UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Surface Reconstruction From 3D Point Clouds

Gonçalo Nuno Vinhas Leitão

Tese para obtenção do Grau de Doutor em
Engenharia Informática
(3º cycle studies)

Orientador: Prof. Doutor Abel João Padrão Gomes

Covilhã, dezembro de 2020

Estas provas de doutoramento (3º ciclo de estudos) no ramo de Engenharia Informática realizaram-se no dia 6 de novembro de 2020. O júri destas provas foi constituído pelo Doutor Abel João Padrão Gomes, professor associado da Universidade da Beira Interior, Doutor António Fernando Vasconcelos Cunha Castro Coelho (arguente), professor associado da Faculdade de Engenharia da Universidade do Porto, Doutor Frutuoso Gomes Mendes da Silva, professor auxiliar da Universidade da Beira Interior, Doutor Mauro Jorge Guerreiro Figueiredo (arguente), professor adjunto do Instituto Superior de Engenharia da Universidade do Algarve, Doutor Daniel Simões Lopes, professor auxiliar do Instituto Superior Técnico da Universidade de Lisboa. O Doutor Mário Marques Freire, Vice-Reitor da Universidade da Beira Interior, presidiu a estas provas de doutoramento por delegação do Reitor da Universidade da Beira Interior.

To my wife and my sons

Acknowledgements

I would like to express my utmost gratitude to Professor Abel Gomes, my advisor, for his guidance, support, encouragement, and patience. This thesis would not have been possible without his experience, knowledge, and invaluable help.

Thanks to my family for having accompanied and encouraged me to pursue this purpose, throughout this journey.

My special and affectionate thank you to my wife and my sons for their patience and constant presence. For them, my love and gratitude.

Resumo

A triangulação de uma nuvem de pontos de um objeto 3D é um problema complexo, uma vez que depende da complexidade da forma desse objeto, assim como da densidade dos pontos extraídos desse objeto através de um *scanner* 3D particular.

Na literatura, existem essencialmente duas abordagens na reconstrução de superfícies a partir de nuvens de pontos: interpolação e aproximação. Em geral, as abordagens de interpolação estão associadas aos métodos simpliciais, ou seja, a métodos que geram diretamente uma malha de triângulos a partir de uma nuvem de pontos. Por outro lado, as abordagens de aproximação estão habitualmente associadas à geração de uma função implícita global —que representa uma superfície implícita— a partir de funções locais de forma, para em seguida gerar uma triangulação da dita superfície implícita.

Os métodos simpliciais dividem-se em duas famílias: triangulação de Delaunay e triangulação baseada em crescimento progressivo da triangulação (i.e., *mesh growing*). Tendo em conta que o primeiro dos métodos apresentados nesta dissertação se enquadra na categoria de métodos de crescimento progressivo, foquemos a nossa atenção por ora nestes métodos. Um dos maiores problemas destes métodos é que, em geral, se baseiam no estabelecimento de limites de ângulos diédricos (i.e., *dihedral angle bounds*) entre triângulos adjacentes, para assim tomar a decisão sobre qual triângulo acrescentar à frente de expansão da malha. Tipicamente, também se usam limites para os ângulos internos de cada triângulo. No decorrer desta dissertação veremos como é que este problema foi resolvido.

O segundo algoritmo introduzido nesta dissertação também é um método simplicial, mas não se enquadra em nenhuma das duas famílias acima referidas, o que nos faz pensar que estaremos na presença de uma nova família: triangulação baseada em atlas de vizinhanças sobrepostas (i.e., atlas of charts) ou estrelas de triângulos (i.e., *triangle star*). Este algoritmo gera um atlas da superfície que é constituído por estrelas sobrepostas de triângulos, ou seja, produz-se a cobertura total da superfície, resolvendo assim um dos problemas comuns desta família de métodos de triangulação direta que é o do surgimento de furos ou de triangulação incompleta da superfície.

O terceiro algoritmo refere-se a um método implícito, mas, ao invés de grande

parte dos métodos implícitos, utiliza uma abordagem de interpolação. Ou seja, as funções locais de forma interpolam os pontos da nuvem. É, talvez, um dos poucos métodos implícitos que podemos encontrar na literatura que interpola todos os pontos da nuvem. Desta forma resolve-se um dos maiores problemas dos métodos implícitos que é o do arredondamento de forma resultante do *blending* das funções locais que geram a função global, em particular ao longo dos vincos da superfície (i.e., *sharp features*).

