0.01
128^2	248	0.19	0.02	0.00
256^2	493	0.13	0.02	0.00
512^2	985	0.04	0.01	0.00
1024^2	1970	0.01	0.00	0.00

Table 4.4: Statistics for Case D: Non-trivial geometry, non-trivial scalar field. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. For the methods that use a voxel grid, the texture sizes are shown under #V.

Chapter 5

Comparison of methods in 3D

Different methods for ray-casting isogeometric models were described in chapter 3. This chapter deals with how these were implemented in 3D.

5.1 3D framework

Fuchs and Hjelmervik [8] developed a framework for visualizing isogeometric volumes, and this framework was used as a basis for further implementation. Before project start, the framework already contained implementations for the direct method and the voxelized method, so these were not implemented as part of this project.

5.1.1 Software and tools

The software used was mostly dependent on the existing framework. The framework had been implemented in C++, and with OpenGL as the graphics API for interacting with the GPU. Additional libraries are used for components such as GUI, but since these are not relevant for this project they will not be covered here in detail. During the project the C++ code was compiled using the GNU Compiler Collection (gcc 4.8.4) on a system running Ubuntu 14.04.

5.1.2 Differences compared to the 2D prototype

While developing the 2D prototype, some choices were made to simplify the problem of ray-casting isogeometric models. This included limiting the number of volume blocks to one, and assuming that each view-ray only intersects with the model a maximum of two times. These limitations do not apply for the 3D implementation.

In the prototype a view-ray was defined to be a point in the geometry along with a directional vector. In the direct method, an intersection point was found by approximating the roots to a function with a small tolerance. The function was based on an expression of the view-ray as well as another expression for the spline model boundary. In the 3D implementation, a view-ray is defined by an even number of

intersection points with the boundaries of the isogeometric model. These are found by solving the problem of finding approximations of the surfaces of the volume. For this the approach in [10] is used, which is an efficient implementation due to the use of the hardware tesselator of modern GPUs. Finding the intersections are done in the first pass while rendering. The results are stored in textures, including the parameter for each intersection point.

It was difficult to evaluate the performance of the different methods in the prototype. Evaluating performance by measuring the time spent rendering would not give an accurate description, as the setup was not reflecting the 3D setup well since all of the code ran on the CPU. While the ray-casting happens in parallel on the GPU, they are sequential on the CPU. Some features like texture interpolation that could be hardware accelerated in 3D, was limited to software implementations in the prototype. In the 3D implementation the performance of the methods can be compared by measuring the time for drawing a complete frame.

5.1.3 Limitations

Fuchs and Hjelmervik [8] presented different approaches for approximating the inverse of a spline function. While implementing the new ray-casting methods, only the Newton-Raphson method has been used for this purpose. However, using another approach should not pose any difficulty in the implementation.

It has also been assumed that color mapping is carried out with a 1D transfer function. This is for the same reason as with the 2D prototype to reduce the scope of the project, and does not affect the results of comparing different methods in any meaningful way.

5.2 Implementation of ray-casting methods

The direct and the voxelized methods were implemented in the framework ahead of time, and will therefore not be covered in great detail here. The 2D prototype was limited to isogeometric models of only one volume block. In the general case and in the 3D implementation, an isogeometric models may consist of multiple volume blocks. The voxel grid is created in a pre-process step by first determining a bounding box enclosing the whole isogeometric model. The model is then sampled with orthogonal projection from one direction. This means that even though the isogeometric model may consist of multiple volume blocks, the result will be one voxel grid texture for the whole model.

5.2.1 Boundary accurate methods

For the boundary accurate method, the approach to finding intersections are identical to the approach by the direct method. The parameter of the first intersection point is fetched from one of the textures generated in the first pass, and the corresponding

point in the geometry is determined by evaluating the spline function ϕ defining the geometry. The next step is to sample the model between pairs of intersection points, along the view-ray in the geometry. For the boundary accurate method all the internal sample points are attempted sampled from the voxel grid. In some cases, the voxel grid will not contain data at a sample point. These points are marked as invalid, and the boundary accurate method skips these sample points.

Subsequent sample points are calculated using the geometric representation of the first intersection point and the direction of the view-ray. Coordinates in the texture ranges from 0 to 1, and is determined based on the position of the current sample point in the geometry. This is done by equation 3.2, similarly to the 2D prototype except that the vectors are now three dimensional. Since points in the geometry are used to determine points in the texture, sampling internally in multiple volume blocks does lookup in the same voxel grid.

The texture representing the voxel grid is rectilinear, but the isogeometric model may not be. Each entry in the texture has therefore two elements; one scalar value, and a value that indicates whether or not the current point is inside the model. For each sample point the indicator is checked before using the scalar value. When the distance between the current sample point and the first intersection point is longer than the distance between the two intersection points, the sampling from the texture ends. The scalar at the out intersection point is then used. Sampling can alternatively end earlier if the pixel color is saturated while compositing along the view-ray.

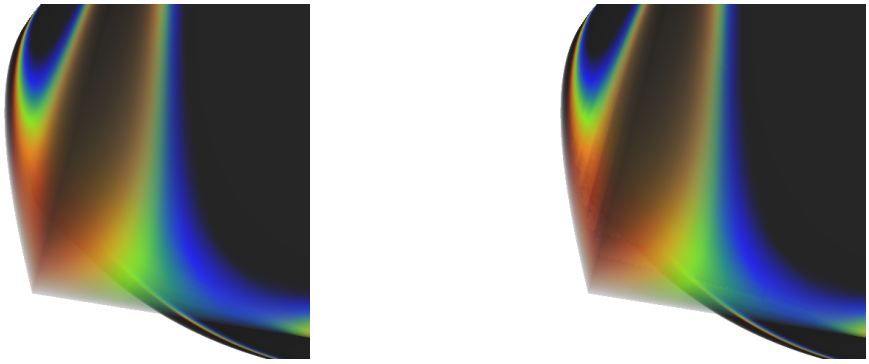
Similar to the prototype, the problem of sampling at the edge of the texture is solved by clamping to border. In OpenGL this is implemented by setting the appropriate flag as texture parameters, see listing 5.1. This causes interpolating with the border value to return an invalid sample.

Listing 5.1: C++ implementation of setting the texture wrap parameter for the boundary accurate method.

```
float color[] = { 1.0f, 1.0f, 1.0f, 1.0f };
glTexParameterfv(GL_TEXTURE_3D, GL_TEXTURE_BORDER_COLOR, color);
glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_BORDER);
```

While the boundary accurate method gave very good visual accuracy results in the 2D prototype, it turns out it can introduce artifacts in 3D for specific view-angles, see figure 5.1. The reason for this turns out to be because the sampling distance between the first intersection point and the next sample point can vary, and the change can be very big from one pixel to the next adjacent to it. While two intersection points for two adjacent pixels may have almost the same scalar value, if the sample distance to the next sample point is different, the scalar value at the next sample point may be completely different. An illustration of the problem in 2D is shown in figure 5.2. Some areas along the boundary of the model may therefore not be sampled at all.

To reduce the effect of this problem the spline model can be sampled in these areas.



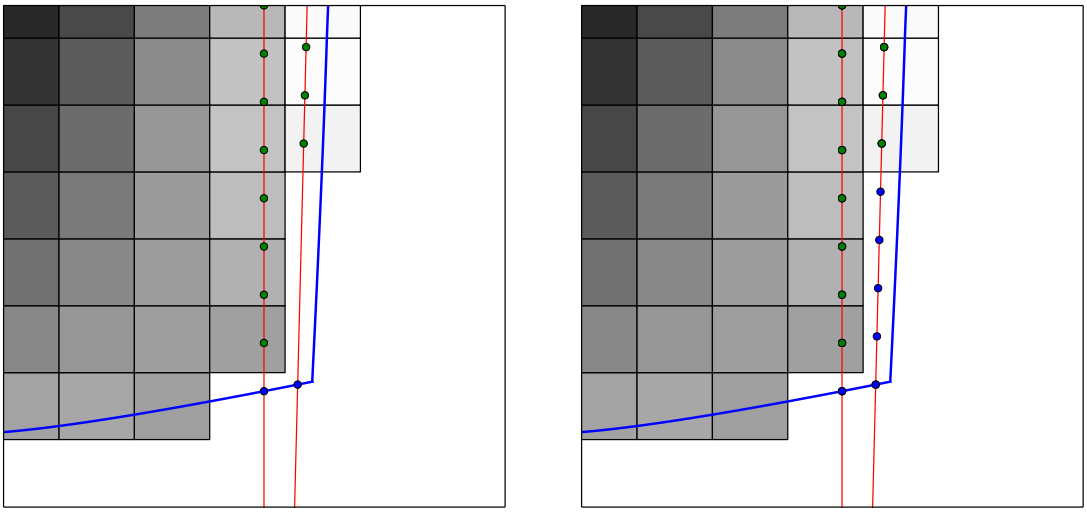
(a) Direct method. (b) Thin boundary accurate method.

