BILAPLACIAN RECONSTRUCTION OF POINT CLOUDS

by

ORIOL GIRALT GARCIA

MASTER IN INNOVATION AND RESEARCH IN INFORMATICS
IN
COMPUTER GRAPHICS AND VIRTUAL REALITY

Facultat de Informàtica de Barcelona
Universitat Politècnica de Catalunya
Director: Antonio Chica Calaf - Computer Systems
Barcelona, Spain

25 January, 2022

Candidate's Declaration

This is to certify that the work presented in this thesis entitled, "Bilaplacian reconstruction of point clouds", is the outcome of the research carried out by ORIOL GIRALT GARCIA under the supervision of Dr. TONI CHICA CALAF, Associate Professor of,Facultat de Informàtica de Barcelona (FIB), Universitat Politècnica de Catalunya (UPC).

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma, or other qualifications.

Signature of the Candidate

ORIOL GIRALT GARCIA

Contents

Acknowledgement

First and foremost I am extremely grateful to my supervisor, Prof. and Dr. Toni Chica for their invaluable advice, continuous support, and patience during my Masters Thesis study. Their immense knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life. Without his tremendous understanding and cheering in the past few years, it would be impossible for me to complete my study.

Finally, my deep and sincere gratitude to my family for their continuous and unparalleled love, help and support. I am forever indebted to my parents for giving me the opportunities and experiences that have made me who I am.

Abstract

A key process of the geometry processing pipeline is the reconstruction of surfaces from point clouds. The traditional problem addressed by surface reconstruction is to recover the digital representation of the shape that has been inputted, where the data could potentially contain a wide variety of drawbacks. The goal of this thesis would be to test the Bilaplacian Smoothness in order to enforce the smooth prior to the surface reconstruction. By considering our thesis goal we will build an application that not only will solve different sparse linear systems of equations using different possible methods for position, normal, and smoothness equation's constraints but also will make use of more complex and effective surface reconstruction solving techniques such as the multigrid or quadree reconstruction.

Chapter 1

Introduction

1.1 Introduction to Point Cloud Surface Reconstruction

The modeling, recognition, and analysis of the elements that encompass us is a long-term goal in the Computer Graphics field. One of the main objectives is obtaining a digital representation of input data that possibly contains relevant information if we take a look at it. Surface reconstruction is concerned with recovering such information, where the basic problem is to capture a point cloud that could not necessarily be a sample of the real world and reconstruct as much information as possible concerning the inputted data.

Surface reconstruction came to importance primarily as a result of the ability to acquire point clouds thanks to many different techniques and hence there are very close ties between how the data is acquired and the method used to reconstruct it. Early on, these techniques ranged from active methods such as optical laser-based range scanners, structured light scanners, and LiDAR scanners, as well as passive methods such as multi-view stereo scanners. Recently, a recent trend has seen the massive proliferation of point clouds from real-time domestic scanners such as the Microsoft Kinect. As the diversity, ease of use, and popularity of 3D acquisition methods continues to increase, also does the need for the development of new surface reconstruction techniques.

Most acquisition methods tend to produce point clouds containing a variety of properties and imperfections that implies significant challenges for surface reconstruction methods. These properties, in conjunction with the scanned shape's complexity, conditions the kind of reconstruction methods that will be faster or even optimal. The diverse set of techniques used will not be constrained to any kind of shape or form and should be able to generate a satisfying solution depending on the quality of the point cloud.

For our research, we will receive as input a document with a point cloud list and its respective associated normals. As for output, we want to compute the resulting implicit function of the point cloud list which we will use to extract its polygonal mesh pattern through algorithms such as the Marching Cubes. We will obtain those solutions by forcing the implicit function to pass through the point cloud's relative position with a gradient close to its normal value and supposing that the 0-isosurface has to be soft. In our implementation, we will stick to the 2-dimensional coordinates in order to implement all our presented algorithms.

The rich quantity of algorithms used either for the equation system construction or the solver would provide a strong variety of possibilities, making use of the main aptitudes of each possible combination and providing a detailed taxonomy of existing methods' efficiency and performance.

Chapter 2

Related Work

There are many papers related to the point cloud reconstruction; some of them focus on the Surface or Volumetric Smoothness while others prioritize the Geometric Primitives or the Global Regularity of the output surface. In our case, we want to focus on the efficiency and performance of the reconstruction in terms of execution time.

One of the most notorious papers which are related to our approaches is the SSD: Smooth signed distance surface reconstruction [1]. In this paper, the authors propose a new variational formulation for the problem of reconstructing a watertight surface defined by an implicit equation from a finite set of oriented points. As in the Poisson Surface Reconstruction approach, discretizations of the continuous formulation are reduced to the solution of sparse linear systems of equations. But rather than forcing the implicit function to approximate the indicator function of the volume bounded by the implicit surface, in this formulation the implicit function is forced to be a smooth approximation of the signed distance function to the surface. Since an indicator function is discontinuous, its gradient does not exist exactly where it needs to be compared with the normal vector data. The smooth signed distance has an approximate unit slope in the neighborhood of the data points. As a result, the normal vector data can be incorporated directly into the energy function without implicit function smoothing.

Strictly related to the previously mentioned one we have the Smooth Signed Distance Surface Reconstruction and Applications [2] paper in which the authors present a similar formulation where the implicit function is forced to be a smooth approximation of the signed distance function to the surface. The formulation allows for a number of different efficient discretizations, reduces to a finite-dimensional least squares problem for all linearly parameterized families of functions that do not require the specification of boundary conditions, and it is particularly good at extrapolating missing and/or irregularly sampled data. The resulting algorithms are significantly simpler and easier to implement than alternative methods. The implementation of the authors is based on a primal-graph octree-based hybrid finite element-finite difference discretization, and the Dual Marching Cubes isosurface extraction algorithm is very efficient and produces high quality crack-free adaptive manifold polygon meshes. After the geometry and topology are reconstructed, the color information from the points is smoothly extrapolated to the surface by solving a second variational problem which also reduces to a finite-dimensional least-squares problem.

Another relevant paper is the Reconstruction of Solid Models from Oriented Point Sets [3] which presents a novel approach to the surface reconstruction problem that takes as its input an oriented point set and returns a solid, water-tight model. The idea of this approach is to compute the characteristic function of the model (the function that is equal to one inside the model and zero outside of it). Specifically, it provides an efficient method for computing the Fourier coefficients of the characteristic function using only the surface samples and normals, computing the inverse Fourier transform to get back the characteristic function and using isosurface techniques to extract the boundary of the solid model. The advantage of our approach is that it provides an automatic, simple, and efficient method for computing the solid model represented by a point set without requiring the establishment of adjacency relations between samples or iteratively solving large systems of linear equations.

Relating to the most recent stage of development we can mention the State of the Art in Surface Reconstruction from Point Clouds [4]. This survey focuses on those relating to the reconstruction from point clouds of static objects and scenes acquired through 3D scanners, wherein the point cloud contains a considerable level of imperfection. Furthermore, they concentrate on methods that approximate the input point cloud.

Given a surface with boundaries, there exist many methods for painting and surface completion for handling missing data. Though one may use such approaches for reconstruction by first reconstructing a surface with boundary from a point cloud, this can be quite challenging given other imperfections in the data. These methods are not covered in this survey and we refer the reader to the recent survey on surface completion [5].

The paper Evaluating surface reconstruction of open scenes [6] addresses the evaluation of algorithms that are meant to reconstruct a watertight surface from a point cloud acquired on an open scene. The objective of the authors is to set a rigorous protocol measuring the quality of the reconstruction and to propose a quality metric that is informative with respect to the various qualities that such an algorithm should have, and in particular it’s capacity to interpolate and extrapolate accurately. In addition, they use publicly available data and our implementation is open-source. The authors argue that a rigorous evaluation of surface reconstruction of open scenes needs to be performed on synthetic data where a perfect continuous ground truth surface is available. Moreover, the authors developed their own LiDAR simulator of which they give a description in the present paper.

One of the main problems we will deal with is going to be the point cloud data. The point cloud is considered an unstructured form when it does not contain any connectivity information between adjacent points and structure information. In the paper, A survey on surface reconstruction techniques for structured and unstructured data [7] various types of surface reconstruction techniques are proposed to overcome the problems of point cloud and the limitations of existing techniques. Besides, the authors make use of soft computing techniques to enhance the performance and overcome the downsides of existing techniques. Therefore, the objective of this paper is to conduct a survey of the existing techniques in the surface reconstruction on structured or unstructured data. Generally, the authors will only focus on the interpolation and approximation techniques, learning-based techniques, and soft computing techniques. The outcome of this paper can be used to help the readers in understanding and finding suitable surface reconstruction techniques in representing the objects and solving their case studies.

Another paper that is worth being mentioned is the Poisson Surface Reconstruction [8] In which a novel approach expresses surface reconstruction as the solution to a Poisson equation. The main difference of this approach compared to our implementation is that they don't take into account the point positions on the equations, leading to producing a mesh with the same form but not necessarily the same size. This was improved on the Screened Poisson Surface Reconstruction [9] where they extend the previously mentioned technique to explicitly incorporate the points as interpolation constraints.

Furthermore, the Robust Poisson Surface Reconstruction paper [10] proposes a method to reconstruct surfaces from oriented point clouds with non-uniform sampling and noise by formulating the problem as a convex minimization that reconstructs the indicator function of the surface's interior. Compared to previous models, the presented reconstruction is robust to noise and outliers because it substitutes the least-squares fidelity term by a robust Huber penalty [11]. This allows to recover sharp corners and avoids the shrinking bias of least squares. The authors choose an implicit parametrization to reconstruct surfaces of unknown topology and close large gaps in the point cloud. For an efficient representation, they approximate the implicit function by a hierarchy of locally supported basis elements adapted to the geometry of the surface. The hierarchical structure of the basis speeds up the minimization and efficiently represents clustered data. They also advocate for convex optimization, instead of isogeometric finite-element techniques, to efficiently solve the minimization and allow for non-differentiable functionals.

The Progressive Discrete Domains for Implicit Surface Reconstruction [12] paper propose a progressive coarse-to-fine approach that jointly refines the implicit function and its representation domain, through iterating solver, optimization, and refinement steps. Many global implicit surface reconstruction algorithms formulate the problem as a volumetric energy minimization, trading data fitting for geometric regularization. As a result, the output surfaces may be located arbitrarily far away from the input samples. This breaks the strong assumption commonly used by popular octree-based and triangulation-based approaches that the output surface should be located near the input samples. As these approaches refine during a pre-process, their cells near the input samples, the implicit solver deals with a domain discretization not fully adapted to the final isosurface. There are several advantages to this approach: the discretized domain is adapted near the isosurface and optimized to improve both the solver conditioning and the quality of the output surface mesh.

Reconstruction of implicit curves and surfaces via RBF interpolation [13] paper focuses on theoretical and practical issues in using radial basis functions (RBF) for reconstructing implicit curves and surfaces from point clouds. The authors study the conditioning of the problem and give some insight on how the problem parameters and the results have to be taken in order to achieve meaningful solutions and avoid artifacts. Moreover, a strategy for decreasing the conditioning of the problem is suggested and a general framework for preconditioning and solving the problem, even for large datasets, is also provided.

For the optimized tree structure, we can refer to the paper Quad Trees: A Data Structure for Retrieval on Composite Keys [14] in which appears the keyword and concept of *Quadtree* for the first time.

Finally, for the *Multigrid* optimization, we can cite the P. Wesselling book in which presents many choices of multigrid methods with varying trade-offs between the speed of solving a single iteration and the rate of convergence with said iteration [15]. We will also make reference to the An Introduction to Algebraic Multigrid paper [3] which presents the Algebraic Multigrid (**AMG**) for solving linear systems based on multigrid principles, but in a way that only depends on the coefficients in the underlying matrix. This technique is the same multigrid technique used for the iterative solvers of the Eigen library.

Last but not least, is it worth mentioning the book Foundations of Multidimensional and Metric Data Structures [16]. This book provides a thorough treatment of multidimensional point data, object and image-based representations, intervals and small rectangles, and high-dimensional datasets, providing this way an excellent and valuable reference tool for our implementations.

Chapter 3

Point Cloud Data Description

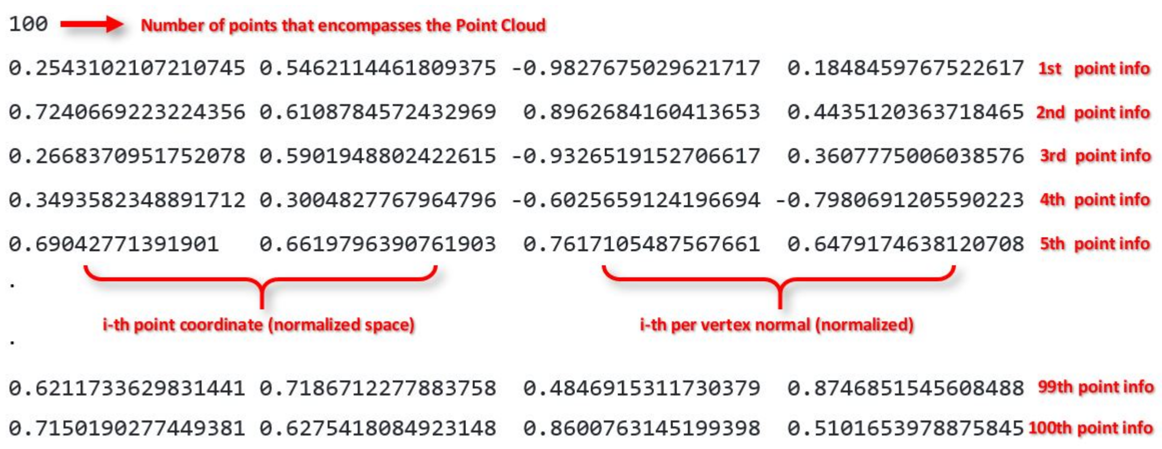
3.1 Description of the Input Data

Point Clouds can be presented in many different ways. Often reconstruction methods have different types of input requirements associated with the point cloud input. The minimum requirement would be a set of points that sample the surface. Working with the bare points, however, may have the problem of not having enough information for the reconstruction of certain types of point cloud algorithms. On the other side, other types of input can be extremely beneficial in reconstruction from challenging point clouds.

As we previously mentioned, providing just the point that samples the surface would not be enough for our purposes. We want to know more input information, in particular, the orientation of the inputted surface or the smoothness of the surface. Thus, the presence of per-vertex surface normals in our input data is a must.

Surface normals are extremely useful input for reconstruction methods. For smooth surfaces, the normal, uniquely defined at every point, in the direction perpendicular to the point's tangent space. The tangent space intuitively represents a localized surface approximation at a given point. Surface normals may be oriented, where each normal is consistently pointing inside or outside of the surface, or unoriented and lack such a direction. Normals that are oriented provide extremely useful cues for reconstruction algorithms and hence, we will use this type for our input data. Note that if certain information associated with the point cloud is not present, obtaining an orientation can be challenging.

Therefore, the input file will have the following scheme shown in figure 3.1



```
100
0.2543102107210745 0.5462114461809375 -0.9827675029621717  0.1848459767522617
0.7240669223224356 0.6108784572432969  0.8962684160413653  0.4435120363718465
0.2668370951752078 0.5901948802422615 -0.9326519152706617  0.3607775006038576
0.3493582348891712 0.3004827767964796 -0.6025659124196694 -0.7980691205590223
0.69042771391901   0.6619796390761903  0.7617105487567661  0.6479174638120708
.

.

0.6211733629831441 0.7186712277883758  0.4846915311730379  0.8746851545608488
0.7150190277449381 0.6275418084923148  0.8600763145199398  0.5101653978875845
```

Number of points that encompasses the Point Cloud
1st point info
2nd point info
3rd point info
4th point info
5th point info
i-th point coordinate (normalized space)
i-th per vertex normal (normalized)
99th point info
100th point info

Figure 3.1: Format schema that presents the Input data providing both vertex and normals values for each point of the cloud.

Note that not only the surface normals have a normalized value which is obvious but also the coordinate points are also normalized. This will be needed for solving our equation system since for obtaining a valid approximation of the surface reconstruction, the equation values that are provided to the solver must be between 0 and 1.

3.2 Point Cloud Artifacts

As we previously discussed in the previous chapter, there are many different ways of generating a point cloud input. Most of them are generated by gadgets which sometimes are not accurate at all. Those points representations could have downsides or artifacts, which are an important factor to take into account for surface reconstruction. Most of those properties will have an important impact on the behavior of reconstructing the surface. Now, we will proceed to describe some of the most important elements that may affect the performance of our reconstructions:

- Sampling density. The distribution of the points sampling the surface is referred to as sampling density. Scans typically produce a non-uniform sampling on the surface, where the sampling density spatially varies. This can be due to the distance from the shape to the scanner position, the scanner orientation, as well as the shape's geometric features Ideally, we would like to have a uniform point distribution where each point is equidistant from its neighbor. However, this is far from reality and we typically receive a nonuniform sampling of the surface. Many surface reconstruction algorithms must be able to estimate a notion of sampling density at every point, and hence the level of non-uniformity in the sampling can have a great impact on estimation accuracy.

Sampling density is important in surface reconstruction for defining a neighborhood. A neighborhood is a set of points close to a given point that captures the local geometry of the surface, such as its tangent plane. A neighborhood should be large enough so that the points sufficiently describe the local geometry, yet small enough so that local features are preserved. Under uniform sampling density, a neighborhood may be constructed at every point in the same manner. For instance, one can define a neighborhood at a point $p \in P$ via an ϵ – ball, defined as the set of points $N\epsilon(p) \subset P$ such that each $y \subset N\epsilon(p)$ satisfies $\|py\| < \epsilon$, under a single value ϵ used at all points.

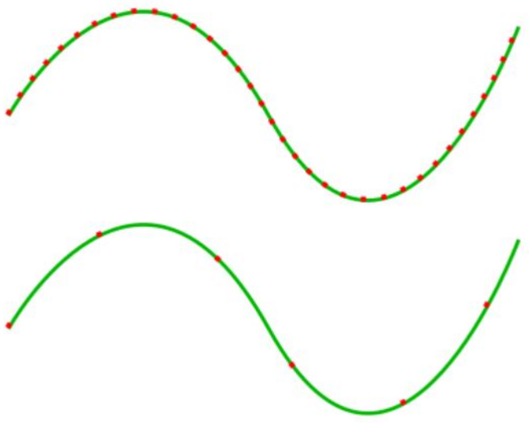


Figure 3.2: Both top and bottom images represent a point cloud. The green line represents the theoretical surface to reconstruct whereas the red dots represent the captured points. As we can observe, the top image presents a good sampling density whereas the bottom image presents a poor sampling density.

- Noise and Outliers. The noise and outliers are commonly due to structural artifacts in the acquisition process. Points that are randomly distributed near the surface are traditionally considered to be noise. The goal of surface reconstruction algorithms is to produce a surface that passes near the points without overfitting the noise. Since our iterative solver will impose smoothness on the output, which will be able to correct the surface noise. Points that are far from the true surface are classified as outliers. Note that spatially varying noise poses a significant challenge for the surface reconstruction algorithms, therefore we will use Gaussian noise during our surface reconstruction in which the values at any pair surface points are identically distributed and statistically independent and hence uncorrelated.