O que é comum aos três métodos é a abordagem de interpolação, quer em métodos simpliciais quer em métodos implícitos, ou seja a linearização da superfície sujeita a reconstrução. Como se verá, a linearização da vizinhança de cada ponto permite-nos resolver vários problemas colocados aos algoritmos de reconstrução de superfícies, nomeadamente: sub-amostragem de pontos (*point sub-sampling*), amostragem não uniforme (*non-uniform sampling*), bem como formas vincadas (*sharp features*).

Palavras-chave

Computação Gráfica, Geometria Computacional, Malha, Nuvem de Pontos, Reconstrução de Superfícies, Triangulação

Resumo alargado

Este resumo alargado, escrito em Língua Portuguesa, tem como objetivo descrever de forma breve o trabalho de investigação subjacente a esta tese de doutoramento. Começando por fazer o enquadramento da tese, o resumo prossegue com a descrição do problema que se pretende resolver, com a respetiva hipótese de investigação (*thesis statement*), bem como com uma possível solução para o problema previamente formulado. O resumo alargado termina com a descrição da estrutura da tese e com a apresentação das principais conclusões, sem deixar de apresentar também algumas linhas de investigação futura.

Enquadramento da Tese

A reconstrução de superfícies é um problema que visa a construção de uma superfície, quer ela seja paramétrica, simplicial ou implícita, a partir de um conjunto de pontos desorganizado (ou nuvem de pontos) que resultou da amostragem da superfície de um objeto 3D. Obviamente, a sua visualização requerá sempre a sua triangulação de qualquer superfície, com a exceção das superfícies simpliciais que já são trianguladas.

Este problema de reconstrução de uma superfície a partir de uma nuvem de pontos 3D, previamente adquirida por meio de um scanner 3D, é um importante tópico de investigação nos domínios da computação gráfica e da geometria computacional, com aplicações variadas em realidade virtual, animação computacional, engenharia reversa, visão computacional, biomedicina, bioinformática molecular e, ainda, em património cultural.

O problema geral da reconstrução de superfícies é desafiante devido aos problemas subsidiários que se colocam quando a amostragem (densidade) de pontos não é uniforme, quando existem zonas com falta de pontos, quando existem vincos e ápices, quando existe ruído ou quando existe desalinhamento dos pontos. Apesar dos muitos esforços de investigação que têm sido realizados nas últimas décadas para ultrapassar esses problemas, não se pode dizer que existe um algoritmo que resolva todos eles simultaneamente.

Assim, o foco deste trabalho é o de conceber e implementar algoritmos de reconstrução de superfícies a partir de nuvens de pontos que não sejam sensíveis aos problemas referidos acima. No entanto, contrariamente ao que acontece com

outros algoritmos, pretende-se que a reconstrução da superfície seja realizada evitando recorrer a critérios que imponham demasiadas limitações geométricas ou de outra natureza, como é o caso dos limites de ângulos diédricos entre triângulos adjacentes ou de ângulos internos de triângulos.

Descrição do Problema

Os algoritmos para reconstrução de superfícies podem agrupar-se em três categorias de métodos: métodos de interpolação, métodos de aproximação e métodos híbridos [GVJ+09].

Todos estes métodos garantem genericamente a correta reconstrução da superfície desde que a nuvem de pontos seja suficientemente densa. Se a nuvem de pontos não for suficientemente densa, a reconstrução da superfície será muito provavelmente incorreta. Ou seja, a qualidade da amostragem da nuvem de pontos determina a qualidade ou a correção da superfície que será reconstruída. Os problemas que poderão surgir na reconstrução da superfície são os seguintes: deriva da triangulação (*mesh drifting*), reconstrução incompleta da superfície (*mesh holes*), arredondamento e chanfradura de vincos e ápices (*sharp features*).

O foco desta tese reside na resolução destes problemas de reconstrução de superfíces através de três métodos de interpolação. Os dois primeiros são métodos simpliciais, pois geram diretamente uma malha de triângulos a partir dos pontos da nuvem de entrada, ou seja, os vértices dos triângulos gerados pelo processo de reconstrução são pontos da nuvem de entrada. O terceiro método pertence à categoria de métodos implícitos, os quais estão normalmente associados a abordagens de aproximação, e não de interpolação, como é o caso. Por outras palavras, o terceiro método produz uma superfície implícita que interpola todos os pontos da nuvem de entrada.