Figure 5.1: The boundary accurate method can introduce visual artifacts for specific view-angles.

This will prevent there from being sudden changes in sample distances due to the approximation of the geometry by the voxel grid. As opposed to the original boundary accurate method which only uses the shell boundary of the spline model, this approach uses a thicker boundary where the spline model extends further out than the voxel grid. This alternative approach will therefore be called the *thick* boundary accurate method, or thick method. The original boundary accurate method will also be called the thin method to distinguish between them. Figure 5.3 shows the same example as in figure 5.1 with the thick method, and the serrated visual artifacts have reduced visibility. The additional spline evaluations will however most likely lead to a lower performance than the thin boundary accurate method.

The thick method starts by evaluating the spline model at an intersection point. It will then traverse the volume, and may sample from the spline model at additional sample points before sampling from the voxel grid. While approximating the inverse of the geometric spline, the Newton-Raphson method will have quadratic convergence given that the initial guess is good. The sampling distance between sample points is typically small, so a good initial guess is to use the parameter at the previous sample point. After the thick boundary accurate has sampled from the voxel grid, it may need a sample where the voxel grid texture indicates it has no valid value. Attempting to sample from the spline model here can be problematic, as there may not be a good guess for the parameter for the Newton-Raphson method.

This problem can be solved in different ways. One way is to store parameters in the voxel grid texture. This could mean having a good initial guess for the Newton-Raphson method after having sampled the voxel grid texture. By interpolating parameter values in the texture the guess might be better. Without interpolation the closest parameter value could be used, but this would typically not be particularly close compared to the next intersection point. It does however drastically increase the texture size. Larger textures may be a better use of the memory space, because they may increase the visual accuracy more than using this method.



(a) Sample points when using the boundary accurate method. The spline model is only sampled where the view-rays intersect with the isogeometric model.

(b) A solution to the problem by modifying the boundary accurate method slightly. The spline model is additionally sampled at internal points where the voxel grid texture does not contain data.

Figure 5.2: A 2D illustration of the problem with the boundary accurate method. The blue line represent the boundary of the spline model, while the greyscale rectangles represent the voxels in the voxel grid. The red lines represent two of the view-rays from the camera below the model, that cast upward through the model. Blue dots represent sample points where the spline model has been sampled, while green dots indicate samples from the voxel grid. Due to the area inside the spline model that has no voxels, the boundary accurate method has a larger sample distance between the first sample points on the rightmost view-ray. This leads to a somewhat different scalar value in this case, and therefore also a noticeable different resulting pixel color. The end result can be sharp transitions of color in the model, causing visual artifacts.

Another solution is to ray-cast from out-intersection points in the opposite direction. This would be similar as for the in-intersection points, and a good parameter guess is updated for each subsequent point into the volume. If the volume is largely opaque however, these sample points may not contribute to the end result because the color is saturated before the ray reaches these sample points. This means that some work may be wasted. The compositing would also be more complicated. While the normal ray composites from front to back, the opposite ray does back-to-front compositing. Handling two compositing schemes at the same time would also increase the number of registers and variables needed in the shader.

The solution used in the thick method is to run more iterations of the Newton-Raphson method when approximating the parameter $\tilde{p}_i$ of the first sample point g_i

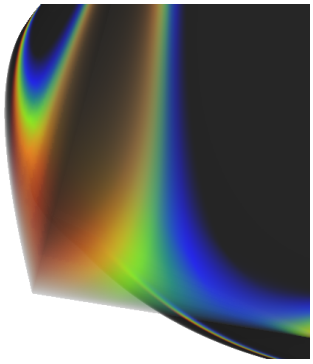


Figure 5.3: The thick method reduces the visual artifacts introduced by the thin boundary accurate method.

following the sample points from the voxel grid. Normally the Newton-Raphson method has three conditions for termination. The first is after a given number of iterations, the second is if the distance between g_i and $\tilde{g}_i = \phi(\tilde{p}_i)$ is below a certain threshold, and the last one is if $\tilde{g}_i$ is inside the pixel frustum. The last termination condition could lead to using the parameter for a point far from the actual sample point. This condition is therefore disabled for the first sample point from the spline model after the voxel grid. In addition, the number of allowed iterations is doubled to 16 compared to the normal 8.

Note that in the example shown in figure 5.1, the visual artifacts are worst on a part of the model close to the observer. How much these affect the result when the model is further into the scene is difficult to tell. Another solution may therefore be to use a hybrid method.

5.2.2 Hybrid methods

For the hybrid method in the 2D prototype, there was a geometric criterion that when met it had been assured that there was pixel accuracy for sampling from the voxel grid. This criterion was trivial in that case, as the eye and all sample points were limited to one plane. In OpenGL, projection is achieved by the matrix-vector multiplication

$$\begin{bmatrix} A & 0 & B & 0 \\ 0 & C & D & 0 \\ 0 & 0 & E & F \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_{es} \\ y_{es} \\ z_{es} \\ w_{es} \end{bmatrix} = \begin{bmatrix} x_{cs} \\ y_{cs} \\ z_{cs} \\ w_{cs} \end{bmatrix}, \tag{5.1}$$

where the 4x4 matrix is the so-called *projection matrix*, and vectors in eye space (es) are transformed into vectors in clip space (cs). Similarly to Hjelmervik [10], this will be used as a basis for calculating the maximum projected error from a given sample point that ensures pixel accuracy. Vectors in clip space are transformed to screen space (ss) by dividing the x_{cs}, y_{cs} and z_{cs} components by the w_{cs} component. For the x component

this becomes

$$x_{ss} = -\frac{Ax_{es} + Bz_{es}}{z_{es}}. \tag{5.2}$$

Pixel accuracy is assured when all the sample points for a ray is projected into the current pixel in screen space. From the pixel center, there is a maximum perturbation of half the pixel size that is acceptable to be part of the same pixel. If the normalization and the projection is reversed, we have the maximum perturbation in eye space. This is of interest because it can be used as a basis for a geometric criterion.

With perturbation ϵ in eye space equation 5.1 becomes

$$\begin{bmatrix} A & 0 & B & 0 \\ 0 & C & D & 0 \\ 0 & 0 & E & F \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_{es} + \epsilon_{es,x} \\ y_{es} + \epsilon_{es,y} \\ z_{es} + \epsilon_{es,z} \\ w_{es} \end{bmatrix} = \begin{bmatrix} x_{cs} + \epsilon_{cs,x} \\ y_{cs} + \epsilon_{cs,y} \\ z_{cs} + \epsilon_{cs,z} \\ w_{cs} \end{bmatrix}, \tag{5.3}$$

and subsequently the first component of the screen space coordinate becomes

$$x_{ss} + \epsilon_{ss,x} = \frac{A(x_{es} + \epsilon_{es,x}) + B(z_{es} + \epsilon_{es,z})}{-(z_{es} + \epsilon_{es,z})}. \tag{5.4}$$

To simplify matters, $\epsilon_{es,x} = \epsilon_{es,y} = \epsilon_{es,z} = \epsilon_w$, since it is based on the width of a pixel. This means that there will be a bounding sphere with radius ϵ_w around a given point, where any sample point that is bounded by this sphere will be pixel accurate. Equation 5.4 can then be written as

$$\epsilon_w = -\frac{Ax_{es} + z_{es}(B + x_{ss} + \epsilon_{ss,x})}{A + B + x_{ss} + \epsilon_{ss,x}}. \tag{5.5}$$

Inserting 5.2 into 5.5 results in

$$\epsilon_w = \frac{\epsilon_{ss,x} {z_{es}}^2}{A(|x_{es}| - z_{es}) - \epsilon_{ss,x} z_{es}} \tag{5.6}$$

which describes the maximum allowed perturbation in eye space that ensures pixel accuracy based on the width of a pixel. Similarly, the maximum allowed perturbation based on pixel height is given by

$$\epsilon_h = \frac{\epsilon_{ss,y} {z_{es}}^2}{C(|y_{es}| - z_{es}) - \epsilon_{ss,y} z_{es}}. \tag{5.7}$$

Deciding if the voxel grid can be used is then determined similarly as in the 2D prototype, which was covered in section 3.7. The distance V_d between two diagonally adjacent voxels is now based on three dimensions. To have pixel accuracy, the sample point must be within the pixel frustum. When sampling from the voxel grid, the result is an interpolated scalar s based on the surrounding voxel points $(v_{i,k})_{k=0}^{7}$. However, the point ι where $\rho(\phi^{-1}(\iota)) = s$ can be anywhere within the

bounding box defined by $(v_{i,k})_{k=0}^{7}$. This bounding box must therefore be within the pixel frustum to achieve pixel accuracy. Since the sample point may be close to a voxel point, the point ι can in the worst case be V_d in distance from the sample point. To simplify the criterion, there can be assurance that the pixel frustum encloses all points with distance V_d from the sample point when $0.5\epsilon_h > V_d$ and $0.5\epsilon_w > V_d$. If the tests pass for a sample point, the voxel grid can be sampled with pixel accuracy. Listing 5.2 shows the implementation in the fragment shader.