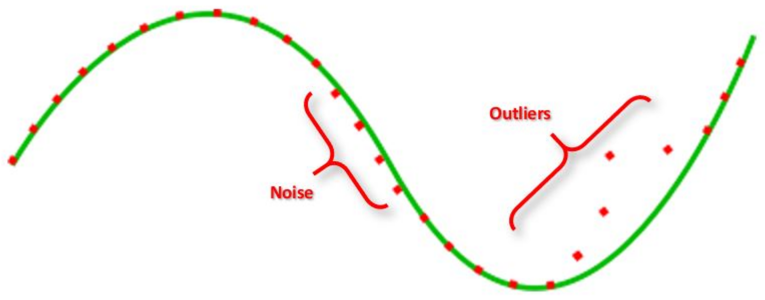


Figure 3.3: Image of a point cloud representation with noise and outliers. The green line represents the theoretical surface to reconstruct whereas the red dots represent the captured points. As we can observe, while the noise tends to follow the shape of the surfaces despite he deviates, the outliers are from being a trustworthy representation.

- Missing data. A motivating factor behind many reconstruction methods is dealing with missing data. Missing data is due to such factors as limited sensor range, high light absorption, and occlusions in the scanning process where large portions of the shape are not sampled. Although the aforementioned artifacts are continually improved upon, missing data tends to persist due to the physical constraints of the device. We note that missing data differs from nonuniform sampling, as the sampling density is zero in such regions

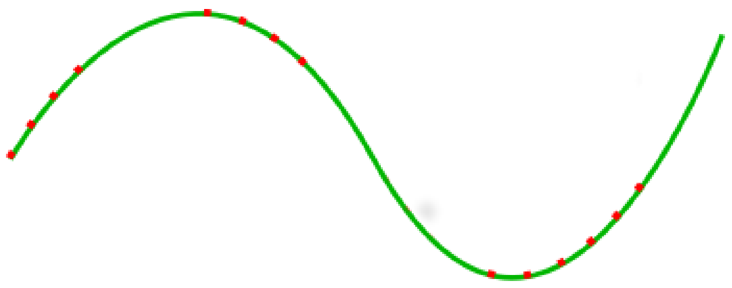


Figure 3.4: Image of a point cloud representation with missing data. The green line represents the theoretical surface to reconstruct whereas the red dots represent the captured points. As we can observe, there are some chunks where the point data is missing.

3.3 Surface Normals

Surface normals are extremely useful input for reconstruction methods. For smooth surfaces, the normal, uniquely defined at every point, in the direction perpendicular to the point's tangent space. The tangent space intuitively represents a localized surface approximation at a given point. Surface normals may be oriented, where each normal is consistently pointing inside or outside of the surface or may lack such a direction. Normals that are oriented provide extremely useful cues for reconstruction algorithms.

Normals that have consistent directions, either pointing in the inside or the outside of the surface are referred to as being oriented. Knowledge of the exterior and interior of the surface has proven extremely useful in surface reconstruction.

There are numerous ways to compute oriented normals. If the original 2D range scans are known, then the 2D lattice structure provides a way of performing consistent orientation since one always knows how to turn clockwise around a given vertex. For instance, if we denote the point in a range scan at point (x,y) as $p_{x,y}$ then one can take the normal at $p_{x,y}$ simply as the cross product between $(p_{x+1,y})$ and $(p_{x,y+1})$. If the point cloud is noisy, then this method can produce rather noisy normals, since it does not use nearby points in overlapping scans.

3.4 Surface Smoothness

The surface smoothness prior constrains the reconstructed surface to satisfy a certain level of smoothness, while ensuring the reconstruction is a close fit to the data. The most general form is the *local* smoothness, which strives for smoothness only in close proximity to the data. The output of such approaches are typically surfaces that smooth out noise associated with the acquisition while retaining boundary components where there exists insufficient sampling (or simply no sampling). Due to their generality, these methods can be applied to many shapes and acquisition devices, yet absent of additional assumptions handling severe artifacts poses a significant challenge.

In contrast, *global* smoothness (such as *Laplacian* or *Bilaplacian*) seeks higher-order smoothness, large-scale smoothness, or both. High order smoothness relates to the variation of differential properties of the surface: area, tangent plane, curvature, etc. Large-scale relates to the spatial scale where smoothness is enforced, not just near the input. It is common for these methods to focus on reconstructing individual objects, producing watertight surfaces. As a result, this limits the class of shapes to objects that can be acquired from multiple views, captured as completely as possible.

Chapter 4

Methodology

4.1 Continuous Formulation

We are concerned with the problem of reconstructing a watertight surface S defined by an implicit equation $S = \{x : f(x) = 0\}$ approximating a finite set of oriented points $D = (p_1, n_1), ..., (p_N, n_N)$, where p_i is as surface location sample, and n_i is the corresponding surface normal sample oriented towards the outside of the object. If the function is continuous, then this surface is watertight (closed). We will assume that $f(x) < 0$ inside and $f(x) > 0$ outside of the object. In both Surface Reconstruction [3] and Poisson Reconstruction [9] the implicit function is forced to be the indicator function of the volume bounded by the surface S. This function is identically equal to zero outside, to one inside, and discontinuous exactly on S, as shown in the Figure 4.3.

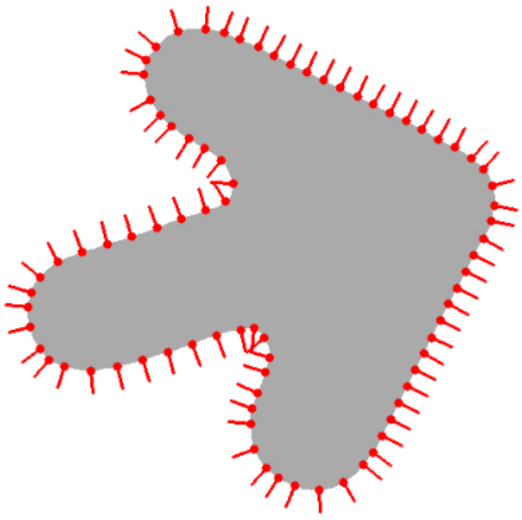


Figure 4.1: A 2D oriented point cloud as a sample of a 2D curve represented as a level set of an indicator function(gray=1,white=0).

Since the points p_i are regarded as samples of the surface S, and the normal vectors as samples of the surface normal at the corresponding points, the implicit function should satisfy $f(p_i) = 0$ and $\nabla f(p_i) = n_i$ for all the points $i = 1, ..., N$ in the data set. Interpolating schemes, which require parameterized families of functions with very large numbers of degrees of freedom, are not desirable in the presence of measurement noise. For an approximating scheme, which is what we are after, we require that these two interpolating conditions are satisfied in the least-squares sense. Moreover, we will add the smoothing prior condition. As a result, we consider the problem of minimizing the following data energy:

$$\varepsilon = \lambda_0 \varepsilon_{\mathcal{D}0}(f) + \lambda_1 \varepsilon_{\mathcal{D}1}(f) + \lambda_2 \varepsilon_{\mathcal{R}}(f) \tag{4.1}$$

λ_0 and λ_1 are positive constants used to give more or less weight to each one of the two energy terms. The normalization by the number of points is introduced to make these two parameters independent of the total number of points. λ_2 is another positive parameter. Increasing λ_2 concerning λ_0 and λ_1 produces a smoother result. To solve this problem, we will construct an equation system $Ax = b$ and will make use of an iterative solver in order to obtain a truthful reconstruction of the surface. We can divide the equations into three types; equations based on point cloud position, equations based on the surface normal, and equations based on the smoothing prior.

4.2 Point Equations

The point equations are meant to force the reconstructed isosurface to pass through the input points. Given the point cloud points, we will compute its enveloping field box of the size of the desired outputted resolution dividing the space into many nodes. For each point p we will obtain the node n_{ij} to get its relative coordinates. The interpolation of the 4 node corner points has to be $\varnothing$. This way, for each point of the point cloud $p = (x, y)$ and its respective grid node $v = \{v_{0,0}, v_{1,0}, v_{0,1}, v_{1,1}\}$, we could add the following equation to our system:

$$(1 - y)[(1 - x)v_{0,0} + xv_{1,0}] + y[(1 - x)v_{0,1} + xv_{1,1}] = 0 \tag{4.2}$$

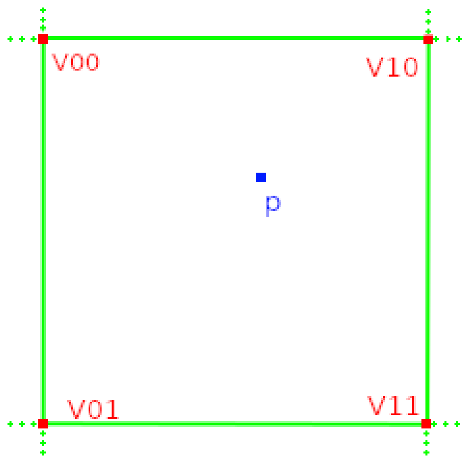


Figure 4.2: Graphical example about the point $p = (x, y)$ and the four node corners $v = \{v_{0,0}, v_{1,0}, v_{0,1}, v_{1,1}\}$ that conforms the grid cell.

4.3 Normal Equations

4.3.1 Gradient Equations

The gradient $\nabla f(x)$ of a function $f(x)$ which presents first-order continuous derivatives is a vector field normal to the level sets of the function, and in particular to the surface S. If the gradient $\nabla f(x)$ does not vanish on the surface S, then S is a manifold surface with no singular points.

Without loss of generality, we will further assume that the gradient field on the surface S is unit length, which allows us to directly compare the gradient of the function with the point cloud normal vectors. Since the points p_i are regarded as samples of the surface S, and the normal vectors as samples of the surface normal at the corresponding points, for an interpolatory scheme the implicit function should satisfy $f(p_i) = 0$ and $\nabla f(p_i) = n_i$ for all the points $i = 1, ..., N$ in the data set. This way, for each sampled point, we will have to add a point equation to our system:

$$\nabla f(p) = (\frac{\partial f(p)}{\partial x}, \frac{\partial f(p)}{\partial y}) = n \tag{4.3}$$

$$\frac{\partial f(p)}{\partial x} = n_x \tag{4.4}$$

$$\frac{\partial f(p)}{\partial y} = n_y \tag{4.5}$$

4.3.2 Sampling Equations

The sampling equation is an alternative to the gradient method. The main objective of the sampling equations is to reduce the number of equations we have in our system to obtain a smaller matrix which could improve the solver resolution time. However, with the sampling method, we are losing the trustworthiness resolution of the reconstructed surface.

In order to add the sampling equations we will perform as follows: We will add two extra point equations the same way we described in the previous chapter. These two points will be samples by displacing the original point of the cloud through the two possible directions of its normal. The displacement has to be a very small value in order to not create any undesired distortion (in our case we have selected the value *1/64* for the displacement).

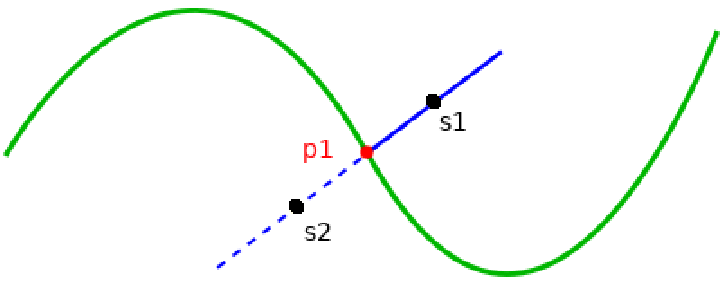


Figure 4.3: An illustrated example about how to obtain the sampling points *s1* and *s2* from the given point of the point cloud *p*.

4.4 Smoothing Equations

The smoothing equations are built through a Laplacian smoothing. The Laplacian smoothing is an algorithm meant to smooth a polygonal mesh. For each corner in a mesh that is **not** on the margins of the field, the energy is chosen based on local information of the corners neighbors of the field grid. Our case in which a mesh is topologically a rectangular grid in which each internal vertex is connected to exactly four neighbors. Afterward, this operation produces the Laplacian of the mesh. More formally, the smoothing equations may be described per-vertex as:

$$\bar{x}_i = \frac{1}{N} \sum_{j=1}^{N} \bar{x}_j \tag{4.6}$$

Where N is the number of adjacent node corners, i,$\bar{x}_j$ is the position of the j-th adjacent corner neighbor and $\bar{x}_i$ is the new energy value for the i-th corner. For our surface reconstructor we have implemented two different possible systems in order to add the smoothing equations:

- One Dimensional Laplacian (1D). In this case, we will iterate through each node corner of the field two times. In the first pass, we will add one equation per node corner taking into account only the horizontal neighbor corners (left, and right). Later in the second pass, we will only take into account the vertical neighbor corners (top and bottom).
- Two Dimensional Bilaplacian (2D). In this case, we will first iterate through each node corner once. On this pass, we will add one equation per node corner taking into account the four neighbor nodes (left, right, top and bottom). Later we will iterate through the neighbors that are two positions away from the objective corner.

4.5 Iterative Solvers

An iterative solver method is a mathematical procedure that uses an initial value to generate a sequence of improving approximate solutions for a class of problems, in which the n-th approximation is derived from the previous ones. An iterative method is called convergent if the corresponding sequence converges for given initial approximations.

Direct methods attempt to solve the problem by a finite sequence of operations. In the absence of rounding errors, direct methods would deliver an exact solution. However, iterative methods are often useful even for linear problems involving many variables, where direct methods would be prohibitively expensive even with the best available computing power.

We will make use of the iterative solving sparse linear systems of Eigen library [17]. In Eigen, there are several methods available to solve linear systems when the coefficient matrix is sparse. Because of the special representation of this class of matrices, special care should be taken to get a good performance. Depending on the properties of the matrix, the desired accuracy, Eigen solvers allow tuning those steps to improve the performance. The three iterative solvers that present Eigen are the **Conjugate Gradient**, the **Least Squares Conjugate Gradient** and the **BiCGSTAB** solvers.

The conjugate gradient method means to be the classic iterative conjugate gradient method whose matrix is positive-definite (a symmetric matrix A is called positive definite if the real number $x^T Ax$ is positive for every nonzero real column vector x [18]). From now on we will suppose our build-in matrix A is positive definite. This method is recommended for large symmetric problems, therefore is a potential solver option.

The least-squares conjugate gradient is a conjugate gradient solver for sparse or dense rectangular least-square problems. This least-squares conjugate gradient solves for min $|A'Ax - b|^2$ without forming A'A, therefore there is no need to compute the transposed matrix of A.

Finally, the biconjugate gradient stabilized method (BiCGSTAB), is a variant of the biconjugate gradient method (BiCG) with a faster and smoother convergence than the original BiCG. The vectors x and b can be either dense or sparse. To speed up the convergence, we can add the **IncompleteLUT** preconditioner. If we add the IncompleteLUT preconditioner two dropping rules are used :

1. Any element whose magnitude is less than some tolerance is dropped. This tolerance is obtained by multiplying the input tolerance droptol by the average magnitude of all the original elements in the current row.
2. After the elimination of the row, only the fill largest elements in the **L** (lower) part and the fill largest elements in the **U** (upper) part are kept (in addition to the diagonal element).

Note that fill is computed from the input parameter fill factor which is used the ratio to control the fill relatively to the initial number of nonzero elements.

4.6 Multithreading

Some of Eigen's sparse iterative solver methods can exploit the multiple cores present in the hardware. In particular, Eigen uses the number of threads specified by **OpenMP**. We have to take into account that in most operating systems it is very important to limit the number of threads to the number of physical cores, otherwise, significant slowdowns may occur, especially for operations involving dense matrices.

However, Eigen's matrix-matrix product kernel is fully optimized and already exploits nearly 100 percent of the CPU capacity. Consequently, there is no room for running multiple such threads on a single core, and the performance would drop significantly because of cache pollution and other sources of overheads. Eigen does not limit itself to the number of physical cores because OpenMP does not allow to know the number of physical cores, and thus Eigen will launch as many threads as cores reported by OpenMP.

4.7 Multigrid

In numerical analysis, a multigrid method is an algorithm for solving differential equations using a hierarchy of discretizations. They are an example of a class of techniques called multiresolution methods which are very useful in problems exhibiting multiple scales of behavior. Multigrid can solve a linear system with N unknowns with an asymptotic cost of only *O(N)*.

Multigrid methods achieve optimality by employing two complementary processes: *smoothing* and *coarse-grid correction*; The main idea of multigrid is to accelerate the convergence of a basic iterative method (known as relaxation, which generally reduces short-wavelength error) by a global correction of the fine grid solution approximation from time to time, accomplished by solving a coarse problem. The coarse problem, while cheaper to solve, is similar to the fine grid problem in that it also has short- and long-wavelength errors. It can also be solved by a combination of relaxation and appeal to still coarser grids. This recursive process is repeated until a grid is reached where the cost of direct solution there is negligible compared to the cost of one relaxation sweep on the fine grid. The multigrid cycle typically reduces all error components by a fixed amount bounded well below one, independent of the fine grid mesh size.

In contrast to other methods, multigrid methods are general can treat arbitrary regions and boundary conditions and they do not depend on the separability of the equations or other special properties of the equation. The multigrid approach often scales linearly with the number of discrete nodes used. In other words, it can solve these problems to a given accuracy in several operations that are proportional to the number of unknowns having this way the advantage over other methods.

Our multigrid implementation is simple can be resumed as follows:

- Start with a low resolution system $Ax_0 = b$ and find the solution x_0.
- Use x_0 as guess initial solution for the system $Ax_1 = b$.
- Reproduce the previous steps until we reach the x_i solution.

Chapter 5

Quadtree Reconstruction

5.1 Introduction to Quadtree

A quadtree is a data structure in which each internal node has exactly four children. A quadtree is an appropriate structure for storing information that will be retrieved on composite keys. Quadtrees are the two-dimensional analog of octrees and are most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions.

The location of the points records with two-dimensional keys will be stored in a tree with out-degree four at each node corner. Each node corner will store one record and will have up to four sons, each a node corner. The root of the tree divides the universe into four quadrants, namely for example *TopLeft*, *TopRight*, *BottomLeft*, and *Bottom-Right* corners. The following Figure 5.1 shows the correspondence between a simple tree and the records it represents.

Quadtrees may be classified according to the type of data they represent, including areas, points, lines, and curves. Quadtrees may also be classified by whether the shape of the tree is independent of the order in which data is processed. The most common type of quadtree is the Region Quadtree which is also the one that we will use for our surface reconstruction.

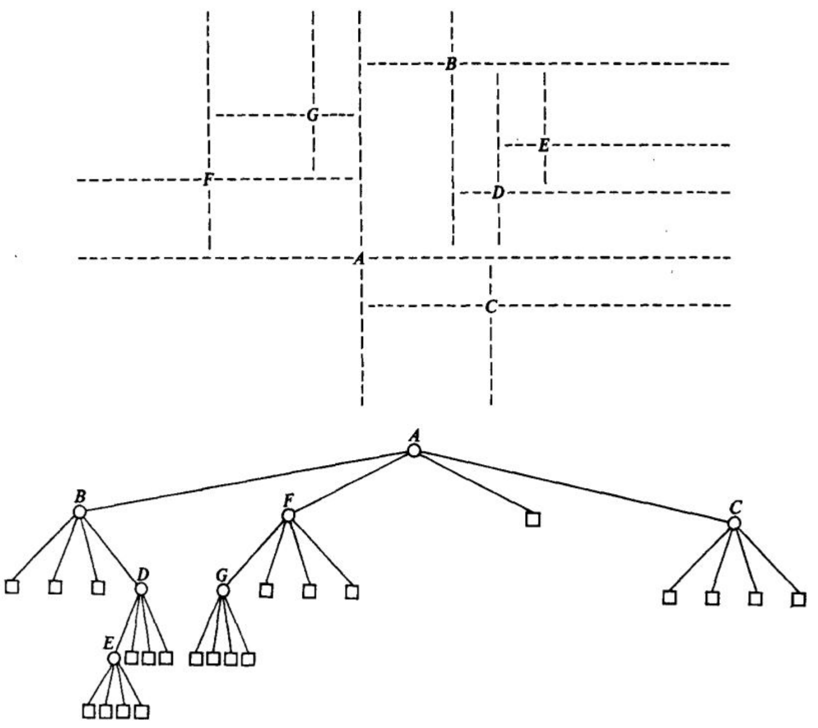


Figure 5.1: Correspondence of a quad tree to the records it represents. Records A, B, C, D, E, F, G. Null subtrees are indicated by boxes, but they do not appear explicitly in computer memory.