Hipótese de Investigação

Os trabalhos de investigação realizados tiveram como objetivo encontrar novos algoritmos de reconstrução de superfícies que superassem os problemas acima mencionados. Neste sentido, os novos algoritmos, entretanto propostos nesta dissertação, deveriam ser menos sensíveis à densidade da nuvem de pontos de entrada e às variações de forma da superfície.

Por conseguinte, a investigação ao longo deste trabalho foi no sentido de explorar uma base de conceitos (ou critérios) dos domínios da geometria e da topologia que permitisse a correta reconstrução de superfícies, independentemente de se usar métodos simpliciais ou implícitos. Assim, a hipótese de investigação (*thesis statement*) que deu origem à presente tese, pode ler-se como se segue:

> *É possível reconstruir corretamente superfícies a partir de nuvens de pontos, independentemente das variações de amostragem e de forma da superfície, utilizando para isso uma estrutura de critérios assente nos conceitos de planaridade, regularidade, vizinhança e* manifoldness.

De modo mais concreto, pode dizer-se que os dois primeiros algoritmos de reconstrução da superfícies propostos nesta dissertação tiram partido pleno destes três conceitos, ao passo que o terceiro algoritmo só não tira partido do critério de regularidade, porque a triangulação da superfície definida implicitamente é feita utilizando o algoritmo de triangulação baseada em cubos marchantes (*marching cubes algorithm*) [LC87].

Percurso da Investigação

Com o objetivo de demonstrar positivamente a hipótese acima mencionada, os trabalhos de investigação conducentes à presente dissertação de doutoramento tiveram como principais etapas as que a seguir se descrevem.

- *Algoritmo baseado no crescimento progressivo da triangulação (mesh growing)*: O ponto de partida para o desenvolvimento deste algoritmo, designado por algoritmo PCR (acrónimo de Proximidade, Coplanaridade e Regularidade), resultou do objetivo de eliminar os limites impostos aos ângulos diédricos (i.e., *angle bounds*) entre triângulos adjacentes, tão comuns nos algoritmos de crescimento progressivo, como se pode constatar nos algoritmos propostos por Wongwaen [WTS12], Wang *et al.* [WZZW13] e Xumin *et al.* [XLC14]. No algoritmo PCR, não existem limites impostos aos ângulos diédricos, sendo que a triangulação da superfície cresce dando prioridade às regiões mais coplanares ou de mais baixa curvatura. Este algoritmo baseia-se em três critérios geométricos: proximidade, coplanaridade e regularidade. Como se verá, a proximidade torna o algoritmo menos sensível à falta de uniformidade na densidade dos pontos da nuvem de entrada. A coplanaridade evita o fenómeno de deriva da triangulação,

bem como torna desnecessária a imposição de limites aos ângulos diédricos entre triângulos adjacentes. A inovadora função de regularidade permite produzir malhas com triângulos que tendem a ser regulares, não havendo necessidade de uma etapa de regularização após a construção da malha.

- *Algoritmo baseado em estrelas compatíveis de triângulos*: A ideia principal deste algoritmo, designado por CTC (*Compatible Triangle Charts*), é a de construção de um atlas de estrelas compatíveis na triangulação, sendo que cada estrela de triângulos representa uma vizinhança (ou carta) que está centrada em cada ponto da nuvem. Portanto, o algoritmo é paralelizável com uma *thread* por estrela, o que significa também que este algoritmo não é de crescimento progressivo. As estrelas de triângulos têm suporte no conceito topológico de estrela de pontos da teoria de grafos. Tal conceito é utilizado no algoritmo PCR, mas só para construir uma estrela de triângulos que é a semente inicial da triangulação. Ao invés, no algoritmo CTC constrói-se uma estrela para cada ponto da nuvem de pontos, verificando-se depois a compatibilidade de estrelas sobrepostas em termos de triângulos. Por conseguinte, o objetivo passou por encontrar critérios que permitissem selecionar os triângulos corretos. O critério principal utilizado na seleção de triângulos foi o da *manifoldness*. Isto é, a adjunção de um triângulo à malha em construção não pode criar qualquer situação de quebra de *manifoldness* na referida malha. Note-se que a construção de cada estrela terá de obedecer aos critérios de planaridade, regularidade e também de *manifoldness*. Através da planaridade evita-se fenómenos de deriva da triangulação (ou seja, atalhos de triângulos), permitindo também lidar com características de forma vincada tais como vincos e ápices. A *manifoldness* de cada estrela garante a sua consistência topológica. A regularidade permite construir estrelas e, por consequência, malhas com triângulos que são tendencialmente regulares.