Listing 5.2: Implementation of the geometric criterion in the OpenGL Shading Language. `P` is the projection matrix, while `MV` is the model-view matrix, transforming vectors in the geometry to eye space. `pixel_size` is a uniform vector defined to be the pixel size in x and y direction in clip space, and `voxel_distance` is a uniform variable set to the distance V_d between two diagonally adjacent voxels. The Boolean variable `useVoxelGrid` is then later used to determine whether the current sample point should be sampled from the spline model or the voxel grid.

```
vec4 current_eyespace = MV * vec4(current_geo, 1.0f);
float A = P[0][0];
float C = P[1][1];
float x = abs(current_eyespace.x);
float y = abs(current_eyespace.y);
float z = current_eyespace.z;

float epsilonW = (pixel_size.x*z*z) / (A*(x-z) - pixel_size.x*z);
float epsilonH = (pixel_size.y*z*z) / (C*(y-z) - pixel_size.y*z);

bool useVoxelGrid = 0.5*min(epsilonW, epsilonH) > voxel_distance;
```

In the 2D prototype, two different variations of the hybrid method were analyzed and compared to each other. The results did however show that the direct/voxelized hybrid never had better visual accuracy than the direct/boundary accurate method. The cost comes at the price of performance, but since the voxelized model approximates the boundaries of the geometry poorly, the increase in visual accuracy compared to the cost is greater on the boundaries than on average. It is likely that when using the direct/voxelized hybrid method, the texture size must be higher in general. The direct/voxelized hybrid method was therefore not implemented in the 3D implementation.

5.3 Comparing the methods

In the 2D prototype, it was shown that it was possible to combine the spline model with a simplified voxelized representation of the same model. This was to visualize an isogeometric model with most likely higher performance, and with visual results within the acceptable threshold. From the comparison in chapter 4, it was evident that

some methods had better properties than others, usually with a tradeoff between visual accuracy and possibly performance. Hybrid methods was introduced and allowed switching between models dynamically based on the resolution of the voxel grid. The hybrid combining the direct method with the pure voxelized method was however considered to not be ideal, and has therefore not been implemented in 3D. Because of the possible visual artifacts from the original boundary accurate method, a slightly modified method was also introduced as an alternative, called the thick method.

In the three first cases presented here, a model of a twisted bar will be used as the isogeometric volume. It is the same as the one used in the in-depth example shown by Fuchs and Hjelmervik [8]. Additionally, the same transfer function that were used in their example has been used in cases A and B. This transfer function provides a model with high transparency, which makes it suited for testing volume rendering performance, and has some sharp transitions, which can make visual accuracy more difficult for discretized models. In all of the cases, a texture of size 512^3 has been used as the voxel grid. Below is a brief overview of the different cases.

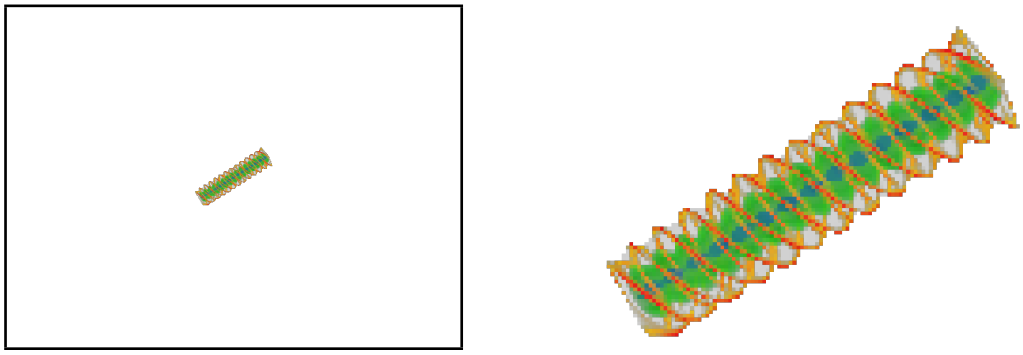
- *Case A: Hybrid switchover*. In this case the distance from the observer to the model has been set such that the threshold where the geometric criterion passes goes through the volume. It is the most complete example as it gives different results for each method, and is therefore also covered in most detail. The purpose is to have an example where all the methods can be tested, and for showing whether or not a hybrid based on a geometric criterion works in practice.

- *Case B: Far*. A similar case to case A, but this time the model is placed further away from the observer. It is placed at a distance where the geometric criterion is always met. This demonstrates how well the alternative methods can perform compared to the direct method.

- *Case C: Visual artifacts*. This case demonstrates the problem with the thin boundary accurate method. Even though the visual accuracy are comparable between the thin method and the thick method in most cases, the thin method can give poor visual accuracy for certain transfer functions and view-angles.

- *Case D: Multiple volume blocks*. While the other cases use a model consisting of only one volume block, isogeometric models usually consist of multiple volume blocks. The purpose of this case is to demonstrate the methods on a more industrially relevant model.

Performance have been measured in frames per second (FPS) while rendering on an NVIDIA GeForce GTX 680 GPU. The scene has been rendered in an OpenGL viewport with the resolution of 640x480. Reference solutions for each case have been generated by using a sample distance of 0.0005, which is half of the lowest sample distances used for the other methods. The supersampling in the reference solutions is also performed with 40 steps, compared to 20 steps for the other methods.



(a) Screen capture of the full 640x480 OpenGL viewport.

(b) Each individual pixel made more easily visible after cropping and scaling the screen capture.

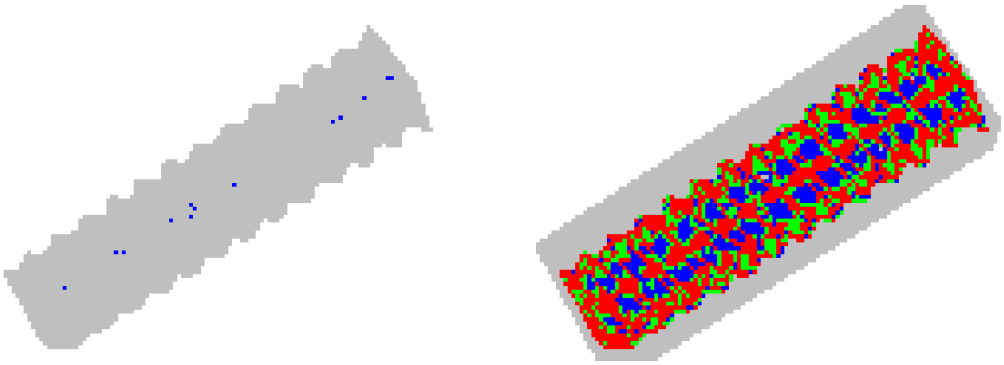
Figure 5.4: Visualization of the reference solution for Case A: Hybrid switchover.

5.4 Case A: Hybrid switchover

Visualization of the twisted bar model can be seen in figure 5.4a. It shows a screen capture of the whole 640x480 viewport while rendering the reference solution. The reason why the model is placed at that distance from the camera is because the hybrid methods will be tested on the same example. To make each individual pixel more easily visible, figure 5.4b shows the same example, but the image has been cropped and scaled with point filtering. The rest of the figures from this example have been similarly cropped and scaled.