The region quadtree represents a partition of space in two dimensions by decomposing the region into four equal quadrants, sub-quadrants, and so on with each leaf node containing data corresponding to a specific subregion. Each node in the tree either has exactly four children or has no children (a leaf node). The height of quadtrees that follow this decomposition strategy (i.e. subdividing sub-quadrants as long as there is interesting data in the sub-quadrants for which more refinement is desired) is sensitive to and dependent on the spatial distribution of interesting areas in the space being decomposed. The region quadtree is a type of *prefix tree*.

A region of a quadtree with a depth of n may be used to represent a grid consisting of $2^n 2^n$ cells, where each cell will contain surface points, or otherwise, it will be empty. The root node represents the entire grid region. If the cell in any region is not entirely surface's points, it is subdivided. The tree resolution and overall size are bounded by the cell and point sizes. Since our region quadtree is used to represent a set of surface point data, regions are subdivided until each leaf contains at most a single point.

Potential savings in terms of space when these trees are used for storing for example images; images often have many regions of considerable size that have the same color value throughout. Rather than store a big 2-D array of every pixel in the image, a quadtree can capture the same information potentially many divisive levels higher than the pixel-resolution-sized cells that we would otherwise require.

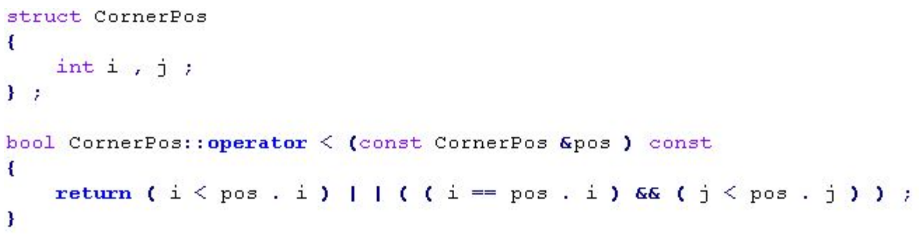
5.2 Adding Surface Equations from a Quadtree

The methodology that we will use to resolve our system equation will be exactly the same as the one we used on the iterative implementation. However, The number of equations generated will be significantly reduced, since we will through each *corner*, avoiding navigating through all the divided grids. Therefore, in order to perform it we need to be able to do some steps efficiently:

- Collect all the *corners* of the terminal quadtree nodes without repetitions. These will become variables in our system. It will also be useful to be able to compare them and sort them so that we can have a structure that translates from a corner to a variable id.
- For the point's positions equations we will find the terminal node that contains the point and will build the equation from the node's corners and the position of the point inside the node.
- For the point's normal equations, regardless of the approach (gradient or sampling), we may follow the same strategy used previously.
- For the smoothing prior equations:
  - Build one equation (or two if it is implemented the 1D solution) for each variable for each corner of a terminal node of the quadtree.
  - The neighbors in the smoothing prior equation are sampled at maximum resolution, but they may not be variables. Those that are not variables will have to be interpolated from the available corners.
  - Only equations that use samples from inside the scalar field will be kept. For this, we need to implement a function that given a sample position in the scalar field tells us if it's inside and another function that tells us the corners we need to interpolate to get the value at that sample's position and with which weights.

Keeping this in mind, we can state that:

- The maximum resolution should be of the form $2k + 1$, where k is the maximum depth of the quadtree.
- Any corner of any node has a pair (i, j), where$(0i < 2k + 1)(0j < 2k + 1)$, that identifies it and allows to define an order via an operator $<$.
- This pair can be computed from a given node if each of them has its box defined in those coordinates. Therefore, the operator needed to establish an order would be as shown in the below figure 5.2:



```cpp
struct CornerPos
{
    int i , j ;
} ;

bool CornerPos::operator < (const CornerPos &pos ) const
{
    return ( i < pos . i ) | | ( ( i == pos . i ) && ( j < pos . j ) ) ;
}
```

Figure 5.2: Code structure in C++ used to define the corner's order.

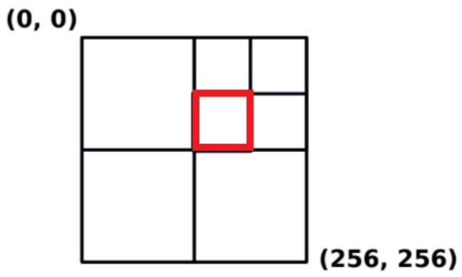


Figure 5.3: Example of a scalar field of size 257x257. The highlighted node has a box of (128, 64) (192, 128). Hence, its corners are: (128, 64),(192, 64),(128, 128),(192, 128)

The smoothing prior equation takes samples around each corner of a quadtree node. These samples are taken at maximum resolution. Using the 5.3, for corner (128, 128), the locations of the 5 samples needed for the smoothing prior equation would be: (128, 128) for the corner itself, (129, 128) for the right neighbor corner, (127, 128) for the left neighbor corner, (128, 129) for the top neighbor corner, and (128, 127) for the bottom neighbor corner.

Therefore, determining if a sample is inside the scalar field is easy. For sample, given the sample s with location (i, j), s is inside if $(0 \leq i < 2k+1) \wedge (0 \leq j < 2k+1)$. If we find the node that contains the sample, it is easy to extract its corners and interpolation weights. If we take the neighbor samples and all samples are corners we will have the easiest case in which the prior equation formula will have a direct solution. Nonetheless, let's take a look at the case below :

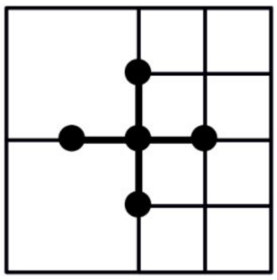


Figure 5.4: Example of a scalar field of size 5x5. Note than the sample (1, 2) is not a corner, thus not having an assigned variable.

The samples for the prior equation would be (2, 2), (1, 2), (3, 2), (2, 1), (2, 3). However, if we take a look at the sample (1, 2) we can observe that it is not a corner, thus it is not having an assigned variable. Moreover, this node is located in the middle of an edge between two nodes. If the values of the scalar field at those locations are $v_{2,2}, v_{1,2}, v_{3,2}, v_{2,1}, v_{2,3}$ then the prior equation is:

$$v_{1,2} + v_{3,2} + v_{2,1} + v_{2,3} 4 v_{2,2} = 0$$

As sample v1,2 is not a corner, we have to obtain it from the interpolation of corners:

$$v1, 2 = 1/2 v_{0,2} + 1/2 v_{2,2}$$

If we substitute the final equation only uses values that will be variables in our system:

$$1/2 v_{0,2} + v_{3,2} + v_{2,1} + v_{2,3} 7/2 v_{2,2} = 0$$

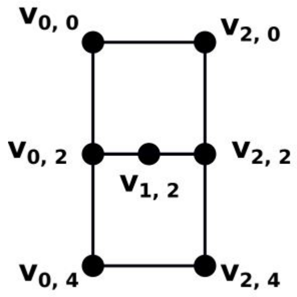


Figure 5.5: Example case in which the sample (1, 2) is not a corner, thus not having an assigned variable. Since the sample is located into an edge conformed by two corners, the value of the sample (1, 2) will have an interpolated value between (0, 2) and (2, 2).

Now let's talk about another corner case example. In this case, the sample will not be located on an edge between two corners but will be located in the middle of a cell. Let's take a look a the below figure 5.6:

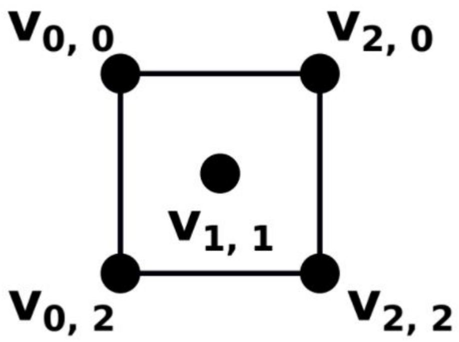


Figure 5.6: Example case in which the sample (1, 1) is not a corner, thus not having an assigned variable. Since the sample is located into a cell conformed by four corners, the value of the sample (1, 1) will have an interpolated value between (0, 0), (0, 2), (2, 0), and (2, 2).

The samples for the prior equation would be (2, 1), (1, 1), (3, 1), (2, 0), (2, 2) where only sample (1, 1) is not a corner. Moreover this node is located in the middle of a cell between four nodes corners . The prior equation is:

$$v_{3,1} + v_{1,1} + v_{2,0} + v_{2,2} 4v_{2,1} = 0$$

Since the sample $v_{1,1}$ is not a corner, we have to obtain it from the interpolation of corners:

$$v_{1,1} = 1/4v_{0,0} + 1/4v_{2,0} + 1/4v_{0,2} + 1/4v_{2,2}$$

Finally, if we substitute:

$$v_{3,1} + 1/4v_{0,0} + 1/4v_{0,2} + 5/4v_{2,0} + 5/4v_{2,2} \text{ - } 4v_{2,1} = 0$$

Chapter 6

Experimental Results

6.1 Description of the tested models

To perform our experiments, we have generated **10** different models, each with different forms and shapes. When we generated the models we have taken into account different aspects that could impact the performance of our surface reconstructor. Hence we present from very simple models such a the *Sphere* model, which is only a simple curvature, to more complex models such as the *Superman* logo which not only combine curves and sharp edges but also presents different levels of layers with inner and outer normal directions.

Moreover, we constructed models whose bounding box presents elongated shapes such as the *Key* or the *Capsule* Models while others tend to have a squared bounding box such as the *Circle* or the *Suzanne* models.

Finally, for each model we have two different versions; the *high-level* resolution version, which contains a high level of points on the cloud, and the *low-level* resolution, which contains at least (or even less) half of the points on the cloud. The mentioned models will be displayed during the following sections.

6.2 Analysis of the Point Cloud Artifacts

6.2.1 Analysis of the impact of the sampling density

In this section, we will evaluate the impact of the noise and the outliers. For this experiment, we are going to make use of the *high* and *low* level point clouds of the given data models. The low-level models try to maintain a uniform distance between the sampled points in order to not mix it with the missing data artifact. Moreover, we will try to maintain the neighborhood large enough so that the points sufficiently describe the local geometry.

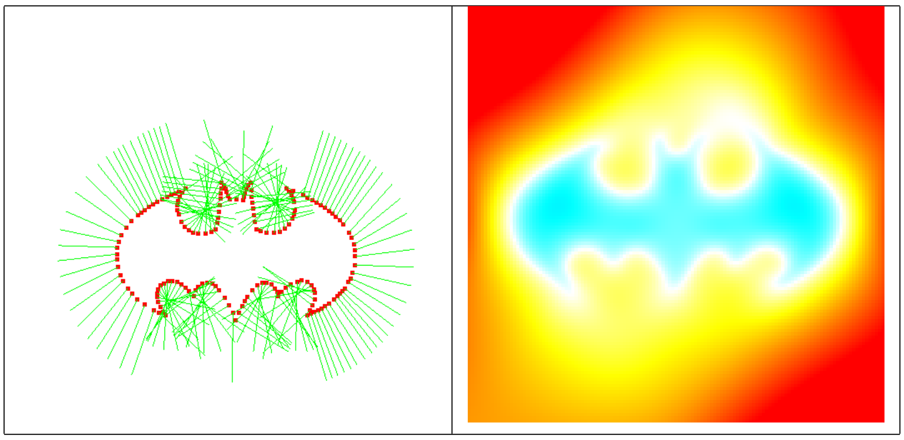


Figure 6.1: Image showing surface reconstruction with a high level of points on the cloud for the *Batman* model. The left image shows the point's and normal's cloud information. The right image shows to reconstructed surface's output image. The relative error is 0,591749.

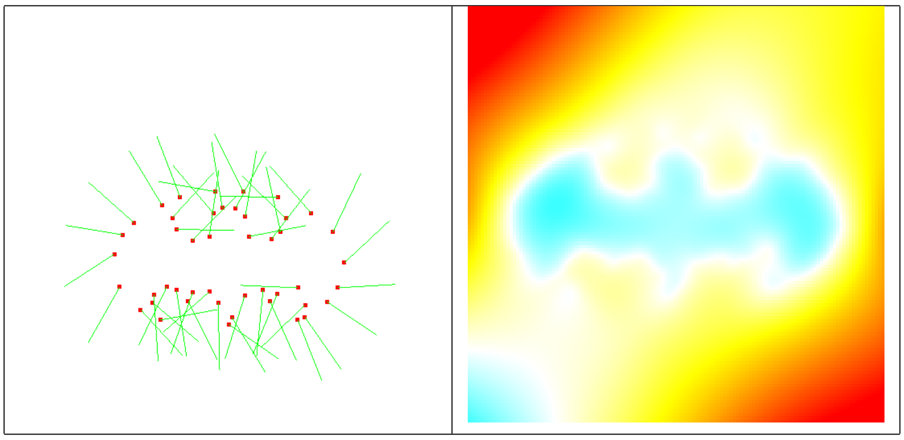


Figure 6.2: Image showing surface reconstruction with a low level of points on the cloud for the *Batman* model. The left image shows the point's and normal's cloud information. The right image shows to reconstructed surface's output image. The relative error is 0,799591.

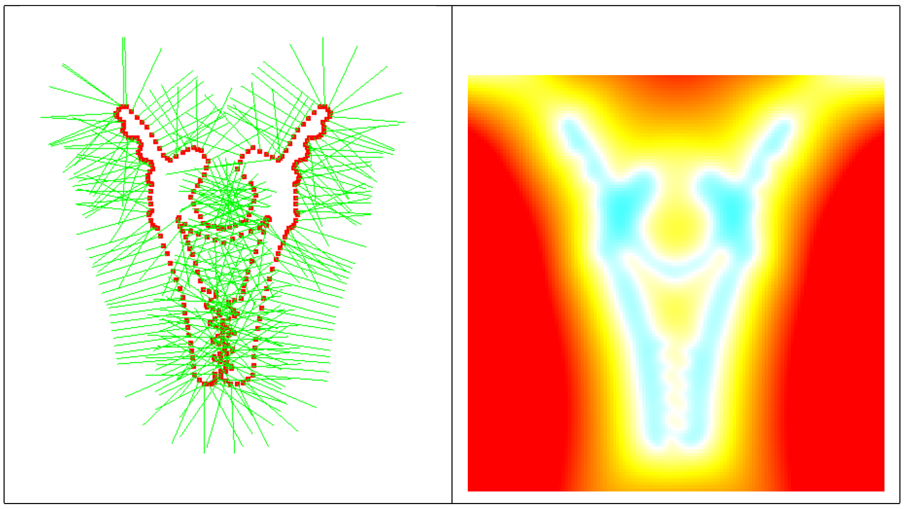


Figure 6.3: Image showing surface reconstruction with a high level of points on the cloud for the *Clip* model. The left image shows the point's and normal's cloud information. The right image shows to reconstructed surface's output image. The relative error is 0,681038.

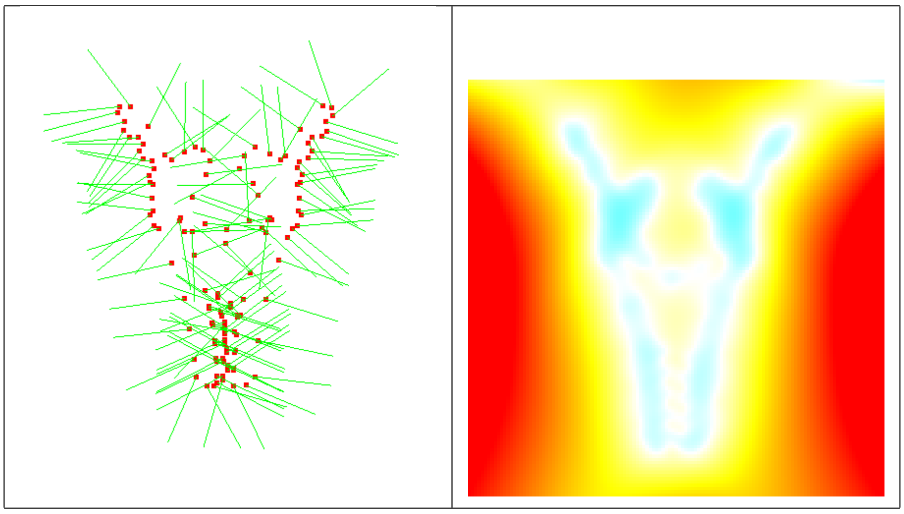


Figure 6.4: Image showing surface reconstruction with a low level of points on the cloud for the *Clip* model. The left image shows the point's and normal's cloud information. The right image shows to reconstructed surface's output image. The relative error is 0,798002.

After taking a look at the figures 6.1, 6.2, 6.3, and 6.4 we can state than number of points on the cloud impacts directly to the quality of the reconstructed surface. The relative error is much lower on the models with a high number of points on the cloud rather than a lower number of points. The reconstructed surfaces of the models with a lower number of points omit the more complex shapes and forms. However, the sections with straight lines or obtuse curves are quite well represented for the low number of data info.

6.2.2 Analysis of the impact of the noise and outliers

In this section, we will evaluate the impact of the noise and the outliers. The noise and outliers impacts directly surface properties, scattering the characteristics of the surface to reconstruct. Therefore, a model with a high level of noise and outliers could deform the surface provoking to completely change the shape of the reconstructed surface or generate a totally different surface shape.

For this experiment, we have added an extension to our executable which will add *Gaussian* noise to the point cloud points before the model is processed. Note that there is no sense in using big values for the standard deviation on the *Gaussian* noise since it would displace the points so much that the model to reconstruct would be unrecognizable.

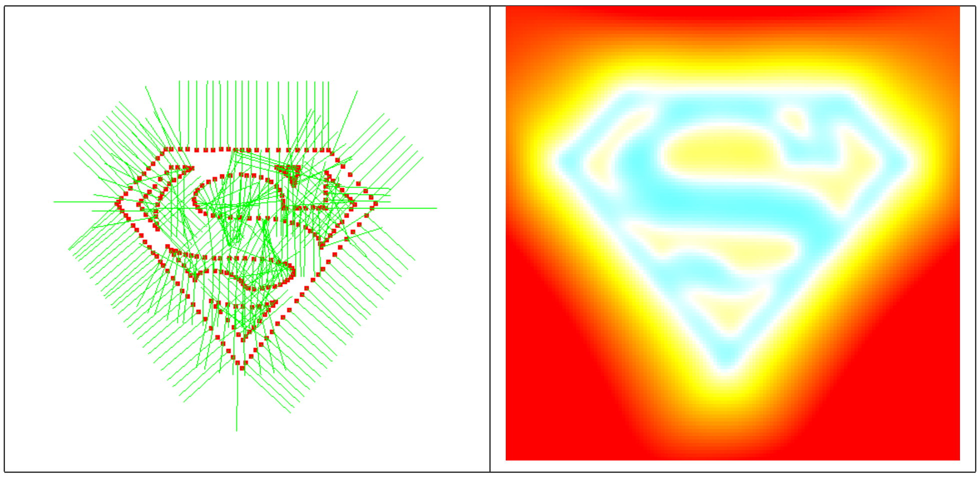


Figure 6.5: Result of reconstructing the *Superman* surface logo with a standard deviation of 0.0 applied to the point cloud. The relative error was 0,723597.