- *Algoritmo baseado na superfície implícita linear*: A ideia de gerar uma superfície implícita através de uma função global que é o resultado da mistura de funções de forma locais não é uma ideia nova. Um dos algoritmos mais populares que utiliza esta ideia para criar uma superfície implícita a partir de um nuvem de pontos é o algoritmo baseado em MPU (*Multilevel Partition-of-Unit*), o qual foi introduzido por Ohtake *et al.* [OBA+03]. No algoritmo baseado em MPU, cada função de ponderação está centrada no centro do respetivo subdomínio (célula terminal da árvore de octantes, também designada por *octree*) com suporte esférico de raio que se so-

brepõe aos limites do seu subdomínio. Deste modo, o centro da função de ponderação leva em consideração o centro do subdomínio, mas não tem em conta a distribuição dos pontos de amostragem no interior do subdomínio, os quais definem a função de aproximação local. No entanto, o centro de um subdomínio pode não ser representativo do conjunto de pontos situados no interior desse subdomínio. No algoritmo aqui proposto, e que se designa por algoritmo baseado em LIS (*Linear Implicit Surface*), cada ponto de amostragem é o centro de um subdomínio definido pela sua estrela de pontos, a qual determina uma função linear local dada pelo seu plano tangente. Isto significa que o algoritmo baseado em LIS codifica um método de interpolação. Por outras palavras, a LIS interpola todos os pontos de amostragem da nuvem de entrada. É, tanto quanto sabemos, o primeiro algoritmo da literatura que produz uma superfície implícita que interpola todos os pontos de amostragem sem ter a necessidade de utilizar cálculo matricial que, como se sabe, é bastante oneroso em termos computacionais. Além disso, o algoritmo LIS evita os problemas inerentes à deriva da triangulação e ao arredondamento e à chanfradura de vincos e ápices.

Principais Contribuições

Considerando a hipótese de investigação (*thesis statement*) supracitada, o trabalho de investigação que conduziu à presente tese, tem como principal contribuição a seguinte:

- É de facto possível reconstruir corretamente superfícies a partir de nuvens de pontos, independentemente das variações de amostragem e de forma da superfície, utilizando para isso uma estrutura de critérios assente nos conceitos de planaridade, regularidade, vizinhança e *manifoldness*.

Em resultado deste trabalho de investigação, existem outras contribuições que devem ser destacadas e que a seguir se indicam:

- A inovadora *função de regularidade* introduzida no Capítulo 3 permite-nos produzir malhas que são formadas tendencialmente por triângulos regulares. Assim, ao invés da esmagadora maioria dos algoritmos de reconstrução de superfícies, não é necessário qualquer passo de pós-processamento para regularizar a malha.

- O conceito de planaridade aplicado transversalmente aos três algoritmos desenvolvidos, ainda que concretizado por diferentes propriedades geométricas, permite evitar o problema da deriva de forma, bem como descartar limites impostos aos ângulos diédricos entre triângulos adjacentes, tão comuns em algoritmos de triangulação direta.

- A compatibilidade de estrelas de triângulos *manifold* deu origem a um novo algoritmo de interpolação que não se enquadra nem na família de algoritmos baseados na triangulação de Delaunay, nem na família de algoritmos baseados no crescimento progressivo da triangulação (*mesh growing*). No algoritmo CTC, a triangulação é feita por compatibilização dos triângulos das estrelas sobrepostas, e não por avanço da frente de expansão. Estamos, pois, na presença de uma nova família de algoritmos de interpolação, aqui designados por algoritmos baseados em atlas de mapas (*atlas of charts*), em que cada mapa é um estrela de triângulos (*triangle star*).

- O algoritmo baseado na LIS, apesar da sua natureza e da sua aplicabilidade à reconstrução de superfícies implícitas, é um método de interpolação, porque as funções lineares locais interpolam os pontos de amostragem. Aliás, pelo que julgamos conhecer da literatura, é um dos poucos algoritmos de reconstrução de superfícies implícitas que interpola os pontos da nuvem de entrada sem usar cálculo matricial. Note-se que este algoritmo também utiliza estrelas de pontos para definir a zona de influência das funções locais de forma.