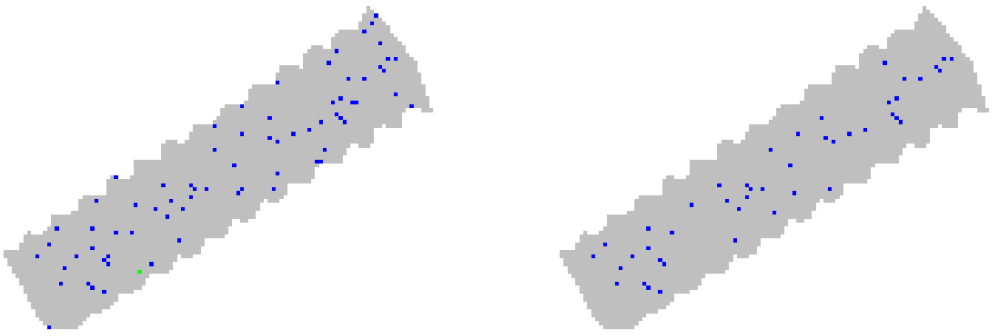
After generating the reference solution, a voxelized representation of the model was created. Due to the shape of the model, this means that the distance between voxels is greater along the z-axis of the model, compared to the other two axes. The isogeometric model is then visualized with each of the different methods. The orientation of the model and the transfer function have been kept the same for all methods so that they can be compared against each other. For each method a range of sampling distances are used, and screen captures are made. These screen captures are then compared to the reference solution with the CIEDE2000 algorithm, which generates a scalar for color difference for each pixel pair. Figure 5.5 shows a comparison of the color differences for the different ray-casting methods. The color differences have been visualized according to table 3.1. Note that the images here don't show the full OpenGL viewport, but have been cropped and scaled with point filtering to make each individual pixel more easily visible.

The direct method does not perfectly portray the reference solution, as can be seen in figure 5.5a. This is due to having a larger sampling distance and fewer steps while supersampling. However, the visualization does not show any green or red pixels, which means that the maximum color difference ΔE is below the threshold of 5.0. Having more grey pixels than blue is also a good indication that the mean color difference ΔE is below the threshold of 1.0. The voxelized method on the other hand,



(a) Direct

(b) Voxelized



(c) Thin boundary accurate

(d) Thick boundary accurate

Color	Description
	No noticeable color difference
	Acceptable color difference
	Noticeable color difference between most color pairs
	Clear color difference between color pairs

(e) Color codes used for CIEDE2000 color difference.

Figure 5.5: Case A: Hybrid switchover. Color differences for different methods compared to a reference solution. The methods based on using a voxel grid have used a texture size of 512^3. All the methods have used the same sampling distance of 0.008, and supersampling with 20 steps.

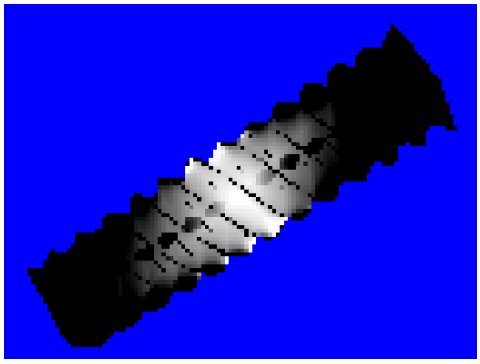
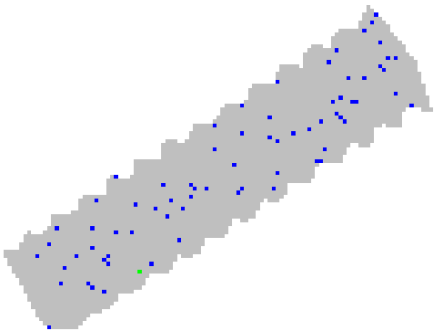


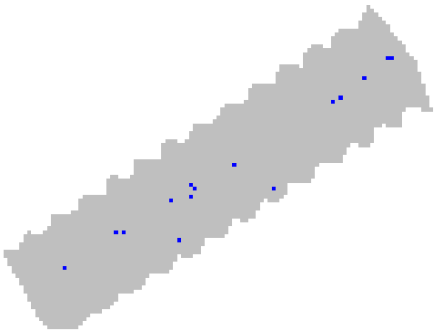
Figure 5.6: Hybrid ratio for Case A: Hybrid switchover. The ratio represents the number of samples from the spline model compared to the number of samples from the voxel grid. For pixels with black color, all of the samples have been sampled from the spline model. The lighter the color is; the more samples have been made from the voxel grid.

does not produce a result with good visual accuracy. This is again due to the problem the voxelized method has of representing the boundary. This can be confirmed by comparing figure 5.5b to the thin boundary accurate method in figure 5.5c, where the latter has a significantly improved color difference by using the boundaries from the spline model. The CIEDE2000 result for the thin method does however show a green pixel, meaning that the color difference ΔE for this pixel is above the threshold of 5.0. The thick boundary accurate method can in theory generate results with better visual accuracy, but only for the pixels where the conditions are such that they can take more samples. Figure 5.5d shows the result for the thick method, and it is apparent that results are improved. Since there are no red or green colored pixels, there is a good indication that the requirements set in section 3.2.2 are met.

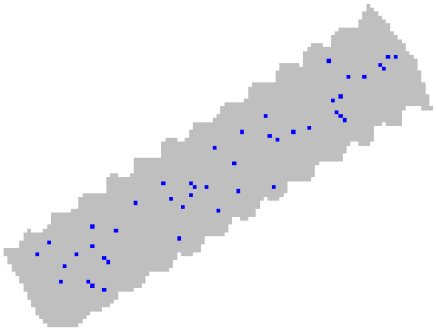
Since the thin method does not meet the requirements for visual accuracy, a reasonable explanation is that in this example, the resolution of the voxel grid is too small to use the method directly. A hybrid method can be used to take advantage of the voxel grid where possible. Figure 5.6 shows the hybrid ratio for this example, based on the geometric criterion explained in section 5.2.2. It illustrates for a hybrid method, how many samples are taken from the voxel grid compared to from the spline model for each ray. Dark color means mostly samples from the spline model, and light color means mostly samples from the voxel grid. Note that since the boundary accurate methods evaluate the spline model at the boundaries, there will be at least one sample from the spline model for each pixel. It can be seen that in the middle of the screen, the geometric criterion is met earlier than at the pixels further from the center. The geometric criterion is based on the size of the pixel frustum, and the closer the observer is to the screen, the wider the pixel frustum gets. Since the pixels in the middle of the screen are the closest to the observer, they will therefore also meet the geometric criterion earlier.



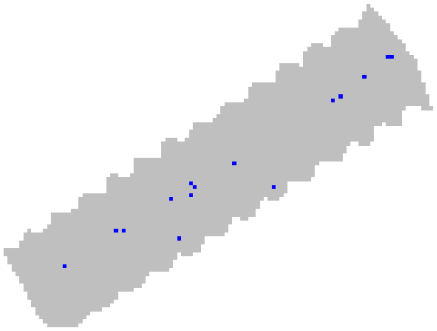
(a) Thin boundary accurate



(b) Hybrid (direct/thin boundary accurate)



(c) Thick boundary accurate



(d) Hybrid (direct/thick boundary accurate)

Color	Description
	No noticeable color difference
	Acceptable color difference
	Noticeable color difference between most color pairs
	Clear color difference between color pairs

(e) Color codes used for CIEDE2000 color difference.

Figure 5.7: Case A: Hybrid switchover. Color differences for the boundary accurate methods and their corresponding hybrid methods. The methods based on using a voxel grid have used a texture size of 512^3. All the methods have used the same sampling distance of 0.008, and supersampling with 20 steps.

Comparing the hybrid ratio to the color difference results for the thin boundary accurate method, it can be seen that the pixel with the largest pixel difference is in the dark area of the hybrid ratio illustration. This means that sampling from the voxel grid here, as was done when ray-casting, is not guaranteed to be pixel accurate. The result could therefore be improved by using a hybrid method.

Illustrations of color differences with hybrid methods are shown in figure 5.7. It is apparent that the visual accuracy improves in the areas such as at the left and right ends where the geometric criterion is not met. For this example, the threshold criteria are met for both of the hybrid methods. The results from the thin and the thick boundary accurate hybrid methods are quite similar, and the color difference illustrations are identical.