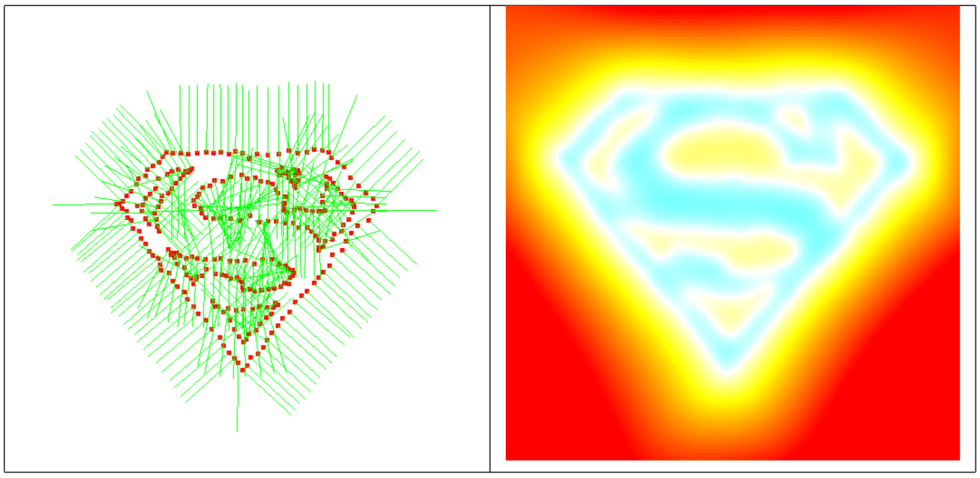


Figure 6.6: Result of reconstructing the *Superman* surface logo with a standard deviation of 0.005 applied to the point cloud. The relative error was 0,699843.

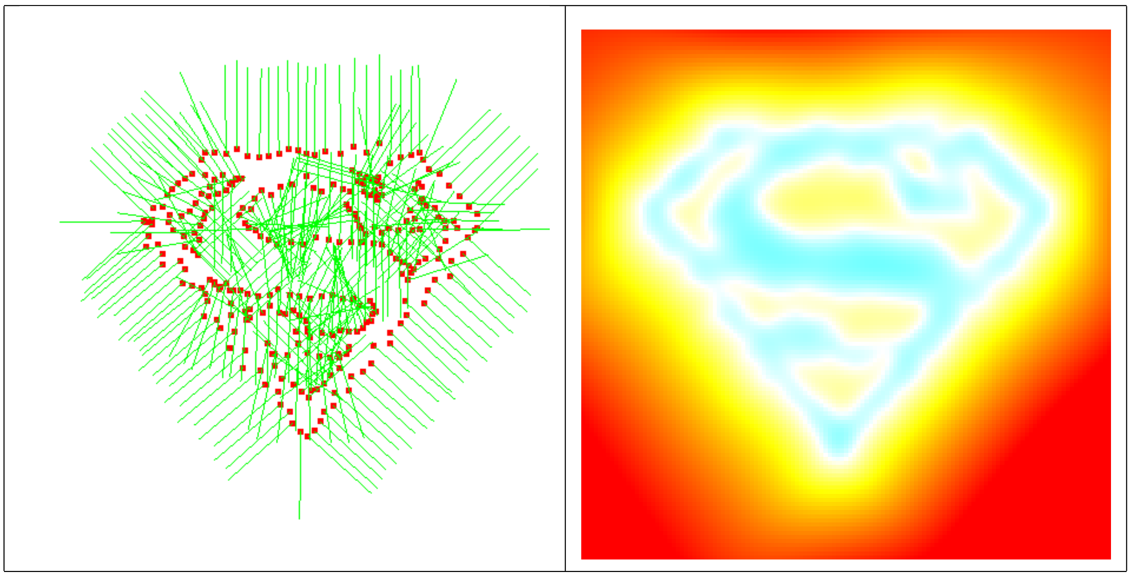


Figure 6.7: Result of reconstructing the *Superman* surface logo with a standard deviation of 0.01 applied to the point cloud. The relative error was 0,746836.

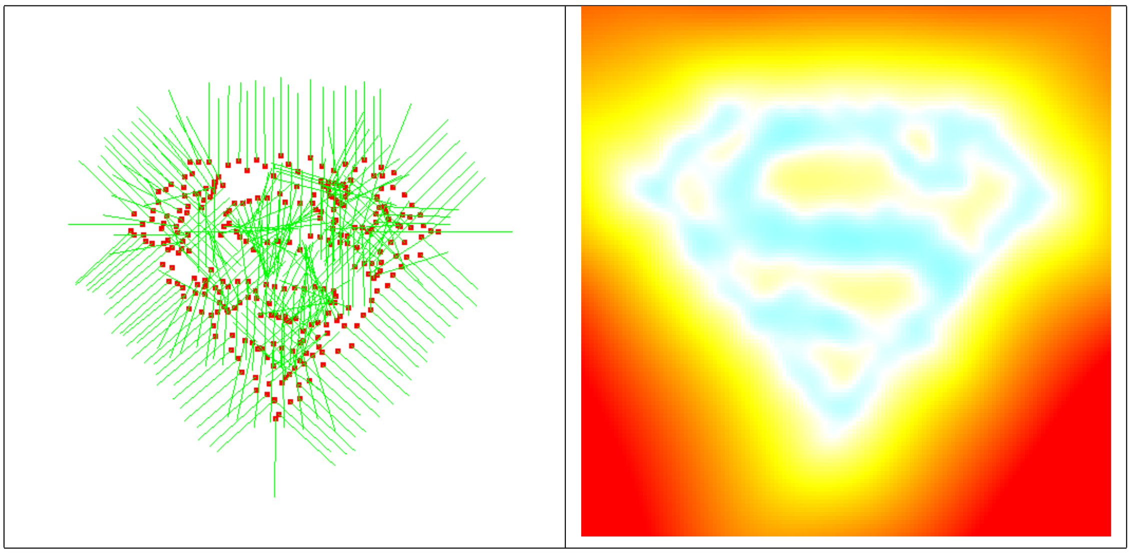


Figure 6.8: Result of reconstructing the *Superman* surface logo with a standard deviation of 0.015 applied to the point cloud. The relative error was 0,809152.

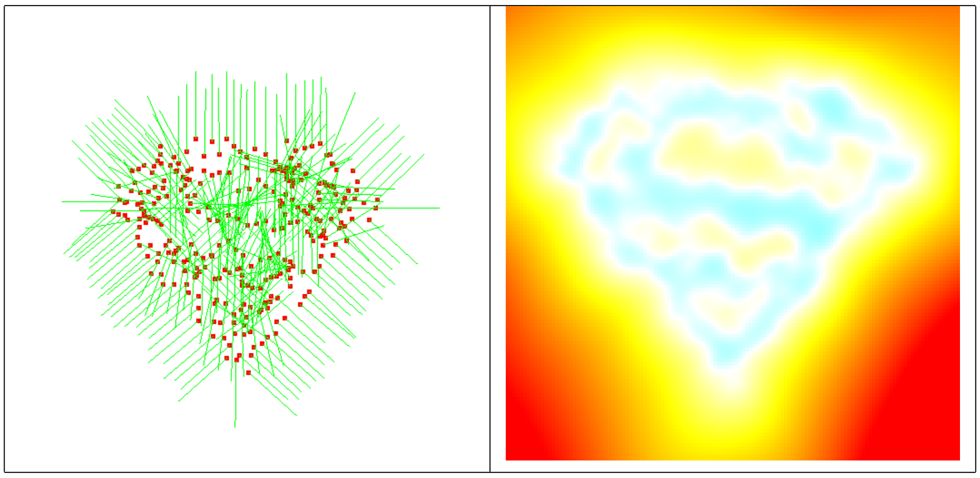


Figure 6.9: Result of reconstructing the *Superman* surface logo with a standard deviation of 0.02 applied to the point cloud. The relative error was 0,840021.

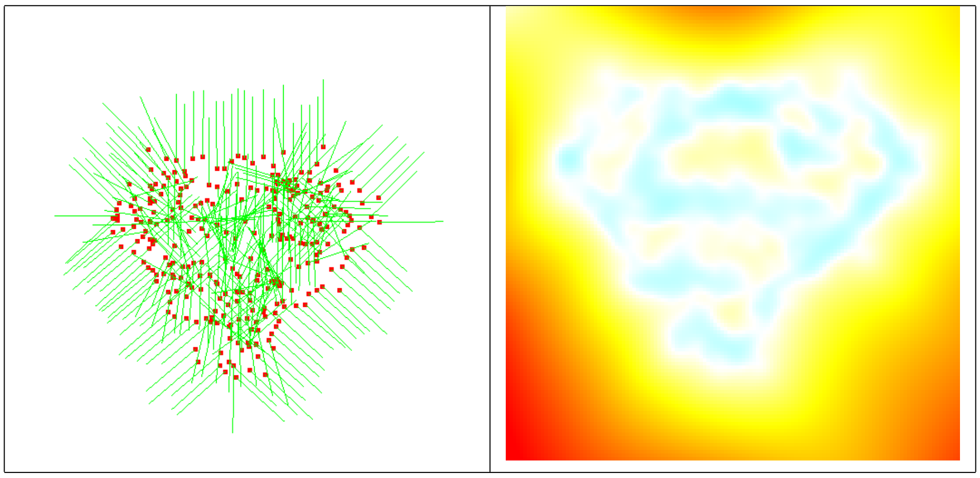


Figure 6.10: Result of reconstructing the *Superman* surface logo with a standard deviation of 0.025 applied to the point cloud. The relative error was 0,873228.

If we take a loot at the figures 6.5, 6.6, 6.7, 6.8, 6.9, and 6.10 we can observe the reconstructed surface given a point cloud set, each figure with a higher value of standard deviation of the Gaussian noise applied to it.

In general terms, we can say that the performance of our implemented system is quite good. We can appreciate that the relative error increases at the time we increase the standard deviation. This is normal since when we increase the deviation the shape of the model to reconstruct becomes more ambiguous. Moreover, if we take a look at the figure 6.10 which presents a high value of standard deviation and where all the points are mostly outliers, despite the relative error value being really high, he mostly preserved the general shape and holes of the original surface.

6.2.3 Analysis of the impact of missing data

In this section, we will evaluate the impact of the missing data. The models with missing data will present sections where the sampling density of points of the cloud will be zero. Note that we will refer to the density in tan percent being 100 percent a model with all the points information on the cloud and 0 percent a model with no point content.

For this experiment, we have added an extension to our executable which will add specify the sampling density we desire and it will generate chunks on the surface model in which the sampling density will be zero.

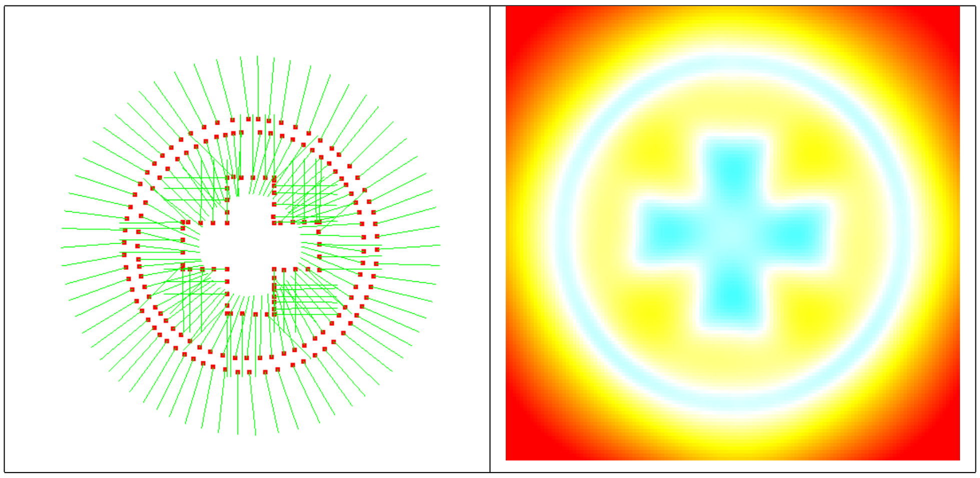


Figure 6.11: Image showing surface reconstruction of the *Pharmacy Cross* model using the 100 per cent of the point cloud points.

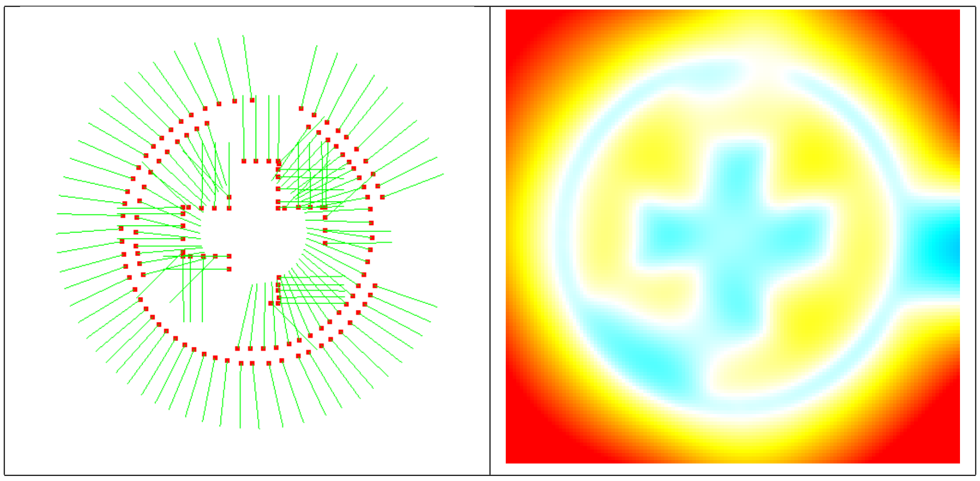


Figure 6.12: Image showing surface reconstruction of the *Pharmacy Cross* model using the 90 per cent of the point cloud points.

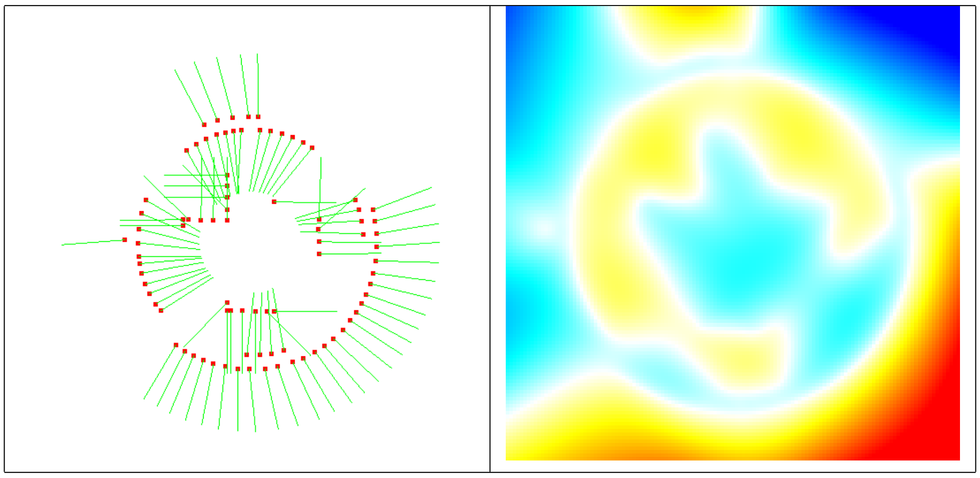


Figure 6.13: Image showing surface reconstruction of the *Pharmacy Cross* model using the 80 per cent of the point cloud points.

If we take a look at the Figures 6.11, 6.12, and 6.13 we can state that the models with big chunks where the sampling density is zero will have a big impact on the reconstructed surface. Moreover, the overall sampling density of the input models will have a direct relationship with the missing data artifact. With a low level of sampling density, the missing data behavior will increase since the quality of the sampled details will be more prone to delete the original path and merge it with the adjacent ones.

6.3 Analysis of the Implemented Methods

6.3.1 Analysis of the performance of the Normal Equations

As we previously mentioned on the section 4.3, we have implemented two different implementations to construct the normal equations. In this chapter, we will analyze the performance of the obtained results comparing the execution time as well as the resulting quality of the reconstructed surfaces for each implemented method. For this study, we have fixed the smoothing equations to *1D* and the iterative solver to *Conjugate Gradient*. We also fixed the number of threads to **8**. Moreover, we have run the executable **25** times to obtain the average of the execution time for each iteration:

Normal	Sys (ms)	Mat (ms)	Solver (ms)	Total (ms)	Iterations	Error	Rel Error
Gradient	26,7	27,9	2714,6	2769,2	2963	9.0753e-06	0.447545
Sampling	26,9	30,6	2553,2	2610,5	2790	9.0562e-06	0.365539

Table 6.1: Content table of average results obtained for the Normals equation's performance analysis with the *Biharmonic* method for *Suzane* model. The System, Matrix, Solver, and Total times are represented in milliseconds. The field size used for the surface reconstruction was *65*.

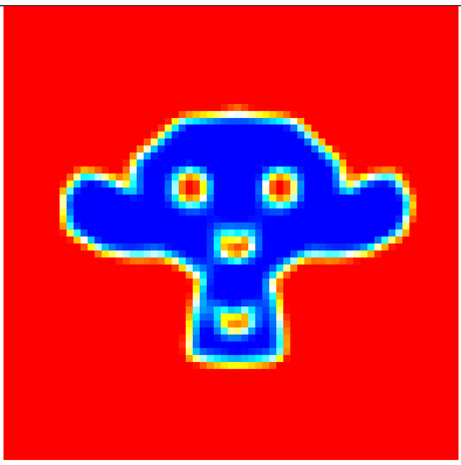
Normal	Sys (ms)	Mat (ms)	Solver (ms)	Total (ms)	Iterations	Error	Rel Error
Gradient	196,7	75,8	3543,4	3815,9	1472	9.7198e-06	0.82808
Sampling	195.3	77.2	3251.3	3523.8	1345	9.6823e-06	0.73078

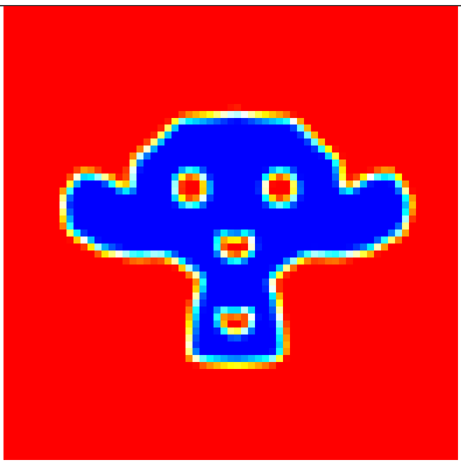
Table 6.2: Content table of average results obtained for the Normals equation's performance analysis with the *Quadtree* method for *Suzane* model. The System, Matrix, Solver, and Total times are represented in milliseconds. The field size used for the surface reconstruction was *1025*.