Organização da Tese

Tendo como objetivo a conceção, o desenvolvimento e a implementação de novos algoritmos para a reconstrução de superfícies a partir de nuvens de pontos, esta tese de doutoramento encontra-se estruturada da seguinte forma:

- *Capítulo 1*: No primeiro capítulo é feita uma descrição do trabalho de investigação realizado e que conduziu à elaboração da presente tese de doutoramento. Além do enquadramento do tema abordado, são indicados os motivos que estiveram na origem deste trabalho.

- *Capítulo 2*: Este capítulo revisita a literatura no que respeita aos métodos para de reconstrução de superfícies a partir de nuvens de pontos, indicando os seus princípios, bem como as suas virtudes e as suas limitações.

- *Capítulo 3*: Neste capítulo é descrito um novo algoritmo de interpolação para a reconstrução de superfícies (PCR), e que se enquadra na família dos algoritmos baseados em crescimento progressivo da triangulação. Este algoritmo tem por base três propriedades geométricas (i.e., proximidade, coplanaridade e regularidade) que permitem superar os problemas inerentes a este tipo de algoritmos.

- *Capítulo 4*: Um outro novo algoritmo para reconstrução de superfícies é descrito neste capítulo. Este algoritmo (CTC) é também um método de interpolação, mas pertence, como se verá, a uma nova família de algoritmos. Essencialmente, o algoritmo produz um atlas de estrelas de triângulos que cobre a superfície completamente.

- *Capítulo 5*: Neste capítulo é proposto um novo algoritmo (LIS) de interpolação que se enquadra na família dos métodos implícitos de reconstrução de superfícies. Portanto, este algoritmo produz uma superfície implícita a partir de uma função global que resulta da combinação ponderada das funções locais que representam os planos tangentes nos pontos de amostragem da nuvem de entrada.

- *Capítulo 6*: Este último capítulo apresenta as principais conclusões dos trabalhos de investigação que conduziram à escrita da presente dissertação, apresentando também algumas questões pertinentes para trabalho futuro.

Considerações Finais

Ao longo dos trabalhos de investigação conducentes à presente dissertação foram desenvolvidos os seguintes algoritmos:

- PCR: algoritmo de crescimento progressivo da triangulação (i.e., *mesh growing*).

- CTC: algoritmo de compatibilização de estrelas de triângulos (i.e., *atlas of charts*).

- LIS: algoritmo de reconstrução de superfícies implícitas lineares.

O que estes algoritmos têm em comum é que todos eles se enquadram na categoria dos métodos de interpolação. Os dois primeiros são algoritmos de triangulação direta, ou seja, os pontos da nuvem de entrada tornam-se vértices da

triangulação da superfície reconstruída, ao passo que o terceiro é uma algoritmo de triangulação indireta, pois requer que se determine uma função paramétrica ou implícita antes de se passar à sua triangulação.

O primeiro é um algoritmo simplicial que se baseia no crescimento progressivo da triangulação (*mesh growing*). O segundo é também um algoritmo simplicial, mas baseia-se na compatibilização de estrelas de triângulos. O terceiro é um algoritmo implícito pois determina uma função implícita que representa a superfície antes de levar a cabo a sua triangulação. Esta função implícita resulta da mistura de funções lineares locais que representam planos tangentes nos pontos de amostragem.

Como se sabe, os algoritmos de reconstrução de superfícies são sensíveis às variações da densidade e da forma da nuvem de pontos resultantes da amostragem gerada por qualquer scanner 3D. O que os algoritmos propostos nesta dissertação têm também em comum é que todos eles foram desenhados para ultrapassar os problemas de variação de densidade e de forma. Para isso, todos eles tiram partido dos mesmos conceitos geométrico-topológicos: proximidade, planaridade, regularidade e, ainda, *manifoldness*. A exceção é que o terceiro algoritmo não utiliza o conceito de regularidade porque a discretização da superfície implícita é feita através do algoritmo de cubos marchantes (*marching cubes*).

Por exemplo, o critério da coplanaridade utilizado no algoritmo PCR encontra paralelo nas funções locais lineares do algoritmo LIS, em que cada função local representa um plano tangente num dos pontos de amostragem. Desta forma evita-se o fenómeno de deriva da triangulação (i.e., atalhos indesejáveis entre regiões que passam próximas umas das outras), bem como o arredondamento de formas vincadas da superfície (i.e., *sharp features*).