More detailed results including performance measurements can be seen in table 5.1. For each subsequent row in each table, the sampling distance is halved. The same sampling distances have been used for the pure voxelized method as well, but the number of sample points differ because the view-rays intersect with the model at different points than for methods that use the spline boundaries. The color difference illustrations in figures 5.5 and 5.7 corresponds to the fifth data row in each table. As expected, the direct method converges towards the reference solution, as was shown in [8]. This is also true for the voxelized method, although it converges more slowly. When comparing the thin and the thick methods, the thick method does consistently generate results with slightly lower color differences at the cost of a moderate drop in performance. The mean color difference converges towards the reference solution for both of these methods. The max color difference does however fluctuate slightly. This is likely due to the reconstruction of the continuous functions from the discretized voxel grid, which is done with trilinear interpolation. Since the isogeometric model is not linear, the reconstruction is only an approximation. It can be seen from the tables that the maximum color difference can also increase with a lower sampling distance for the pure voxelized method. For the boundary accurate methods, the mean color differences are below the threshold for the same sampling distances as for the direct method, but the maximum color differences do not consistently stay below the threshold. They do however for the hybrid methods, and here the difference between the thin and the thick methods are small.

In this example the thick method would be the best alternative to the direct method, since it can meet the threshold requirements with the best performance. However, pixel accuracy is not assured for all the samples from the voxel grid since the geometric criterion is not met for all sample points in the model. This is assured by the hybrid methods, and both of these have performance similar to the direct method.

For the example with the sampling distance of 0.008, the thin boundary accurate hybrid method has about 3% more frames per second while rendering, while the thick boundary accurate hybrid method has about 8% less. Comparatively, the regular thick method has about 2.9 times more frames per second than the direct method. Graphs of the max and the mean color differences for this case is shown in figure 5.8.

(a) Direct method.

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
10	50.343	2.469	16.385	354.9
20	17.393	1.096	3.021	213.4
39	5.030	0.434	0.363	118.9
78	1.683	0.192	0.063	63.2
156	1.131	0.111	0.031	32.8
312	1.062	0.076	0.022	18.4
623	1.023	0.059	0.017	8.7
1246	1.058	0.039	0.009	4.3

(b) Voxelized method.

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
9	71.388	12.448	239.877	2601.4
17	70.058	11.506	209.217	2122.9
33	68.528	10.492	179.643	1571.4
65	66.525	9.265	141.591	1030.0
130	56.639	8.063	106.217	630.4
260	56.646	7.331	87.167	357.3
519	56.392	6.920	78.371	206.2
1037	56.393	6.671	72.561	107.2

(c) Thin boundary accurate

(d) Hybrid (direct/thin boundary accurate)

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
10	50.346	2.494	16.500	849.2
20	17.216	1.130	3.138	740.3
39	6.828	0.480	0.424	583.8
78	6.541	0.256	0.119	414.2
156	5.132	0.205	0.090	263.9
312	4.928	0.185	0.090	156.4
623	4.821	0.173	0.090	86.7
1246	4.825	0.166	0.092	46.1

(e) Thick boundary accurate method.

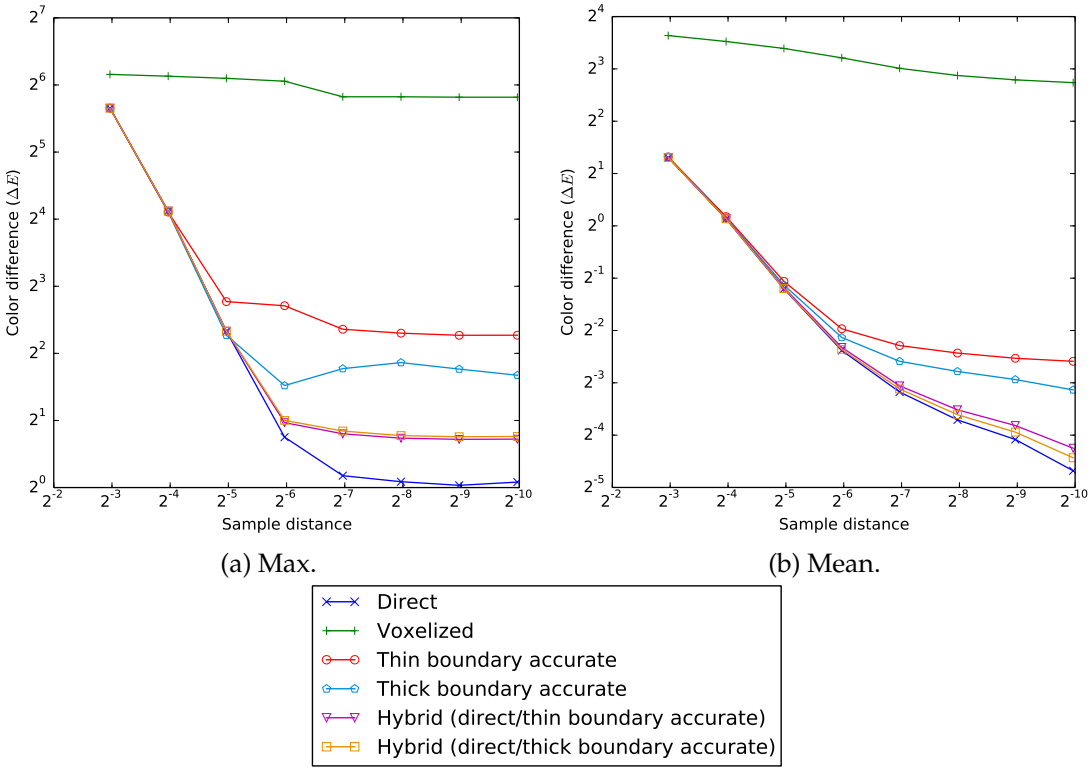
max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
10	50.346	2.475	16.411	363.3
20	17.393	1.103	3.033	216.1
39	5.030	0.442	0.368	120.2
78	1.955	0.200	0.066	64.2
156	1.743	0.120	0.034	33.8
312	1.666	0.087	0.026	18.7
623	1.646	0.071	0.021	8.7
1246	1.650	0.053	0.014	4.4

(f) Hybrid (direct/thick boundary accurate)

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
10	50.346	2.482	16.451	556.9
20	17.216	1.112	3.084	464.5
39	4.824	0.457	0.389	294.7
78	2.868	0.228	0.084	169.7
156	3.416	0.166	0.063	93.8
312	3.636	0.145	0.066	52.6
623	3.401	0.131	0.059	27.2
1246	3.191	0.114	0.051	13.8

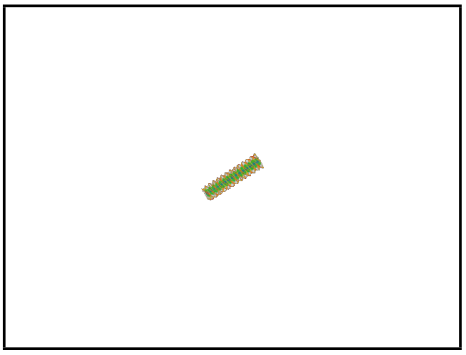
max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
10	50.346	2.472	16.393	346.0
20	17.393	1.098	3.024	202.2
39	5.030	0.436	0.364	113.1
78	2.003	0.195	0.064	59.2
156	1.793	0.115	0.032	30.1
312	1.710	0.082	0.024	17.3
623	1.690	0.065	0.018	8.2
1246	1.695	0.046	0.011	4.1

Table 5.1: Statistics for Case A: Hybrid switchover. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. The FPS column shows the performance in frames per second while rendering on an NVIDIA GeForce GTX 680 GPU, and is marked with a grey background when both of the threshold requirements are met.

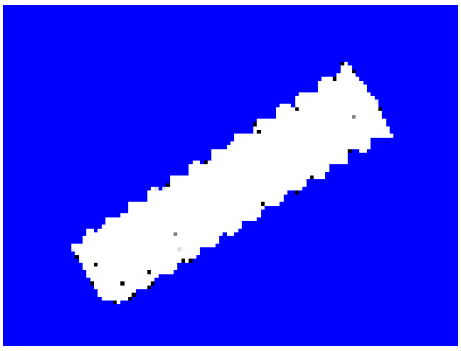


(a) Max. (b) Mean.

Figure 5.8: Graphs of the color differences for different methods in Case A: Hybrid switchover.



(a) Screen capture of the full 640x480 OpenGL viewport.



(b) Hybrid ratio for the thin method. The screen capture has been cropped and scaled. Pixels are dark colored for rays where the distance between the in intersection and the out intersection is smaller than the sampling distance.