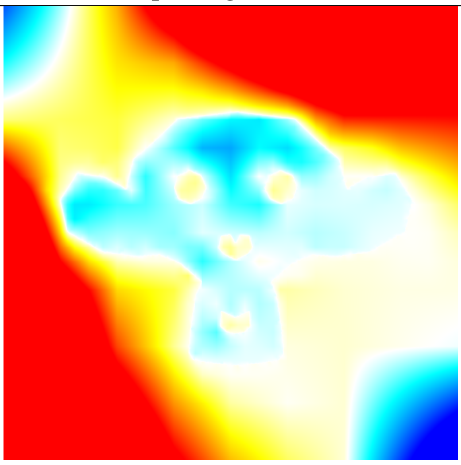
We can appreciate to resulting tables 6.1 and 6.2 than the *Sampling* Normal system equations outperforms the *Gradient* one. Not only the number of iterations performed by the solver to reach a feasible solution of both *Biharmonic* and *Quadtree* surface reconstruction methods was lesser on the *Sampling* system rather than the *Gradient* one, but also the given relative error of the equation system is lesser using the *Sampling* Equations compared to the *Gradient* Equations. As for the execution time we can state than are almost equivalent (it's logical since both presented the same number of equations and unknowns).

This error difference can be appreciated on the 6.14 table where we can observe the surface reconstruction's output images generated by both *Biharmonic* and *Quadtree* methods using the different Normal equation systems. If we take a look at the output images for the *Biharmonic* we cannot appreciate almost any difference between the two reconstructions. However for the *Quadtree* surface reconstruction we can clearly notice the difference between the different methods. Using the *Sampling* method we can obtain a more faithful representation of the surface model, with more clear-shaped lines and lesser noise rather than the *Gradient* system which creates noise inside and outside our reconstructed surface which would have a great impact on the quality.

This way, despite we implemented the *Sampling* equations as a simplification of the *Gradient* ones, we can state that adding *Sampling* equations rather than the *Gradient* equations will generate a better quality solution not only on the gradient field quality but also with a much better solver time, performing less iteration in order to obtain a solution with a smaller error.


Reconstructed surface output at 64px. for the *Gradient* eq. using *Biharmonic* method.


Reconstructed surface output at 64px. for the *Sampling* eq. using *Biharmonic* method.


Reconstructed surface output at 1024px. for the *Gradient* eq. using *Quadtree* method.

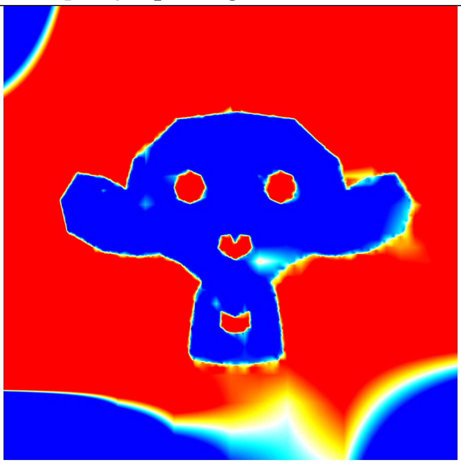

Reconstructed surface output at 1024px. for the *Sampling* eq. using *Quadtree* method.

Figure 6.14: Images obtained from reconstructing *Suzane* model with the Gradient and Sampling system equations with the Biharmonic and Quadtree methods.

6.3.2 Analysis of the performance of the Smoothing Equations

In the chapter 4.4 we talked about the *Laplacian* (1D) and *Bilaplacian* (2D) smoothing equations. In this section, we will evaluate the performance of both methods and the impact in terms of execution time and quality of the resulting from the reconstructed surface. For this study, we have fixed the normal equations to *Sampling* and the iterative solver to *Conjugate Gradient*. We also fixed the number of threads to **8**. Moreover, we have run the executable **25** times to obtain the average of the execution time for each iteration:

Smoothing	Num Eq	Num Unk.	Total (ms)	Iterations	Error	Rel Error
1D	6381	4225	2610,5	2790	9.0753e-06	0.365539
2D	10350	4225	3920.6	3877	8.98361e-06	0.405216

Table 6.3: Content table of average results obtained for the Smoothing equations's performance analysis with the *Biharmonic* method for *Suzane* model. The Total times are represented in milliseconds. The field size used for the surface reconstruction was 65.

If we take a look at the table 6.3 we can see that the 2D smoothing equation system has generated a bigger number of system equations and consequently, the total execution time and the number of iterations is performed to reach a feasible solution was bigger than the system than used the 1D smoothing equations. Moreover, the relative error obtained by the 1D smoothing was lower than the obtained by the system that used the 1D smoothing equations, meaning that the reconstruction obtained better results if we used the 1D *Laplacian* smoothing rather than the 2D *Bilaplacian* equation system.

In fact, if we take a look at the figure 6.16 we can appreciate some lightweight green tones outside the reconstructed surface model but not inside it. However, if we take a look at the figure 6.16 we can see pink spots inside the reconstructed surface, meaning that there is some difference between the two reconstructed surface outputs. Nonetheless, we cannot see any spot that falls just in the middle of the reconstructed silhouette which is the most important part.

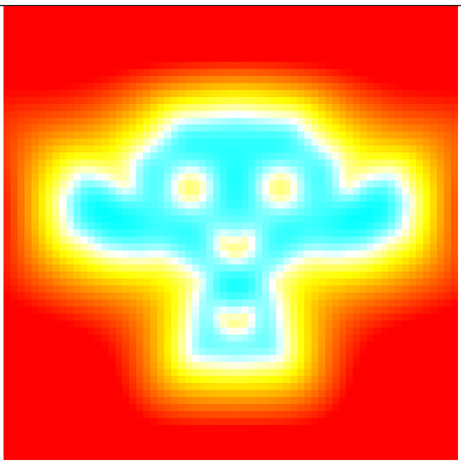
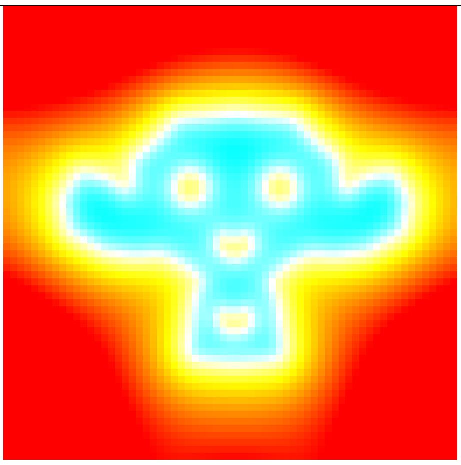
 Reconstructed surface output at 64px. for the *1D* eq. using *Biharmonic* method.	 Reconstructed surface output at 64px. for the *2D* eq. using *Biharmonic* method.

Figure 6.15: Images obtained from reconstructing *Suzane* model with the Gradient and Sampling system equations with the Biharmonic and Quadtree methods.

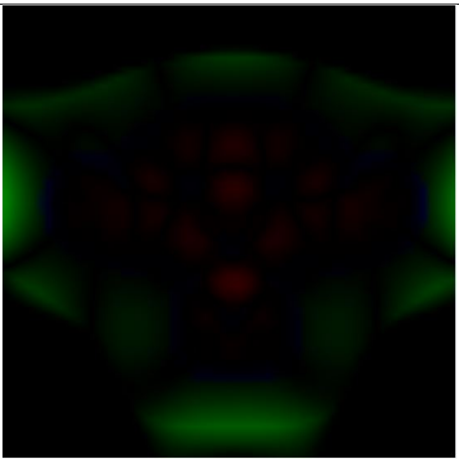
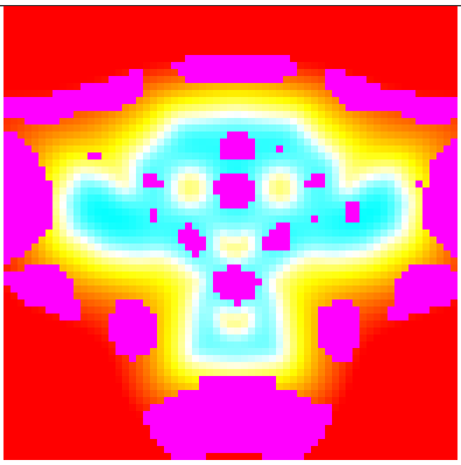
 Image showing the main differences represented in greenscale of using 1D and 2D smoothing eq with the *Biharmonic* method.	 Image showing the main differences represented with pink stains of using 1D and 2D smoothing eq with the *Biharmonic* method.

Figure 6.16: Images representing the main differences obtained by representing *Suzane* model with the 1D and 2D smoothing equations with the Biharmonic method.

6.3.3 Analysis of the performance of the Iterative Solvers

As we already introduced on the section 4.5, Eigen presents three iterative linear solvers to solve equation systems of the form $Ax = b$. This chapter aims to analyze the performance of the obtained results in terms of execution time as well as the resulting quality of the reconstructed surfaces for each iterative linear solver. For this study, we have fixed the smoothing equations to *1D* and the iterative normal equations to *Sampling*. We also fixed the number of threads to 8. Moreover, we have run the executable **25** times to obtain the average of the execution time for each iteration:

It Solver	Sys (ms)	Mat (ms)	Solver (ms)	Total (ms)
BiCGSTAB	26,3	15,9	3166,1	3208,3
ConjGrad	26,9	30,6	2553,2	2610,5
LstSqrConjGrad	20,6	0	7733,3	7753,9

Table 6.4: Content table of average times obtained for the iterative solver's performance analysis with the *Biharmonic* method for *Suzane* model. The System, Matrix, Solver, and Total times are represented in milliseconds. The field size used for the surface reconstruction was **65**.

It Solver	Iterations	Error	Rel Error
BiCGSTAB	1784	8,4094e-06	0,365542
ConjGrad	2790	9,0562e-06	0,365539
LstSqrConjGrad	2790	9,8060e-06	0,365539

Table 6.5: Content table of the error results obtained for the iterative solver's performance analysis with the *Biharmonic* method for *Suzane* model. The field size used for the surface reconstruction was **65**.

As we can appreciate to resulting tables 6.4 and 6.6 the iterative linear system with better execution time is the *Conjugate Gradient* solver. However, if we take a look at tables 6.5 and 6.7 the *BICGSTAB* solver has performed a lesser number of iterations to reach a feasible solution and its error is lower than the other ones, meaning that he found a more precise reconstruction for the surface. Finally, we can state that the *Least Squares ConjugateGradient* obtained the worst performance. Not only it was over three times slower than the *Conjugate Gradient* but also obtained the worst error meaning that he reached the worst surface reconstruction.

It Solver	Sys (ms)	Mat (ms)	Solver (ms)	Total (ms)
BiCGSTAB	195,6	72,6	8021,7	8289,9
ConjGrad	195,3	77,2	3251,3	3523,8
LstSqrConjGrad	160,2	0	9233,2	9393,4

Table 6.6: Content table of average time results obtained for the iterative solver's performance analysis with the *Quadtree* method for *Suzane* model. The System, Matrix, Solver, and Total times are represented in milliseconds. The field size used for the surface reconstruction was **1025**.

It Solver	Iterations	Error	Rel Error
BiCGSTAB	1729	8,4610e-06	0.73078
ConjGrad	1345	9.6823e-06	0.73078
LstSqrConjGrad	1345	9.8806e-06	0.73078

Table 6.7: Content table of the error results obtained for the iterative solver's performance analysis with the *Quadtree* method for *Suzane* model. The field size used for the surface reconstruction was **1025**.

Although the Matrix Build time is zero, the impact of this computation is insignificant compared to the other times. Despite the error of the *BICGSTAB* being less than the other two, if we take a look at the figure 6.17 we cannot appreciate any significant difference between the reconstructed images obtained for each of the iterative linear solvers.

However, if we take a look at the figure 6.18, we can see the main differences between the three methods. As we can observe the differences between the *Conjugate Gradient* and the *Least Squares Conjugate Gradient* are null meaning they performed the same surface output reconstructions. Nonetheless, with the *BICGSTAB* comparisons we can appreciate a light difference especially outside the reconstructed surface, but inside or across the reconstructed surface of the model the are no significant differences. Hence, we can state that the *Conjugate Gradient* is the best choice since it outperforms the other ones in terms of execution time.

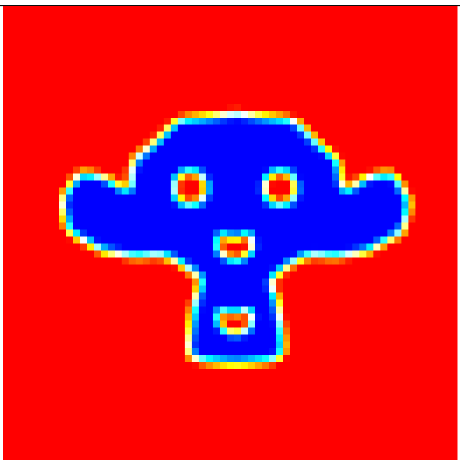
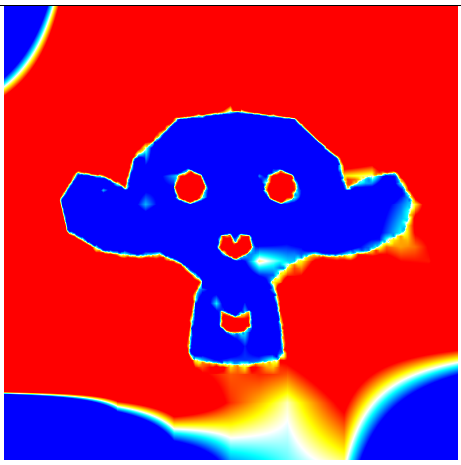
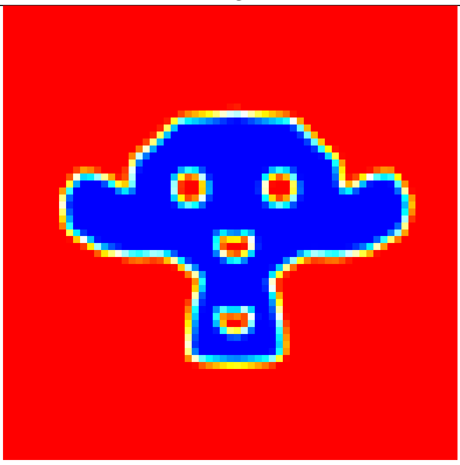
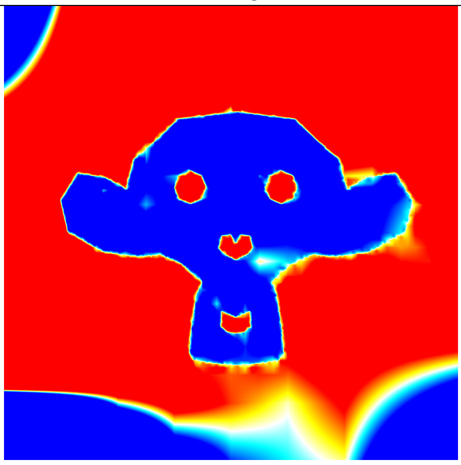
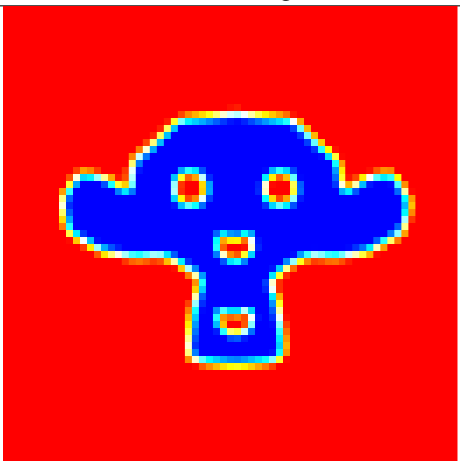
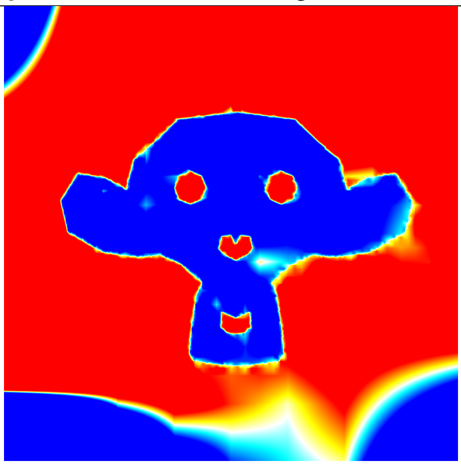
 Reconstructed surface output at 64px. for the *BICGSTAB* solver using *Biharmonic* method.	 Reconstructed surface output at 1024px. for the *BICGSTAB* solver using *Quadtree* method.
 Reconstructed surface output at 64px. for the *ConjugateGradient* solver using *Biharmonic* method.	 Reconstructed surface output at 1024px. for the *ConjugateGradient* solver using *Quadtree* method.
 Reconstructed surface output at 64px. for the *LeastSquares* solver using *Biharmonic* method.	 Reconstructed surface output at 1024px. for the *LeastSquares* solver using *Quadtree* method.

Figure 6.17: Images obtained from reconstructing *Suzane* model with the BICGSTAB, ConjugateGradient, and LeastSquaresConjugateGradient iterative linear solvers with the Biharmonic and Quadtree methods.