Figure 5.9: Visualization of the reference solution and the hybrid ratio for Case B: Far.

5.5 Case B: Far

In this case, the twisted bar model is placed further from the camera. A screen capture of the OpenGL viewport is shown in figure 5.9a. The purpose of this case is to show an example where the alternative ray-casting methods can visualize the isogeometric model with a noticeable improved frame rate compared to the direct method. The model is placed at a distance from the camera where the geometric criterion is always met, see figure 5.9b for an illustration of the hybrid ratio. This means that all samples from the voxel grid should be pixel accurate, and it is expected that the visual accuracy will be within the thresholds.

The results are shown in table 5.2. All the methods except the voxelized method meet the threshold requirements on the fourth row. The performance is noticeably improved for the boundary accurate methods, especially for the thin method, compared to the direct method. The thick method does not achieve the same level of performance, but has slightly better visual accuracy than the thin method. Since the geometric criterion is always met, the hybrid methods do not improve the visual accuracy in any way, compared to their non-hybrid counterparts. As expected, they do have a slight drop in performance due to the additional code needed to check the geometric criterion for each sample point.

For this case the thin method is the best alternative to the direct method, having good visual accuracy and significantly improved performance. Since the geometric criterion is always met, there is also pixel accuracy for all the samples from the voxel grid. Using the hybrid method leads to a slight drop in performance, but would also assure that samples are pixel accurate if the model is moved closer to the camera. In this example, the thin hybrid method has about 6 to 9 times more frames per

second depending on the sampling distance. The thick hybrid method is also a good alternative, with about 2 to 3 times more frames per second depending on the sampling distance.

(a) Direct method.

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
11	47.421	2.574	17.562	438.1
20	23.578	1.136	3.476	251.7
40	5.991	0.436	0.392	137.6
79	2.725	0.191	0.064	75.2
158	1.509	0.108	0.030	38.9
315	1.096	0.073	0.021	21.6
630	1.030	0.055	0.015	10.3
1260	0.597	0.034	0.006	5.1

(b) Voxelized method.

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
9	69.935	21.283	271.742	2783.2
17	68.929	20.828	261.404	2322.7
33	68.610	20.298	259.387	1777.2
65	68.529	20.284	260.961	1218.8
129	70.255	20.466	267.287	770.2
258	73.742	20.591	271.339	452.5
515	73.976	20.712	276.327	263.7
1029	73.978	20.796	278.968	143.9

(c) Thin boundary accurate

(d) Hybrid (direct/thin boundary accurate)

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
11	47.428	2.601	17.698	977.0
20	23.783	1.165	3.533	864.1
40	6.662	0.480	0.438	699.3
79	3.002	0.254	0.114	511.5
158	2.952	0.206	0.095	331.0
315	3.149	0.190	0.097	194.8
630	3.519	0.178	0.097	108.5
1260	3.524	0.169	0.097	58.4

(e) Thick boundary accurate method.

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
11	47.428	2.601	17.698	974.0
20	23.783	1.165	3.533	840.2
40	6.662	0.480	0.438	657.7
79	3.002	0.254	0.114	466.6
158	2.952	0.206	0.095	292.4
315	3.149	0.190	0.097	169.5
630	3.519	0.178	0.097	94.1
1260	3.524	0.169	0.097	49.4

(f) Hybrid (direct/thick boundary accurate)

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
11	47.428	2.587	17.634	702.3
20	23.783	1.149	3.508	513.5
40	6.662	0.462	0.418	312.1
79	2.851	0.231	0.093	191.8
158	2.952	0.171	0.072	109.9
315	3.149	0.148	0.073	63.1
630	3.519	0.133	0.066	31.3
1260	3.299	0.116	0.062	16.7

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
11	47.428	2.587	17.634	672.7
20	23.783	1.149	3.508	507.5
40	6.662	0.462	0.418	303.4
79	2.851	0.231	0.093	191.8
158	2.952	0.171	0.072	114.7
315	3.149	0.148	0.073	66.5
630	3.519	0.133	0.066	34.1
1260	3.299	0.116	0.062	17.7

Table 5.2: Statistics for Case B: Far. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. The FPS column shows the performance in frames per second while rendering on an NVIDIA GeForce GTX 680 GPU, and is marked with a grey background when both of the threshold requirements are met.

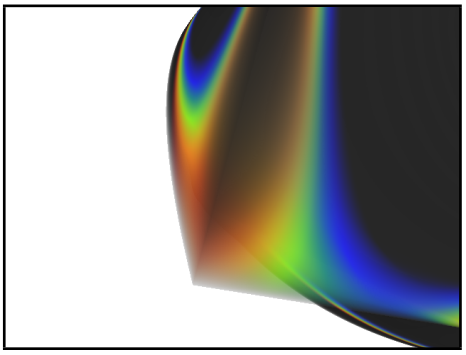


Figure 5.10: Visualization of Case C: Visual artifacts. The image shows a screen capture of the full 640x480 OpenGL viewport while rendering the reference solution. The geometric criterion is never met, meaning that the whole model is too close to the camera to use the voxel grid for any sample point.

5.6 Case C: Visual artifacts

In the previous example, the thin method was a better alternative to the direct method than the thick method. In section 5.2.1 however, it was shown that the thin method could generate results with visual artifacts. An example where this is the case will be analyzed here. As was explained in section 5.2.1, this could happen when choosing specific view angles, combined with tailored transfer functions. Visualization of the reference solution can be seen in figure 5.10. The transfer function used in this case is largely opaque and black for most scalar values, but with small peaks where color is introduced. This means that if large sampling distances are used while ray-casting, these peaks may be completely skipped, generating a result with low visual accuracy.

The color difference results are shown in table 5.3. The first results here have poor visual accuracy, even for the direct method. This is due to sampling with a small sampling distance, and the transfer function containing peaks. Since the transfer function also is largely opaque, ray-casting mostly stops with early ray termination. Halving the sampling distance does therefore not always increase the maximum number of sample points. The first three rows of the table for the direct method have the same number for the maximum number of sample points. However, the average number of sample points do change, which is the reason why the measured frames per second changes.

Once again the thin method drastically improves the visual accuracy compared to the voxelized method. It does not satisfy the threshold requirements for any of the sampling distances used here however, but this does not come as a surprise since the geometric criterion is not met for any of the sample points. This is also the explanation why all of the hybrid methods generate the exact same visual results as the direct method. The hybrid methods do however suffer a slight penalty to performance compared to the direct method, due to having to test the geometric criterion for each internal sample point.

The thick method has once again the best visual accuracy of the boundary accurate methods. The visual accuracy is actually within the set thresholds for both the maximum and the average color difference for the lowest sampling distances used here. However, it should be noted that since the geometric criterion is not met, there is no guarantee that the sampling is pixel accurate, which means that the result could be above the thresholds for other examples. Since the transfer function is largely opaque, it means that the model is mostly sampled close to the boundaries facing the camera. This means that when ray-casting with the thick method, a large share of the sample points will be from the spline model, reducing the overall performance compared to the thin method. The measured frames per second also drops faster when halving the sampling distance compared to the thin method, since a bigger share of the sampling points come from the spline model.

Both of the hybrid methods will in this case generate results with the same visual accuracy as the direct method, although with a slight performance penalty. For the thick hybrid version, there is a larger performance penalty than for the thin method, due to having a more complicated test for each sample point in the shader in addition to testing the geometric criterion. In this example the thin hybrid method has on average a 2% drop in frames per second, while the thick hybrid method has roughly 8-15% drop in performance compared to the direct method.

(a) Direct method.

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
5	56.893	8.510	104.362	208.4
5	49.037	7.243	84.485	210.2
5	29.127	4.559	31.176	168.5
6	10.440	1.912	4.101	138.7
8	2.990	0.567	0.237	102.2
12	0.817	0.152	0.017	66.0
21	0.288	0.057	0.002	38.4
39	0.261	0.023	0.000	20.4

(b) Voxelized method.

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
10	107.733	16.984	473.553	1079.9
19	106.170	15.426	374.003	1009.0
37	107.600	14.753	334.957	849.2
73	105.742	15.278	376.315	703.3
146	106.952	14.647	349.405	525.0
291	107.055	12.855	291.048	349.7
581	89.510	11.869	254.333	221.6
1161	88.594	11.281	228.135	127.7

(c) Thin boundary accurate

(d) Hybrid (direct/thin boundary accurate)

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
5	56.893	8.511	104.314	253.7
5	53.606	7.277	85.453	255.2
7	45.581	4.664	33.246	248.3
11	22.729	2.150	5.743	241.3
19	12.051	1.156	1.965	230.8
30	11.856	0.925	1.471	212.6
57	11.856	0.779	1.131	179.5
121	12.323	0.702	0.964	139.0

(e) Thick boundary accurate method.