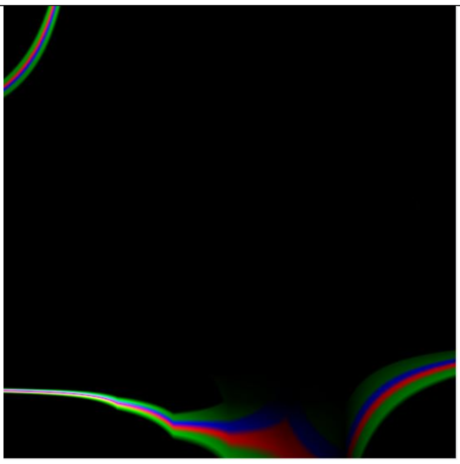
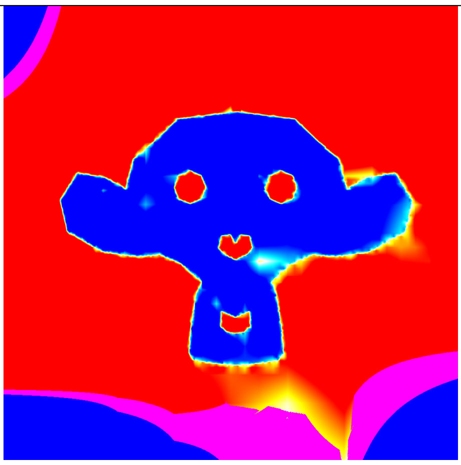
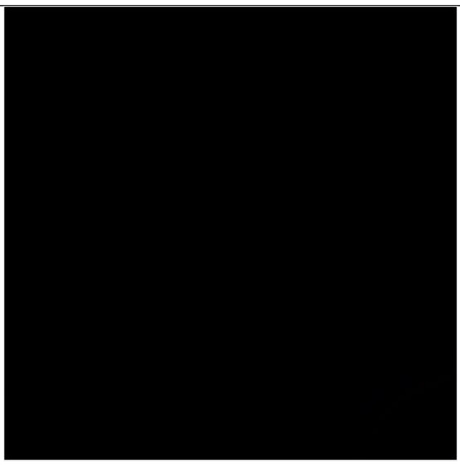
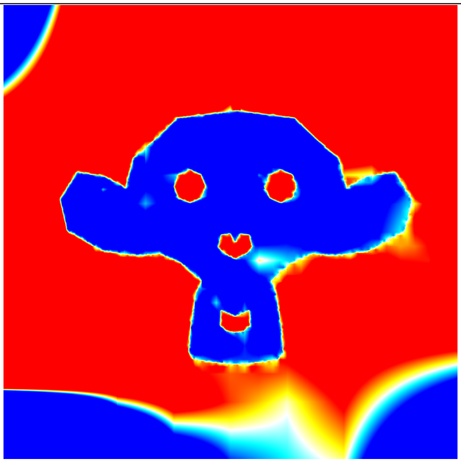
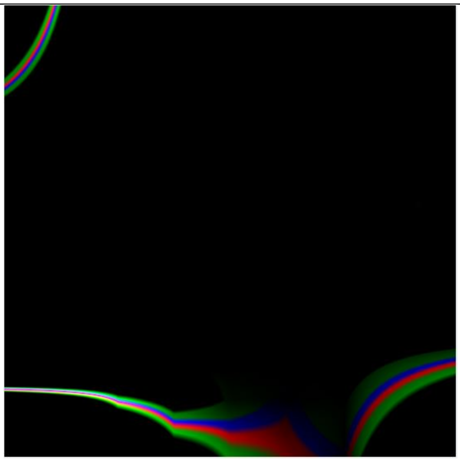
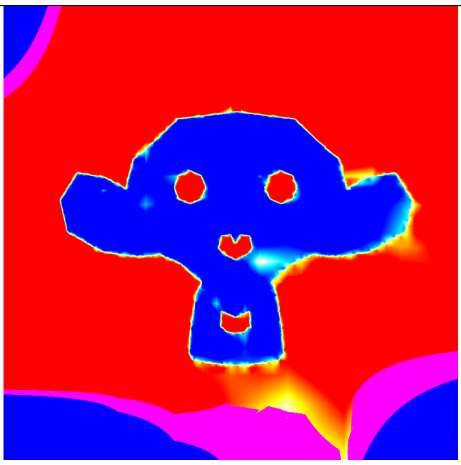
 Image showing the main differences represented in colorscale of using *BICGSTAB* and *ConjugateGradient* solvers with the *Quadtree* method.	 Image showing the main differences represented with pink spots of using *BICGSTAB* and *ConjugateGradient* solvers with the *Quadtree* method.
 Image showing the main differences represented in colorscale of using *ConjugateGradient* and *LeastSquares* solvers with the *Quadtree* method.	 Image showing the main differences represented with pink spots of using *ConjugateGradient* and *LeastSquares* solvers with the *Quadtree* method.
 Image showing the main differences represented in colorscale of using *BICGSTAB* and *LeastSquares* solvers with the *Quadtree* method.	 Image showing the main differences represented with pink spots of using *BICGSTAB* and *LeastSquares* solvers with the *Quadtree* method.

Figure 6.18: Images representing the main differences obtained by representing *Suzane* model with the different iterative solvers with the Quadtree method.

6.3.4 Analysis of the performance of the Multigrid

In this section, we will perform a depth analysis of the performance in terms of execution time and quality of the resultant reconstructed surface of the multigrid against the conventional Biharmonic solver. As we talked about in the chapter 4.7, the multigrid should improve the performance time of resolving the iterative linear system, since at each integration we will introduce a *"Guess"* input as the initial solution. However, we don't know how it will impact the performance of the resultant solution in terms of quality and error. Furthermore, we will iterate many times and we don't know if, in the end, it will be more time-consuming.

For this study, we have fixed the normal equations to *Sampling* and the smoothing equations to *1D*. We also fixed the number of threads to **8**. For the multigrid cases, we will also analyze the performance of using a different number of iterations to reach the desired resolution, which will be the same as the resolution computed by the conventional *Biharmonic* solver. Moreover, we have run the executable **25** times to obtain the average of the execution time for each iteration:

Resolution	Num Eq	Num Un	Iter	Sys Time (ms)	Error	Rel Error
129	34926	16641	11328	43965,2	9,86181e-06	0,361221

Table 6.8: Content table of average results for the surface reconstruction analysis **without** multigrid obtained with the *Biharmonic* method for *Suzane* model. The Total times are represented in milliseconds. The field size used for the surface reconstruction was **129**.

Iter $n^{\underline{o}}$	Resolution	Num Eq	Num Un	Iter	Total Time (ms)	Error	Rel Error
0	65	10350	4225	2790	2714,4	9,05617e-06	0,365539
1	129	34926	16641	5124	18923,5	9,98973e-06	0,361221

Average Total Time for all Multigrid iterations: 21637,9 ms.

Table 6.9: Content table of average results for the surface reconstruction analysis **using** multigrid obtained with the *Biharmonic* method for *Suzane* model. The multigrid algorithm performed **1** iteration. The Total times are represented in milliseconds. The resulted field size of surface reconstruction was **129**.

Iter $n^{\underline{o}}$	Resolution	Num Eq	Num Un	Iter	Total Time (ms)	Error	Rel Error
0	33	4206	1089	714	357,7	8,4196e-06	0,397721
1	65	10350	4225	1288	1271,1	9,9957e-06	0,365539
2	129	34926	16641	5180	18923,5	9,9893e-06	0,361221

Average Total Time for all Multigrid iterations: 20473,1 ms.

Table 6.10: Content table of average results for the surface reconstruction analysis **using** multigrid obtained with the *Biharmonic* method for *Suzane* model. The multigrid algorithm performed **2** iteration. The Total times are represented in milliseconds. The resulted field size of surface reconstruction was **129**.

Iter $n^{\underline{o}}$	Resolution	Num Eq	Num Un	Iter	Total Time (ms)	Error	Rel Error
0	17	2670	289	175	2714,4	9,0329e-06	0,565278
1	33	4206	1089	527	272,4	9,8383e-06	0,397721
2	65	10350	4225	1290	1338,3	9,9830e-06	0,365539
3	129	34926	16641	5182	18814,2	9,8971e-06	0,361221

Average Total Time for all Multigrid iterations: 20464 ms.

Table 6.11: Content table of average results for the surface reconstruction analysis **using** multigrid obtained with the *Biharmonic* method for *Suzane* model. The multigrid algorithm performed **3** iterations. The Total times are represented in milliseconds. The resulted field size of surface reconstruction was **129**.

As we can observe on the tables 6.8, 6.9, 6.10 and 6.11 the multigrid outperformed the conventional biharmonic solver in terms of execution time. The results showed than the multigrid can reach with a feasible solution with half of the execution times than needs the biharmonic solver.

6.3.5 Analysis of the performance of the Multithreating

In the section 4.5 we talked about the possibility to exploit the multiple cores present in the hardware in order to increase the performance of the iterative linear solvers in terms of execution time. This chapter aims to analyze the performance of the obtained results in terms of execution time when we try different values for the threads used during the system solving. For this study, we have fixed the smoothing equations to *1D* and the iterative normal equations to *Sampling*. We also used the *Conjugate Gradient* iterative solver. Moreover, we have run the executable **25** times to obtain the average of the execution time for each iteration.

Num Threads	Num Eq	Num Unknowns	Iterations	System Time ms
24	22221	10046	1345	5517,4
22	22221	10046	1345	4253,8
20	22221	10046	1345	3576,8
18	22221	10046	1345	3552,4
16	22221	10046	1345	3423,6
14	22221	10046	1345	3351,6
12	22221	10046	1345	3345,4
10	22221	10046	1345	3391,6
8	22221	10046	1345	3523.8
6	22221	10046	1345	3708,2
4	22221	10046	1345	4238,8
2	22221	10046	1345	5468,8

Table 6.12: Content table of average results for multhitreading analysis obtained with the *Biharmonic* method for *Suzane* model. The Total times are represented in milliseconds. The field size used for the surface reconstruction was **1025**.

If we take a look at the table 6.12 and chart figure 6.19 we can see the performance difference than we obtain if we use a different number of threads for the iterative linear solver. As we expected using a low number of threads is not the best choice since the solving calculations could be improved in terms of parallelism, having just a few threads for computing a large number of matrix chunks it's not cost-efficient.

However, if have a high value of threads, the obtained are also bad, even could be higher than using a low number of threads. This is because each thread will have a fewer number of matrix chunks and be more expensive to communicate between them and put the resulted data in common rather than computing its respective calculus.

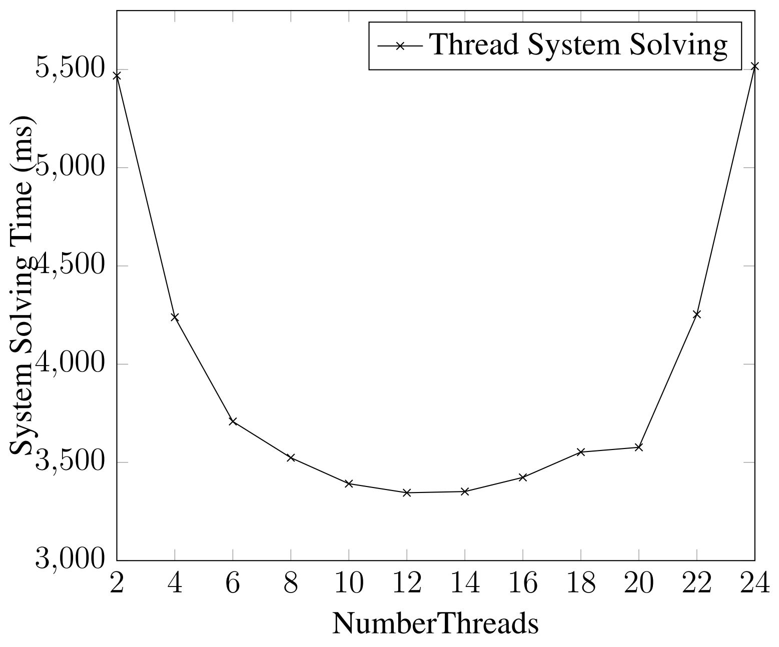


Figure 6.19: Line plot chart showing the linear system resolution time performance of the surface reconstruction for field of resolution **1025**.

Last but not least, we must take into account that the optimal number of threads is directly related to the size of the build-in equation system and the number of equations and number of unknowns we introduce to the iterative linear system. A clear example can be observed at the below table 6.13 and the plot chart 6.20. For a system with a low number of equations and unknowns, the optimal number of threads will be on small values rather than using a bigger number of threads.

Num Threads	Num Eq	Num Unknowns	Iterations	System Time ms
24	3843	857	97	53
22	3843	857	97	49,6
20	3843	857	97	49,4
18	3843	857	97	48,2
16	3843	857	97	47,8
14	3843	857	97	47,9
12	3843	857	97	47,4
10	3843	857	97	47,6
8	3843	857	97	47
6	3843	857	97	46,8
4	3843	857	97	46,9
2	3843	857	97	45,2

Table 6.13: Content table of average results for multhitreading analysis obtained with the *Biharmonic* method using *Sampling* normal equations and *1D* smoothing. The Total times are represented in milliseconds. The field size used for the surface reconstruction was **65**.

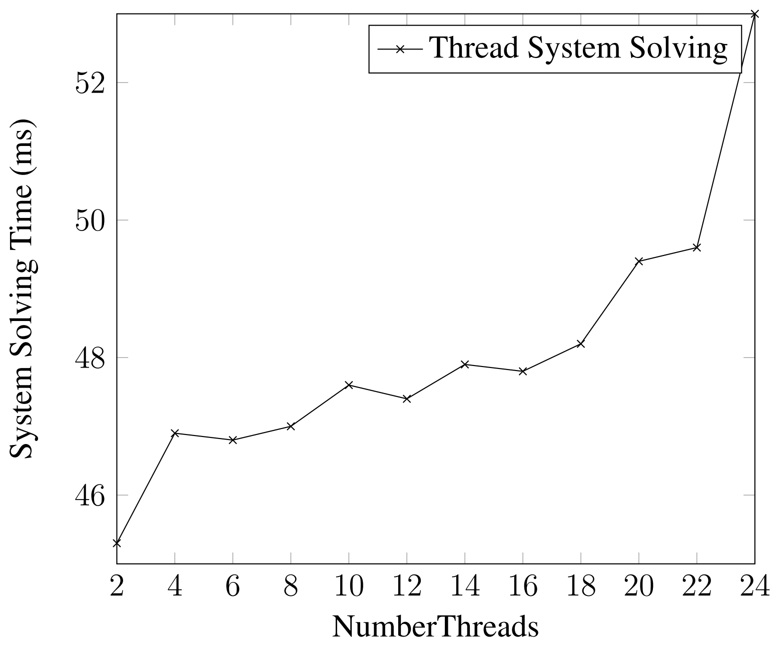


Figure 6.20: Line plot chart showing the linear system resolution time performance of the surface reconstruction for field of resolution **65**.

6.4 Analysis of the performance of the Biharmonic vs Quadtree Solvers

In the section 5 we talked about the *Quadtree* and the capability to partition a two-dimensional space by recursively subdividing it into four quadrants or regions. Unlike the *Biharmonic* system approach, the *Quadtree* structure will only store information of the node corners of the cells which contains a point data of the cloud. The aim of this experiment is to compare the performance of the *Quadtree* against the *Biharmonic* approach in terms of execution time and quality of the resulting from the reconstructed surface. For this study, we have fixed the normal equations to *Sampling*, the smoothing equations to *1D* and the iterative solver to *Conjugate Gradient*. We also fixed the number of threads to *8*. Moreover, we have run the executable *25* times to obtain the average of the execution time for each iteration:

Resol	Equations	Unknowns	Sys (ms)	Mat (ms)	Solver (ms)	Total (ms)
17	2670	289	10,3	3,9	4	18,3
33	2097	389	12,1	74.6	338.4	358,1
65	10350	4225	26,9	30,6	2533,2	2610,5
129	34926	16641	578,4	286,7	42533,9	43965,2

Table 6.14: Content table of average time results obtained for the *Biharmonic* method for different resolutions using the *Suzane* model. The times are represented in milliseconds.

Resol	Iterations	Error	Rel. Error
17	38	7,049e-06	0,651502
33	58	9,3947e-06	0,459492
65	2790	9,6823e-06	0,365539
129	11328	9,86181e-06	0,361221

Table 6.15: Content table of the error results obtained for the *Biharmonic* method for different resolutions using the *Suzane* model.

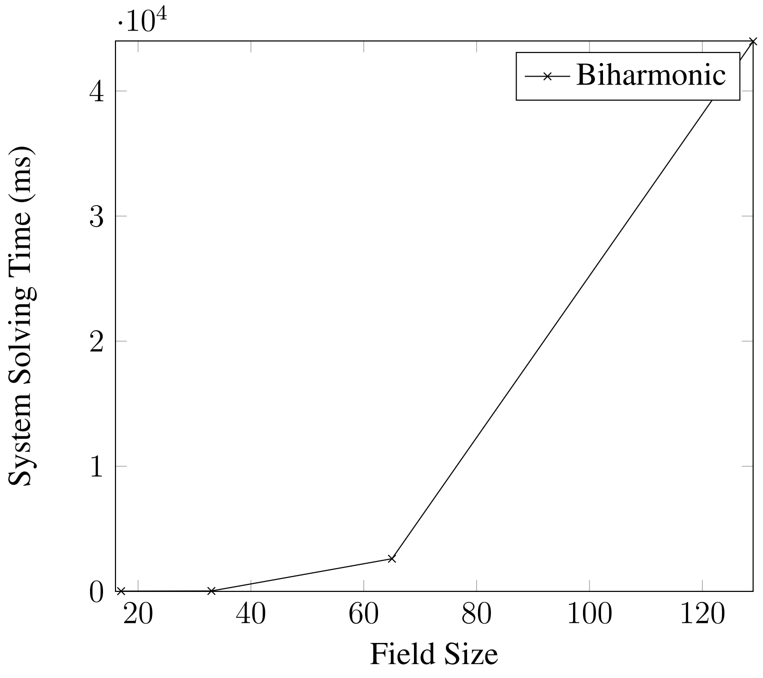


Figure 6.21: Line plot chart showing the total execution time using the *Biharmonic* system for different fields of different resolutions.

Resol	Eq	Unk	Sys (ms)	Mat (ms)	Solver (ms)	Total (ms)
17	2463	167	8,4	4	25,8	38,2
33	2907	389	12	7,4	338,6	358
65	3843	857	21,2	8,7	52,9	82,8
129	5737	1804	37,1	24,3	121,7	183,1
257	9411	3641	70,8	38,1	353,4	462,3
513	15257	6564	131,1	57,8	1221,4	1410,3
1025	22221	10046	195,3	77,2	3251,3	3523,8
2049	29379	13625	257,8	99	8971,1	9327,9
4097	36549	17210	308,6	127,2	20804	21239,8

Table 6.16: Content table of average times obtained for the *Quadtree* for different resolutions using the *Suzane* model. The System, Matrix, Solver, and Total times are represented in milliseconds.

Resol	Iterations	Error	Rel Error
17	38	7,049e-06	0,651502
33	58	9,3947e-06	0,459492
65	97	9,6111e-06	0,420518
129	204	9,7214e-06	0,468835
257	367	9,8558e-06	0,537750
513	759	9,5630e-06	0,632958
1025	1345	9,6823e-06	0,730780
2049	2679	9,7429e-06	0,771719
4097	4989	9,7688e-06	0,781172

Table 6.17: Content table of average results obtained for the *Quadtree* for different resolutions using the *Suzane* model.

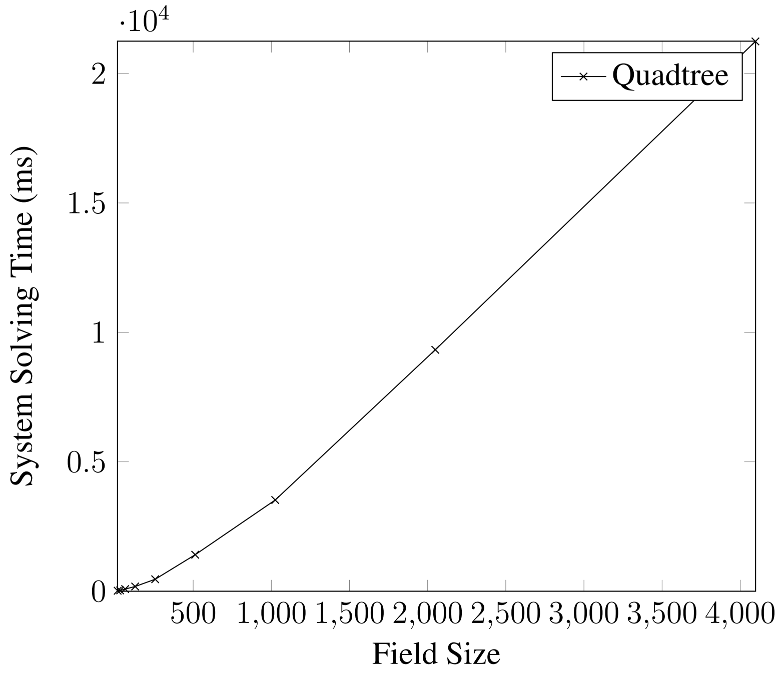


Figure 6.22: Line plot chart showing the total execution time using the *Quadtree* system for different fields of different resolutions.