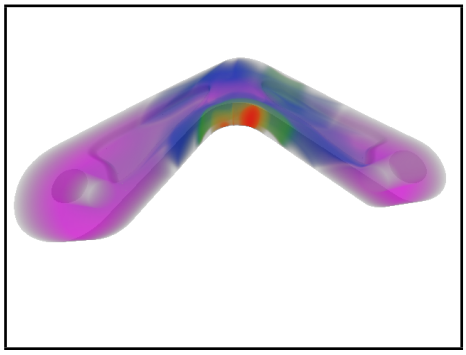
max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
5	56.893	8.510	104.362	204.6
5	49.037	7.243	84.485	206.6
5	29.127	4.559	31.176	167.0
6	10.440	1.912	4.101	136.6
8	2.990	0.567	0.237	101.5
12	0.817	0.152	0.017	65.5
21	0.288	0.057	0.002	38.1
39	0.261	0.023	0.000	20.4

(f) Hybrid (direct/thick boundary accurate)

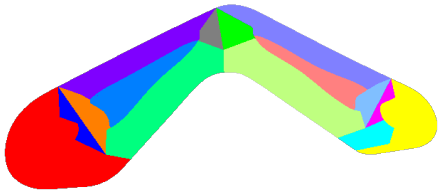
max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
5	56.893	8.512	104.359	241.0
5	49.127	7.256	84.692	241.3
5	29.148	4.609	31.921	202.6
6	12.455	2.050	4.899	168.8
8	6.787	0.818	0.798	113.5
12	4.124	0.451	0.306	68.5
21	3.451	0.377	0.250	39.1
39	3.315	0.355	0.245	22.2

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
5	56.893	8.510	104.362	192.4
5	49.037	7.243	84.485	193.2
5	29.127	4.559	31.176	152.5
6	10.440	1.912	4.101	123.0
8	2.990	0.567	0.237	89.2
12	0.817	0.152	0.017	57.1
21	0.288	0.057	0.002	32.6
39	0.261	0.023	0.000	17.5

Table 5.3: Statistics for Case C: Visual artifacts. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. The FPS column shows the performance in frames per second while rendering on an NVIDIA GeForce GTX 680 GPU, and is marked with a grey background when both of the threshold requirements are met.



(a) Screen capture of the full 640x480 OpenGL viewport.



(b) Visualization of the block structure of the model. A different color has been used on each volume block.

Figure 5.11: Visualization of the model used in Case D: Multiple volume blocks.

5.7 Case D: Multiple volume blocks

All the cases presented so far, including those on the 2D prototype, have been on models consisting of only one volume block. In section 4.1.2 it was argued that solving the ray-casting problem for multiple volume blocks would be similar to solving the ray-casting problem for one volume block multiple times. Here the methods are demonstrated on a more industrially relevant model consisting of 15 volume blocks. The model itself is a demonstration from the TERRIFIC[1] project, and a visualization of the model is shown in figure 5.11. It shows von Mises stress for the bent object, which is a common approach for checking if a design can withstand a given load. This case uses the same view-matrix and transfer function as the example in [8] on the same model.

For this model the direct method needs a small sampling distance to meet the threshold requirements. Both of the boundary accurate methods also fulfill the requirements for this sampling distance, but with better performance. Once again the difference in visual accuracy between these two is small, but the performance difference is more noticeable. Since the whole model is close to the camera, the geometric criterion is never met, and the color difference results for the hybrid methods are identical to the direct method. They do however suffer from slight performance penalties due to the extra test for checking the geometric criterion.

[1] http://www.terrific-project.eu/

(a) Direct method.

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
8	47.788	1.783	11.929	49.8
8	47.788	1.770	11.844	45.7
9	28.672	1.058	4.440	30.4
11	14.071	0.508	1.123	18.8
16	10.572	0.203	0.258	10.6
27	5.852	0.074	0.056	5.7
50	3.347	0.024	0.010	2.9
94	1.129	0.007	0.001	1.5

(b) Voxelized method.

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
8	66.941	5.811	83.015	1104.9
16	51.783	5.876	83.894	899.1
32	51.684	6.092	91.913	651.3
64	51.989	6.231	98.587	429.1
128	53.753	6.346	103.320	247.5
255	54.456	6.418	105.718	134.3
509	54.851	6.458	106.976	85.4
1018	55.872	6.479	107.651	48.0

(c) Thin boundary accurate

(d) Hybrid (direct/thin boundary accurate)

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
8	47.788	1.783	11.929	46.4
8	47.788	1.770	11.847	46.4
9	32.093	1.059	4.442	45.6
11	16.131	0.509	1.128	44.7
16	10.384	0.203	0.259	43.5
27	5.842	0.076	0.058	41.4
50	3.002	0.030	0.013	38.2
94	2.474	0.016	0.005	33.4

(e) Thick boundary accurate method.

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
8	47.788	1.783	11.929	46.9
8	47.788	1.770	11.844	42.7
9	28.672	1.058	4.440	28.4
11	14.071	0.508	1.123	17.6
16	10.572	0.203	0.258	9.9
27	5.852	0.074	0.056	5.3
50	3.347	0.024	0.010	2.7
94	1.129	0.007	0.001	1.4

(f) Hybrid (direct/thick boundary accurate)

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
8	47.788	1.783	11.929	47.7
8	47.788	1.770	11.845	46.6
9	32.093	1.058	4.439	42.5
11	14.061	0.508	1.123	37.2
16	10.384	0.202	0.258	29.7
27	5.842	0.075	0.056	20.9
50	3.004	0.028	0.011	12.7
94	1.377	0.013	0.002	6.4

max(#S)	max(ΔE)	mean(ΔE)	var(ΔE)	FPS
8	47.788	1.783	11.929	47.2
8	47.788	1.770	11.844	43.1
9	28.672	1.058	4.440	28.7
11	14.071	0.508	1.123	17.8
16	10.572	0.203	0.258	9.8
27	5.852	0.074	0.056	5.3
50	3.347	0.024	0.010	2.8
94	1.129	0.007	0.001	1.4

Table 5.4: Statistics for Case D: Multiple volume blocks. max(#S) indicates the maximum number of sample points along a ray. Color differences have been calculated between the results from each method and the reference solution using the CIEDE2000 algorithm. The columns with max/mean/var(ΔE) shows the maximum, mean and variance of the color differences respectively. max(ΔE) and mean(ΔE) values that satisfy the threshold requirements are marked with a grey background. The FPS column shows the performance in frames per second while rendering on an NVIDIA GeForce GTX 680 GPU, and is marked with a grey background when both of the threshold requirements are met.

Chapter 6

Discussion

6.1 General discussion

The research goal for this thesis was to explore if there were ways of improving performance when visualizing isogeometric models. During planning some different approaches were considered, but in the end it was argued that creating reduced models based on voxel grids would have the most potential. Based on this different ray-casting methods were proposed, and compared against an existing solution.

By using the boundaries of the spline model combined with internal sample points from a voxelized representation, visual accuracy was shown to improve drastically compared to using only a voxelized model. This is due to the problem a voxel grid has of representing the boundaries of a model. It was however not enough to satisfy the requirements for visual accuracy at all viewing distances alone. Hybrid methods were then introduced that would in theory choose the voxelized model where it would be "good enough", and the spline model elsewhere. In 2D, the boundary accurate hybrid method showed results with good visual accuracy. The voxelized hybrid method showed consistently worse results, and could in some cases go above the thresholds after the hybrid switchover. This was again likely due to the boundary problem with the voxelized model. Since the boundary accurate hybrid method had substantially better visual accuracy with a relatively small performance penalty, the voxelized hybrid method was not considered to be a good alternative for ray-casting isogeometric models.

When implementing the thin boundary accurate method in 3D, it was discovered that it could in some cases generate results with visual artifacts. The thick method was therefore proposed as an alternative to prevent this. Although this method would cause a further drop in performance, visual artifacts were less visible and the overall visual accuracy was also improved.