As we can appreciate on the tables 6.14, 6.15, 6.16, and 6.17 the *Quadtree* system outperforms the *Biharmonic* system in terms of execution time. If we take a look at the plot 6.21 we can see that when we exceed the field resolution of 65 approximately, the slope of the chart significantly grows.

Moreover, if we take a look a the plot 6.23 we can visually appreciate the big difference between the two different approaches. The difference is so huge that reconstructing a surface with a field size of 129 with the *Biharmonic* method takes about the double of time as reconstructing a surface with a field size of 4097 using the *Quadtree* approach. Obtaining high-resolution reconstructions with the *Biharmonic* method is unfeasible.

However, with the *Quadtree* approach we can obtain high-resolution images using the *Quadtree* system.

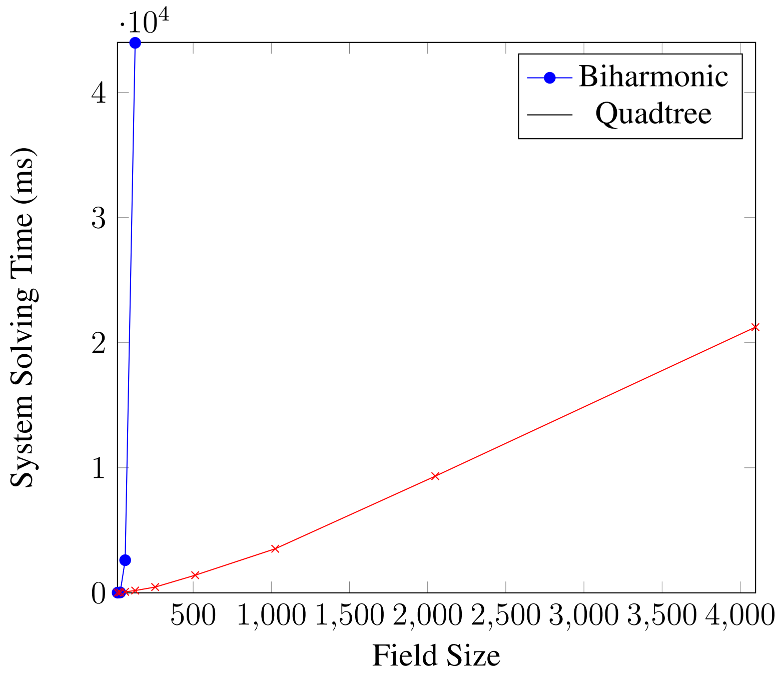


Figure 6.23: Line plot chart showing comparison of the total execution time using the *Biharmonic* system against the *Quadtree* system for different fields of different resolutions.

Nonetheless, if we refer to the quality of the resultant reconstructed surface, we can state that the *Biharmonic* system will produce better quality solutions compared to the *Quadtree* approach. Not only produces solutions with lesser relative error but if we take a look at the chart plots 6.24 and 6.25 we can see that at the time we increase the field size using the *Biharmonic* system, the relative error decreases. However, at the time we increase the field resolution on the *Quadtree* approach, the relative error increases too.

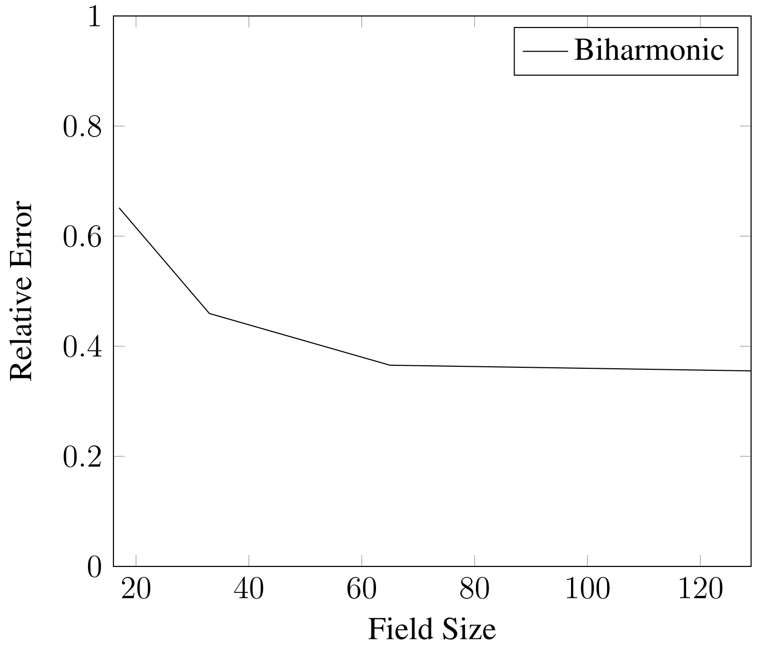


Figure 6.24: Line plot chart showing the relative error using the *Biharmonic* system for different fields of different resolutions.

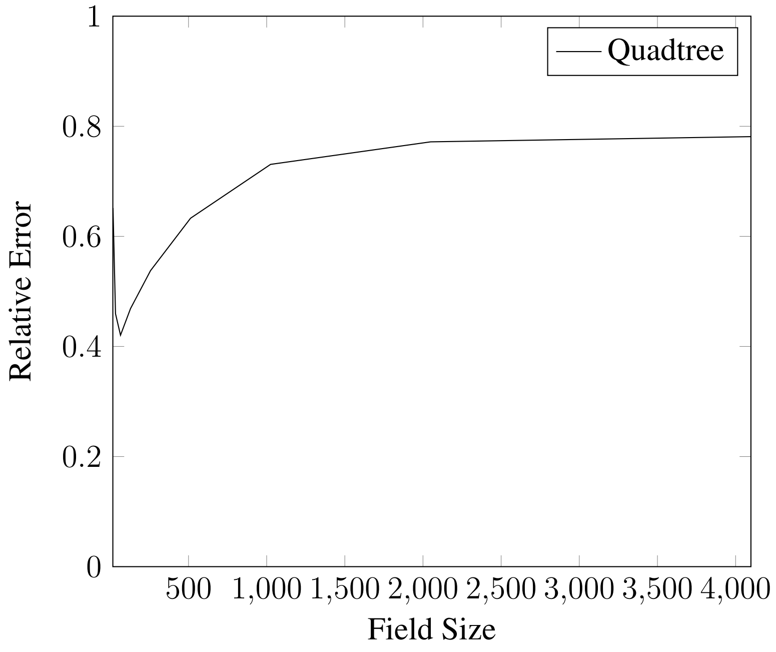


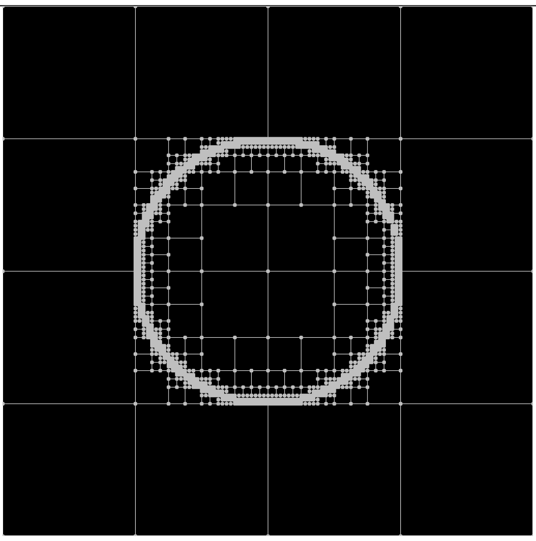
Figure 6.25: Line plot chart showing the relative error using the *Quadtree* system for different fields of different resolutions.

6.5 The importance of properly aligning the model to the field

As we know, the *Quadtree* reconstruction will not subdivide the space field with subdivisions of equal size. On the contrary of the *Biharmonic* solver, which will subdivide the all field into quadrants subdivisions of the same unitary size, the size of the subdivisions of the *Quadtree* may vary depending on the point information and in which section of the field is falling too. Therefore, the obtained subdivided grid may not be perfectly equal or symmetrical.

This matter could drastically impact the quality and the results of our reconstructed surface. Placing the same model in a different position will mean that the resulted grid will be different. Moreover, depending on the number of subdivision levels that the *Quadtree* has fulfilled on the field space, the difference could be even bigger.

Finally, we can discern the models between symmetrical and non-symmetrical. For the non-symmetrical, it would be inevitable to have equally symmetrical subdivisions on the X-Y Cartesian coordinates that would split the field. However, for the symmetrical models that are not equally subdivided in, the resulting output could be disastrous, provoking to the generated output undesired noise and bubbles.


Subdivisions obtained by perfectly aligning the *Circle* model to the gradient field.

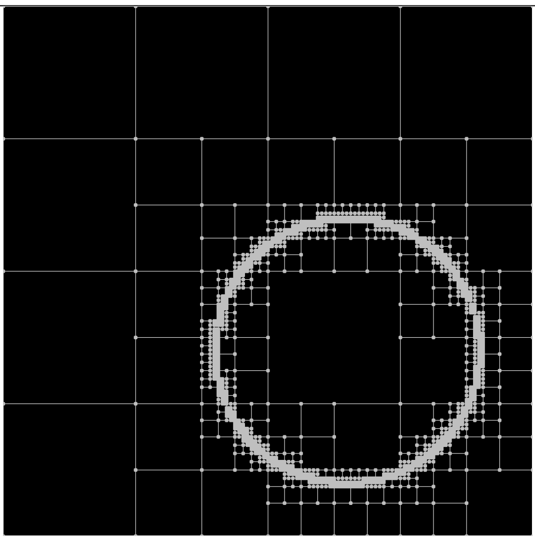

Subdivisions obtained by displacing 0.15 units the *Circle* model to the gradient field.

Figure 6.26: Images showing the subdivisions representations of the *Quadtree* model with a depth level of 2^{10}. A small displacement means a significant difference between both subdivisions.

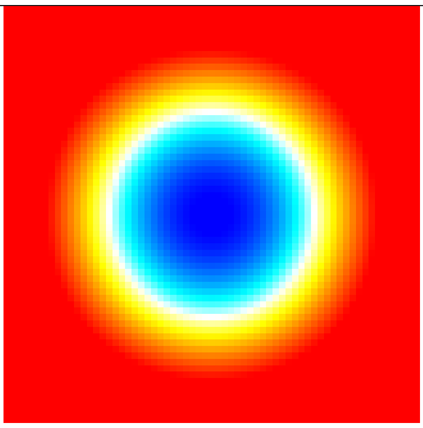
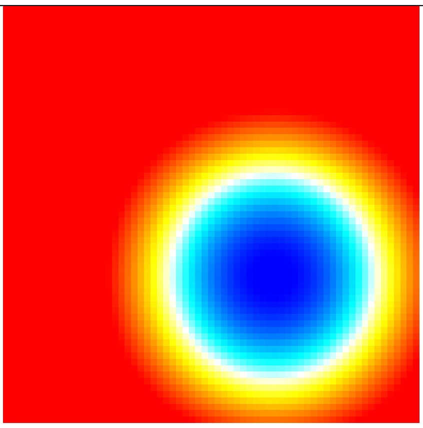
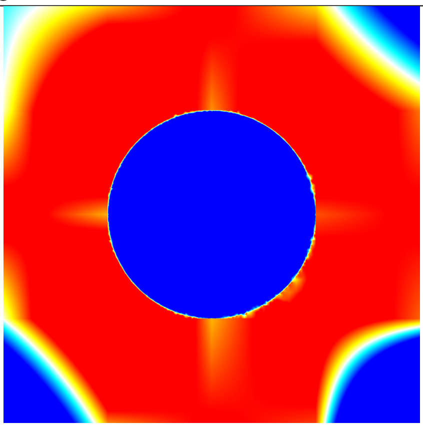
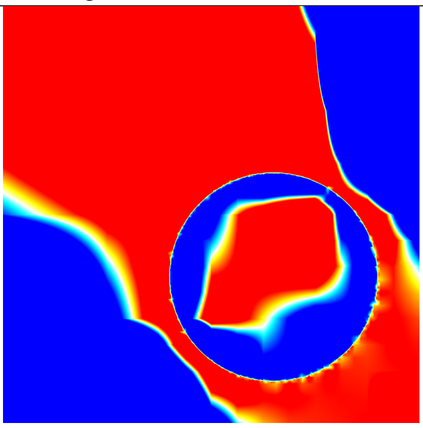
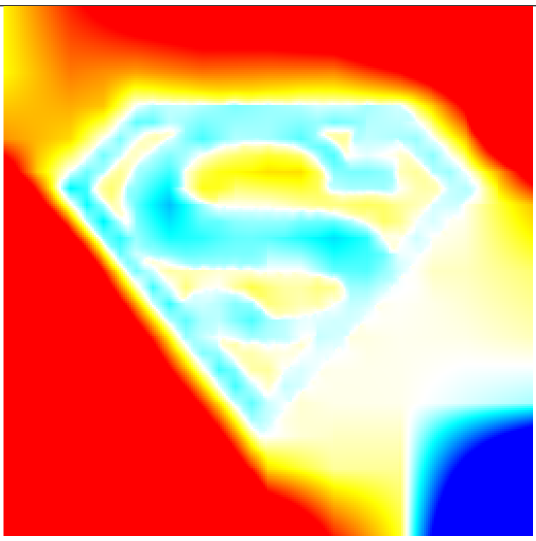
 Reconstructed surface output at 64px. of the *Circle* model aligned to the field using the *Biharmonic* reconstruction method.	 Reconstructed surface output at 64px. of the *Circle* model unaligned to the field. using the *Biharmonic* method.
 Reconstructed surface output at 1024px. of the *Circle* model aligned to the field using the *Quadtree* reconstruction method.	 Reconstructed surface output at 1024px. of the *Circle* model unaligned to the field. using the *Quadtree* method.

Figure 6.27: Images obtained from reconstructing *Circle* model with the Gradient and Sampling system equations with the Biharmonic and Quadtree methods.

Despite the resultant surface reconstruction of the *Biharmonic* solver being perfectly equal, we can appreciate on the figure 6.27 that the reconstructed surface outputs obtained by using the *Quadtree* method are different. The reconstruction with a centered symmetrical model such as the *Sphere* model obtained some noise on the corners of the field, which is irrelevant and could be discarded very easily. However, the noise obtained by reconstructing the *Sphere* model with light displacement concerning the field center provokes a substantial impact on the obtained reconstructed surface. Not only does it provoke an inner hole inside the reconstructed surface but also the corner generated bigger bubbles which are pretty close to the reconstructed surface outline. Those two noise artifacts could cause struggles to generate a quality representation of the surface.

6.6 Additional Results

Now, we will proceed to show some of the additional reconstructed results obtained by running our executable with different inputs and configurations:


Surface reconstruction using the *Quadtree* system with *Gradient* normal eq. The total execution time was 875 ms. The relative error is 0,847308.

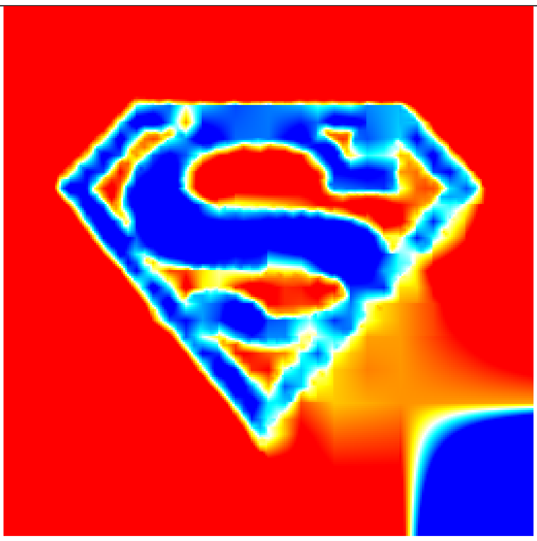
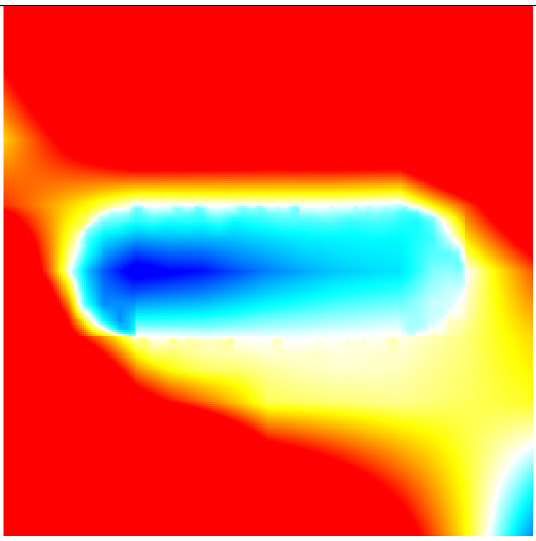

Surface reconstruction using the *Quadtree* system with *Sampling* normal eq. The total execution time was 653 ms. The relative error is 0,757691.

Figure 6.28: Comparison of the performance of the Gradient vs. Samping normal equations on the *Superman* model with a field resolution of **256**.


Surface reconstruction using the *Quadtree* system with *Gradient* normal eq. The total execution time was 119 ms. The relative error is 0,758942.

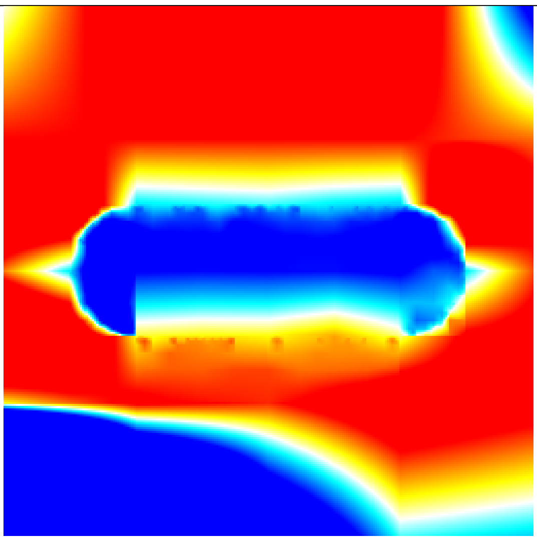
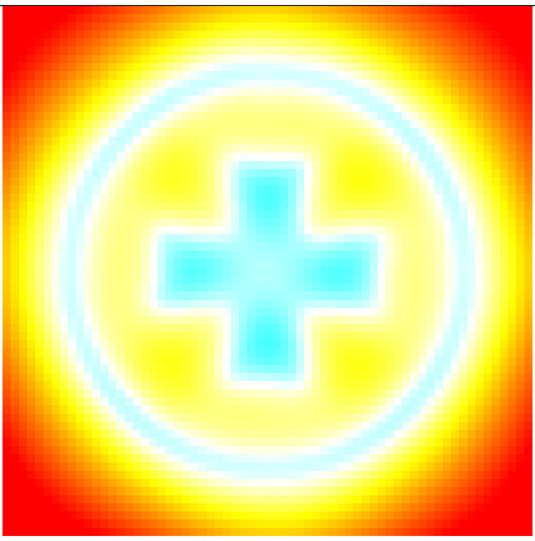

Surface reconstruction using the *Quadtree* system with *Sampling* normal eq. The total execution time was 71 ms. The relative error is 0,547536.

Figure 6.29: Comparison of the performance of the Gradient vs. Samping normal equations on the *Capsule* low resolution model with a field resolution of **256**.


Surface reconstruction using the *Biharmonic* system with *1D* smoothing eq. The total execution time was 1144 ms. The relative error is 0,703391.