6.2 Conclusion

In this thesis it has been shown that it is possible to combine a spline model with a simplified voxelized version to visualize isogeometric models with higher performance, and with good visual accuracy. To have a good assurance of the visual accuracy, a hybrid method should preferably be used over a non-hybrid method. The hybrid methods do however have a slight performance penalty compared to the direct method when the whole isogeometric model is close to the camera, but when the model is beyond a certain distance, the performance is drastically improved.

From the 3D implementation, the two proposed boundary accurate methods generated different results. The thin method had consistently the best performance, but never the best visual accuracy. The main problem with this method however, is that it can generate results with visual artifacts for certain viewing angles and specific transfer functions. In conclusion, the most ideal boundary accurate method would therefore be the thick method. The hybrid version gives a good assurance of high visual accuracy at all viewing distances.

6.3 Future work

The ray-casting methods presented in this thesis works well for the general case, but typically an isogeometric model is largely displayed with homogeneous or transparent colors. When a ray traverses through such areas, there is an opportunity to increase the sampling distance. Currently the implementation uses a fixed step length, but having an adaptive step length could decrease the amount of samples, which would in turn increase performance. This is applicable for both the direct method and for methods based on using a voxel grid. However, how to determine the step length is not immediately obvious. Earlier work has suggested to use a so-called *importance volume*, which could be implemented as a lookup texture for sampling distances. This could be generated based on the spline functions defining the geometry and the scalar field. Basing it on the transfer function as well could in turn allow for potentially higher rendering performance, although the importance volume must then be recreated as soon as the transfer function is changed.

The hybrid methods presented here have been based on a geometric criterion. It assures pixel accuracy by ensuring that the distance between two diagonally adjacent voxels can fit into the pixel frustum. For volumes that are largely homogeneous or transparent, this requirement is needlessly strict. In those cases, doing a lookup in the voxel grid could be sufficiently accurate, even though the geometric criterion is not met. A hybrid method based on the adaptive sampling distance from an importance volume could therefore take more opportunities at sampling from the voxel grid.

Isogeometric models with large homogeneous areas would also in turn lead to a generated voxel grid which contains many individual voxels with roughly the same value. This is not an efficient use of memory space on the GPU. Using sparse textures, which was recently introduced in OpenGL, could reduce the amount of duplicate

information in a texture. With these there are different levels of detail in the same texture, and a texel at any level can be defined to be at the finest level of detail. It is thus possible to cover a large rectilinear homogeneous area with one texel in the texture.

The main problem with the voxelized models are the representation of the boundaries of isogeometric models. In this thesis voxelization leads to voxels of two possible states, either indicated to have a defined value or not. This leads to serrated edge representations. Chen [3] described a technique to more accurately represent curved boundaries with voxel grids. Here the boundaries are used to extrapolate values to the first voxels outside the volume covered by the model. Sampling at these voxels would indicate that they are non-resident, but due to linear interpolation, the spline boundaries are represented at points where the indicator passes a certain value, for instance 0. Even though this is likely to create better representations of the boundaries in the voxel grid, the boundary accurate methods will always be more visually accurate, since they use the spline models directly. But it is possible that with a more accurate voxel grid, the thin method could achieve better results without the problem of visual artifacts, making the thick method obsolete.

Bibliography

[1] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-time rendering*. CRC Press, 2008.

[2] Jim F. Blinn. "Compositing. 1. Theory." In: *IEEE Computer Graphics and Applications* 14.5 (Sept. 1994), pp. 83–87. ISSN: 0272-1716. DOI: 10.1109/38.310740.

[3] Yuan Chen. *Techniques for Three-dimensional Scalar and Vector Field Visualization with Error Evaluation*. ProQuest, 2009.

[4] Cyril Crassin et al. "Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering." In: *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. ACM. 2009, pp. 15–22.

[5] Carl de Boor. "A practical guide to splines." In: *Mathematics of Computation* (1978).

[6] Matthias Eck and Jan Hadenfeld. "Knot removal for B-spline curves." In: *Computer Aided Geometric Design* 12.3 (1995), pp. 259–282.

[7] Klaus Engel et al. *Real-time volume graphics*. AK Peters, Ltd., 2006.

[8] Franz G. Fuchs and Jon M. Hjelmervik. "Interactive Isogeometric Volume Visualization with Pixel-Accurate Geometry." In: *IEEE Transactions on Visualization and Computer Graphics* 22.2 (Feb. 2016), pp. 1102–1114. ISSN: 1077-2626.

[9] John R. Higgins. *Sampling Theory in Fourier and Signal Analysis: Foundations*. Oxford University Press on Demand, 1996.

[10] Jon M. Hjelmervik. "Direct Pixel-Accurate Rendering of Smooth Surfaces." In: *Mathematical Methods for Curves and Surfaces: 8th International Conference, MMCS 2012, Oslo, Norway, June 28 – July 3, 2012, Revised Selected Papers*. Springer Berlin Heidelberg, 2014, pp. 238–247. ISBN: 978-3-642-54382-1.

[11] Thomas J. R. Hughes, John A. Cottrell, and Yuri Bazilevs. "Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement." In: *Computer methods in applied mechanics and engineering* 194.39 (2005), pp. 4135–4195.

[12] Martin Kraus and Kai Bürger. "Interpolating and Downsampling RGBA Volume Data." In: *VMV*. 2008, pp. 323–332.

[13] Eric C. La Mar, Bernd Hamann, and Kenneth I. Joy. "Multiresolution Techniques for Interactive Texture-based Volume Visualization." In: *Proceedings of the 10th IEEE Visualization 1999 Conference (VIS '99)*. VISUALIZATION '99. Washington, DC, USA: IEEE Computer Society, 1999. ISBN: 0-7803-5897-X.

[14] CIE Commision Internationale de l'Éclairage. *Improvement to industrial colour-difference evaluation*. Tech. rep. CIE Technical Report, Publication 142.(CIE Central Bureau: Vienna), 2001.

[15] William E. Lorensen and Harvey E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm." In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), pp. 163–169. ISSN: 0097-8930. DOI: 10.1145/37402.37422.

[16] David Luebke et al. *Level of Detail for 3D graphics*. Morgan Kaufmann, 2003.

[17] William Martin and Elaine Cohen. "Representation and Extraction of Volumetric Attributes Using Trivariate Splines: A Mathematical Framework." In: *Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications*. SMA '01. Ann Arbor, Michigan, USA: ACM, 2001, pp. 234–240. ISBN: 1-58113-366-9. DOI: 10.1145/376957.376984.

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

[19] Gaurav Sharma and Raja Bala. *Digital color imaging handbook*. CRC press, 2002.

[20] Manfred Weiler et al. "Level-of-detail Volume Rendering via 3D Textures." In: *Proceedings of the 2000 IEEE Symposium on Volume Visualization*. VVS '00. Salt Lake City, Utah, USA: ACM, 2000, pp. 7–13. ISBN: 1-58113-308-1. DOI: 10.1145/353888.353889.

[21] Lance Williams. "Pyramidal parametrics." In: *ACM Siggraph Computer Graphics*. Vol. 17. 3. ACM. 1983, pp. 1–11.

[22] Orion Wilson, Allen Van Gelder, and Jane Wilhelms. *Direct Volume Rendering Via 3D Textures*. Tech. rep. 1994.

[23] Young In Yeo, Lihan Bin, and Jörg Peters. "Efficient pixel-accurate rendering of curved surfaces." In: *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM. 2012, pp. 165–174.

[24] Jun-Hai Yong et al. "Degree reduction of B-spline curves." In: *Computer Aided Geometric Design* 18.2 (2001), pp. 117–127.

Figures

[25] Sean Barrett. *Sparse Virtual Textures*. Digital image. 2008. URL: http : / / silverspaceship.com/src/svt (visited on Apr. 27, 2015).

[26] Project CARS. *LOD - Level Of Detail*. Digital image. 2013. URL: http : / / en . pcars . shoutwiki.com/wiki/LOD_-_Level_Of_Detail (visited on Apr. 29, 2015).

[27] Christoph Rezk-Salama and Peter Hastreiter. *Pre- and Post-Classification*. Digital image. 2013. URL: http : / / schorsch . efi . fh - nuernberg . de / roettger / index . php / VolumeRendering/Pre-AndPost-Classification (visited on May 13, 2015).

[28] Henning Wenke and Oliver Vornberger. *Volume Rendering Integral*. Digital image. 2010. URL: http : / / media2mult . uos . de / pmwiki / fields / cg - II - 09 / index . php ? n = VolumeRendering.VolumeRenderingIntegral (visited on May 13, 2015).