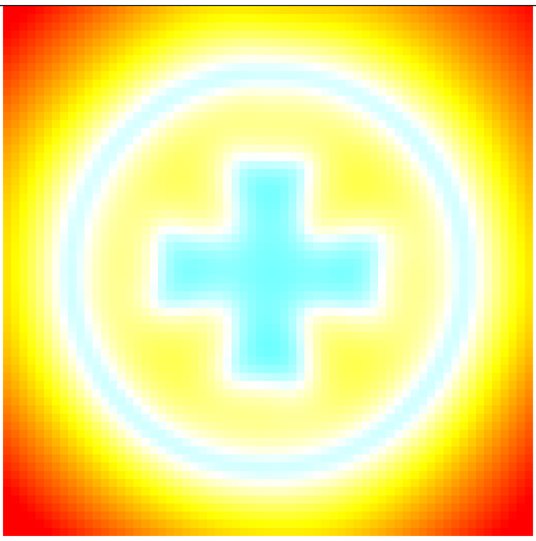
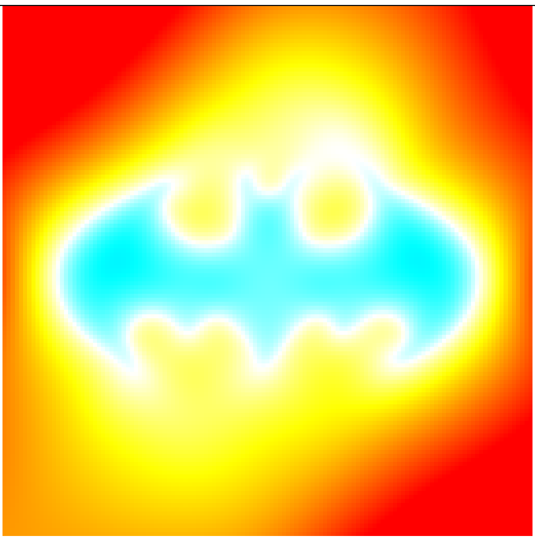

Surface reconstruction using the *Biharmonic* system with *2D* smoothing eq. The total execution time was 1844 ms. The relative error is 0,756499.

Figure 6.30: Comparison of the performance of the 1D vs. 2D smoothing equations on the *Pharmacy Cross* model with a field resolution of **65**.


Surface reconstruction using the *Biharmonic* system with *1D* smoothing eq. The total execution time was 2782 ms. The relative error is 0,596682.

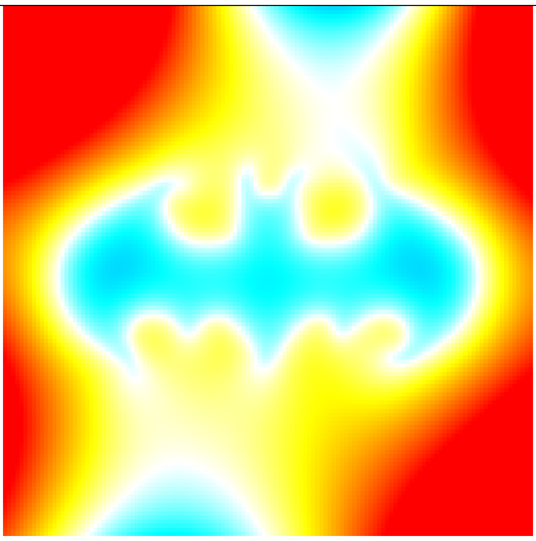

Surface reconstruction using the *Biharmonic* system with *2D* smoothing eq. The total execution time was 3870 ms. The relative error is 0,642997.

Figure 6.31: Comparison of the performance of the 1D vs. 2D smoothing equations on the *Batman* model with a field resolution of **65**.

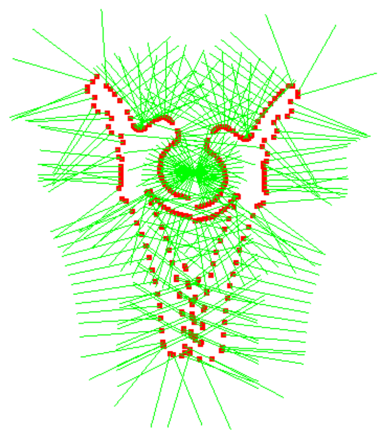
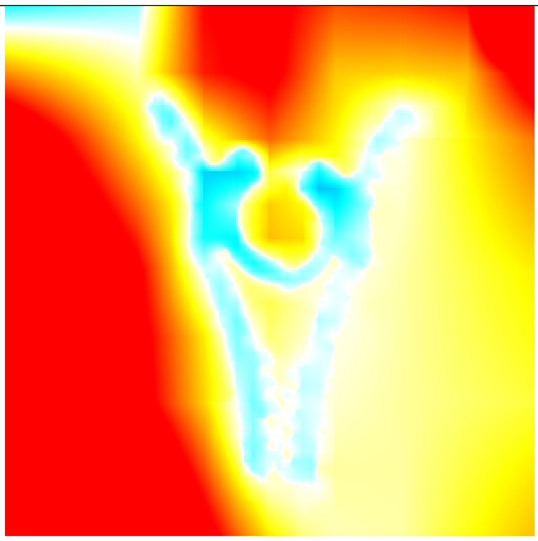
	
Point Cloud viewer with data displacement.	Surface reconstruction using the *Quadtree* system with *1D* smoothing eq. The total execution time was 358 ms. The relative error is 0,81699.

Figure 6.32: Impact of sampling displacement when reconstructing the *Crocodile Clip* model using the *Quadtree* method with *gradient* normal equations and a field resolution of **257**.

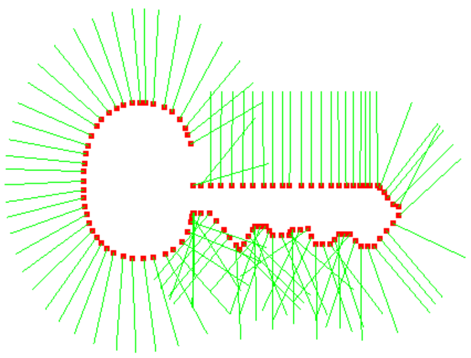
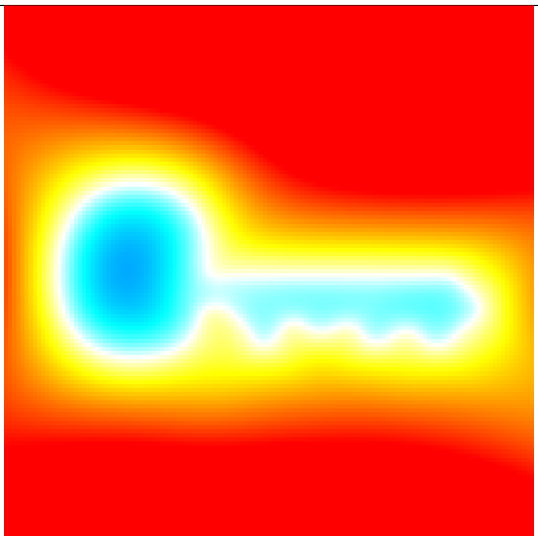
	
Point Cloud viewer with data displacement.	Surface reconstruction using the *Biharmonic* system with *1D* smoothing eq. The total execution time was 3317 ms. The relative error is 0,57324.

Figure 6.33: Impact of sampling displacement when reconstructing the *Key* model using the *Biharmonic* method with *gradient* normal equations and a field resolution of **65**.

Chapter 7

Conclusions

7.1 Conclusions

In this project, we addressed the problem of surface reconstruction by introducing a variational formulation for the problem of reconstructing a watertight surface defined by an implicit equation from a finite set of oriented points. We built an application that not only solves different sparse linear systems of equations using different possible methods for position, normal, and smoothness equation's constraints but also makes use of more complex and effective surface reconstruction solving techniques such as the *Multigrid* or *Quadree* reconstruction. Those implemented methods highlight the strengths and limitations that currently exist in the field, making potential connections across input assumptions, point cloud properties, and implemented approaches that have not been previously considered.

Despite *Bilaplacian* smoothness was meant to enforce a smooth prior to the surface reconstruction, the obtained results were not as expected, presenting a higher execution time and relative error compared to the *Laplacian* smoothness.

The *Quadree* method outperforms the *Biharmonic* approach in terms of execution time, allowing to reconstruct the watertight surface with higher resolutions. However, the relative error is lower using the *Biharmonic* approach, providing more precise reconstructions.

The optimal number of threads to use in each of the surface´s reconstruction is directly related to the number of points of the cloud. A huge number of threads could provoke overhead noise and significaly drop the performance.

7.2 Future Prospects of Our Work

Many different adaptations, tests, and experiments have been left for the future due to lack of time (i.e. the experiments with data are usually very time-consuming, requiring even hours to finish a single run). Future work concerns deeper analysis of particular algorithms, new proposals to try different methods, or simply curiosity.

There are some ideas that I would have liked to try during the description and the development of the fitness functions in Chapter 4. This thesis has been mainly focused on the surface reconstruction of point clouds for two-dimensional models, leaving the study of third-dimensional ones outside the scope of the thesis.

Despite the surface reconstruction of the Quadtree approach outperformed the Biharmonics approach in terms of execution time, the quality of the presented reconstructed surface of the Quadtree leaves much to be desired. The quality of the reconstructed surface of the Biharmonic approach was smooth and polished and seemed to less point dependant on the quantity and quality of the sampled points, whereas the reconstructed surface of the Quadtree presented few noise and bubbles which impacted the output quality, especially when the mesh was not perfectly centered or the subdivisions were not symmetrical. The following ideas could be tested to improve the quality of the Quadtree:

1. One possible solution in order to avoid the noise bubbles to be merged with the reconstructed surface could be to create a series of extra unitary subdivisions during the Quadtree subdivision around the contour of the surface to generate more accuracy on the surface´s reconstructed limits. This way the contrast between big subdivisions and small unitary subdivisions which contain point information will be decreased and separated and hence reducing the chances of merging the noise bubble with the reconstructed surface. Moreover, the artifacts that appear inside the reconstructed surface could be drastically reduced.

2. Another possible optimization that could improve the quality of the results could be to perform a cubical interpolation instead of linear interpolation. In general, cubic interpolation is better than linear interpolation in most aspects such as smoothness of the function and higher accuracy in approximating the original function.

   However, there is at least one aspect where linear interpolation is better: the linear interpolation will not produce the "overshoot" situation. For example, if we have a function that always produces positive y values, using cubic interpolation to approximate this function might give a function that "overshoots" to negative *y*-axis territory even when all the interpolated *y* values are positive. This fact must be taken into account at the time we perform our interpolations since we don't want to generate negative values.

Last but not least, during the performance of our *Quadtree* experiments we have detected some kind of bubble noise, especially at the diagonal corners. This could be possible neglect of the system equations (despite we have meticulously revised the quality of the already implemented ones) or some rule or corner case we forgot to take into account during the implementation of the method. Despite they being meticulously revised, as future work, it would be a good idea to revise that bubbling noise on the diagonal corners.

References

[1] F. Calakli and G. Taubin, "Ssd: Smooth signed distance surface reconstruction," in *Computer Graphics Forum*, vol. 30, no. 7. Wiley Online Library, 2011, pp. 1993–2002.

[2] G. Taubin, "Smooth signed distance surface reconstruction and applications," in *Iberoamerican Congress on Pattern Recognition*. Springer, 2012, pp. 38–45.

[3] R. D. Falgout, "An introduction to algebraic multigrid," 2006.

[4] M. Berger, A. Tagliasacchi, L. Seversky, P. Alliez, J. Levine, A. Sharf, and C. Silva, "State of the art in surface reconstruction from point clouds," in *Eurographics 2014-State of the Art Reports*, vol. 1, no. 1, 2014, pp. 161–185.

[5] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva, "A survey of surface reconstruction from point clouds," in *Computer Graphics Forum*, vol. 36, no. 1. Wiley Online Library, 2017, pp. 301–329.

[6] Y. Marchand, B. Vallet, and L. Caraffa, "Evaluating surface mesh reconstruction of open scenes," vol. 43. Copernicus GmbH, 2021, pp. 369–376.

[7] C. C. You, S. P. Lim, S. C. Lim, J. San Tan, C. K. Lee, and Y. M. J. Khaw, "A survey on surface reconstruction techniques for structured and unstructured data," in *2020 IEEE Conference on Open Systems (ICOS)*. IEEE, 2020, pp. 37–42.

[8] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, 2006.

[9] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," vol. 32, no. 3. ACM New York, NY, USA, 2013, pp. 1–13.

[10] V. Estellers, M. Scott, K. Tew, and S. Soatto, “Robust poisson surface reconstruction,” in *International Conference on Scale Space and Variational Methods in Computer Vision*. Springer, 2015, pp. 525–537.

[11] P. J. Huber, “Robust estimation of a location parameter,” in *Breakthroughs in statistics*. Springer, 1992, pp. 492–518.

[12] T. Zhao, P. Alliez, T. Boubekeur, L. Busé, and Jean-Marc Thiery, “Progressive discrete domains for implicit surface reconstruction,” in *Computer Graphics Forum*, vol. 40, no. 5. Wiley Online Library, 2021, pp. 143–156.

[13] S. Cuomo, A. Galletti, G. Giunta, and L. Marcellino, “Reconstruction of implicit curves and surfaces via rbf interpolation,” vol. 116. Elsevier, 2017, pp. 157–171.

[14] R. A. Finkel and J. L. Bentley, “Quad trees a data structure for retrieval on composite keys,” vol. 4, no. 1. Springer, 1974, pp. 1–9.

[15] P. Wesseling, “Introduction to multigrid methods.” 1995.

[16] H. Samet, “Foundations of multidimensional and metric data structures,” 2006.

[17] G. Guennebaud, B. Jacob *et al.*, “Eigen v3,” http://eigen.tuxfamily.org, 2010.

[18] A. van den Bos, “Appendix c: Positive semidefinite and positive definite matrices,” March.

Appendix A

Description of the executable

Our presented executable is meant to be a complete environment for testing not only the performance of the algorithms and iterative solvers but also to analyze the impact of the noise and artifacts. Here is a quick overview of the executable usage:

Usage:

reconstruction.exe [-f | -b] [Surface Reconstruction | Model Generation]

- The flag *-f* is meant to activate the Surface Reconstruction Mode. It will require the path to the *.svg* or the *.txt* file. If the inputed is an svg extension the program will parse the file and generate the proper .txt file in order to make the reconstruction. If the inputed file is the .txt file it will start the surface reconstruction with the default parameters established.
- The flag -b is used in case the user it has neither *.svg* nor *.txt* files. In this case the executable will generate a basic *.txt* geometric model in order to proceed with the analysis.

Optional Parameters:

[-c | Which Methods will Execute (Biharmonic, Quadtree, or Both)]

- This flag specifies which solvers we want to execute. If we choose *Biharmonic* or *Quadtree* it will only execute the Biharmonic or the Quadtree reconstructors respectively. If the input is *Both* it will execute first execute the Biharmonic reconstruction and then it will proceed with the Quadtree reconstruction. By default, the selected choice will be *Both*.

[-n | normal's size]

- This flag is used to specify the length size of the normals represented on the main viewer window. We have to take into account that relative positions are normalized to one. This way, we suggest keeping small values for the normals size since it could provoke to generate an incredible huge normal that could reach the out of the screen, covering the representation of the points. By default, the normal size is established to 0.2.

[-z | Gaussian noise (between 0-0.025 recommended)]

- This flag is meant to generate Gaussian noise and displace the relative position of the points. All the displacements are relative to the normal's direction. This optional flag is meant to test the impact of noise and outliers artifacts in our surface reconstruction. Applying a big Gaussian value could cause a big displacement disfiguring the aiming model. By default, the Gaussian noise will be set to 0. We recommend not to exceed a value of 0.025.

[-d | Sampling Density (between 0-100)]

- This flag is meant to test sampling density and missing data artifacts, undersampling the number number of points of the point cloud. The value is tan percent and it means the probability that a point *p* is taken into account for the surface reconstruction. When a point falls out of the probability it will not be represented as well as it will not represent a few numbers of the upcoming ones which are chosen randomly to generate a missing data chunk. By default, the sampling density will be set to 100 percent (a point will always be taken into account for the surface representation).

[-r | Resolution]

- This flag means the resolution of the scalar field of the resulted reconstructed surface. We recommend to input values of $2^k + 1$ were *k* is a natural number $\mathbb{N}_1 = \{1, 2, 3, ...\}$, specially if we are using the *Multigrid* solving. By default it will have a resolution of 65.

[-a | normal algorithm (gradient or sampling)]

- This flag specifies which kind of normal equations we will add to our solver, *gradient* equations or sampling equations for our reconstruction. By default it will be set to *gradient*.

[-s | smoothing algorithm (1D or 2D)]

- This flag specifies which kind of smoothing equations we will add to our solve, one dimensional equations *(1D)* or two dimentional equations *(2D)*. By default it will be set to *2D*.

[-x | solver method (Conjugate, Biconjugate or LeastSquares)]

- This flag specifies which iterative solver it will be used to solve or equation system, the *Conjugate Gradient*, the *Biconjugate Gradient* or the *Least Squares Conjugate Gradient*. By default, it will be set to *Conjugate Gradient (Conjugate)*.

[-m | Multigrid Solving Mode on (0 or 1)]

- This flag specifies if the surface reconstruction will be made through a Multigrid or not. Note that this flag will **only** take into account if we are making a *Biharmonic* solving. By default, its value will be set to 0.

[-i | Multigrid Iterations]

- This flag specifies how many Multigrid iterations will generate for our surface reconstruction. Note than this flag will **only** have sense if we are using the multigrid mode for the *Biharmonic* solver. By default the number of iterations to execute will be 2.

[-t | Number Threads (default 8)]

- This flags specify how many threads are specified by *OpenMP*. By default, the number of threads will be set to 8.

[-l | Quadtree Subdivision Levels (default 8)]

- This flag specifies the number of subdivisions that will perform the *Quadtree* reconstruction. The generated field of the Quadree will be on par of 2^n where n is the input value. The default number of subdivisions of the *Quadree* will be 8.

[-g | Quadtree Full Grid Division (0 or 1)]

- This flag forces the *Quadree* to be fully subdivided with unitary texels. By default it will be disbled (zero).

[-p | print logs (0 or 1)]

- This flag enables/disables the print traces than provides information of the execution process. By default it will be enabled (one).

[-q | In Depth Study]

- This flag allows to make multiple surface reconstructions at once. By default, given an established aimed *Resolution* the executable will run all the possible combinations that the executable allows. All the results are stored in a *.csv* table as well as into a *.txt* file. Moreover, all the generated images for every combination are stored in a compressed folder.
- If we specify any of the previously mentioned flags he will ONLY test with that specified value and will not perform the other ones. For example, if we introduce the flag *-x Conjugate* he will perform all the possible combinations but only with the Conjugate Gradient as a solver. This flag doesn't require any input parameter.
- This flag is also conditioned by the ***-c*** flag, which will specify if it has to perform a depth study only for the *Biharmonic* or the *Quadtree* methods or it will perform *Both* in-depth studies, each with its own generated files and compressed folders.

[-y | Color Transfer Max Value]

- This flag allows to set the *max value* of the Scalar Field color transfer function that will generate the output reconstructed image. By default, the *Biharmonic* approach will have a value of 16 while the *Quadtree* will have an of 1 (for the Quadtree we recommend low values since higher values could generate images with pale colors where we cannot appreciate the reconstruction with clarity).