Convém lembrar que o problema de reconstrução de superfícies a partir de nuvens de pontos é um problema mal-colocado, visto existirem um número infindável de superfícies que aproximam uma nuvem de pontos. Além do mais, uma nuvem de pontos não define por si só uma superfície. Portanto, este problema inverso requer que se defina um conjunto de pressupostos e de restrições que determinam como a superfície será reconstruída, ou seja, diferentes pressupostos e restrições geram diferentes superfícies para a mesma nuvem de pontos. A escolha que recaiu nos métodos de interpolação constitui, pois, uma tentativa de reduzir o número de superfícies que podem ser geradas a partir de uma nuvem de pontos.

Abstract

The triangulation of a point cloud of a 3D object is a complex problem, since it depends on the complexity of the shape of such object, as well as on the density of points generated by a specific scanner.

In the literature, there are essentially two approaches to the reconstruction of surfaces from point clouds: interpolation and approximation. In general, interpolation approaches are associated with simplicial methods; that is, methods that directly generate a triangle mesh from a point cloud. On the other hand, approximation approaches generate a global implicit function — that represents an implicit surface — from local shape functions, then generating a triangulation of such implicit surface.

The simplicial methods are divided into two families: Delaunay and mesh growing. Bearing in mind that the first of the methods presented in this dissertation falls under the category of mesh growing methods, let us focus our attention for now on these methods. One of the biggest problems with these methods is that, in general, they are based on the establishment of dihedral angle bounds between adjacent triangles, as needed to make the decision on which triangle to add to the expansion mesh front. Typically, other bounds are also used for the internal angles of each triangle. In the course of this dissertation, we will see how this problem was solved.

The second algorithm introduced in this dissertation is also a simplicial method but does not fit into any of the two families mentioned above, which makes us think that we are in the presence of a new family: triangulation based on the atlas of charts or triangle stars. This algorithm generates an atlas of the surface that consists of overlapping stars of triangles, that is, one produces a total surface coverage, thus solving one of the common problems of this family of direct triangulation methods, which is the appearance of holes or incomplete triangulation of the surface.

The third algorithm refers to an implicit method, but, unlike other implicit methods, it uses an interpolation approach. That is, the local shape functions interpolate the points of the cloud. It is, perhaps, one of a few implicit methods that we can find in the literature that interpolates all points of the cloud. Therefore, one of the biggest problems of the implicit methods is solved, which has to do with the smoothing of the surface sharp features resulting from the

blending of the local functions into the global function.

What is common to the three methods is the interpolation approach, either in simple or implicit methods, that is, the linearization of the surface subject to reconstruction. As will be seen, the linearization of the neighborhood of each point allows us to solve several problems posed to the surface reconstruction algorithms, namely: point sub-sampling, non-uniform sampling, as well as sharp features.

Keywords

Computer Graphics, Computational Geometry, Mesh, Point Cloud, Surface Reconstruction, Triangulation

Contents

List of Figures

List of Tables

List of Acronyms

BPA	Ball Pivoting Algorithm
CTC	Compatible Triangle Chart
LIS	Linear Implicit Surface
LS	Least Squares
MLS	Moving Least Squares
MPU	Multiple Partition Unity
PCR	Proximity, Coplanarity and Regulaty
RBF	Radius Basis Function
UBI	Universidade da Beira Interior

Chapter 1

Introduction

This chapter introduces the surface reconstruction problem in the scope of computer graphics and computational geometry. Also, it briefly describes the motivation behind the research work, and how the research was carried out throughout the Ph.D. programme. At the end of this chapter, we present the organization or structure of this dissertation.

1.1 Motivation

The reconstruction of surfaces is a problem that aims to build a surface, whether it is parametric, simplicial, or implicit, from an unorganized set of points (or point cloud) that resulted from the sampling of the surface of a 3D object. Nevertheless, its visualization will always require triangulation of such a surface, except for simplicial surfaces because they are already triangle meshes.

The problem of reconstructing a surface from a 3D point cloud is an important research topic in computer graphics and computational geometry, with diverse applications in virtual reality, computational animation, reverse engineering, computer vision, biomedicine, molecular bioinformatics, and cultural heritage.

Surface reconstruction is a challenging problem because it is hard to deal with the non-uniform sampling (density) of points, sharp features like creases and apices, sampling noise, and misalignment of the points. Despite the many research efforts that have been carried out in recent decades to overcome these problems, one cannot say that there is an algorithm that solves all of them simultaneously.

Thus, the focus of this work is on the design and implementation of surface reconstruction algorithms from point clouds, solving at the same time the problems mentioned above. However, contrary to what happens with other algorithms, we intend to carry out the surface reconstruction avoiding the use of criteria that impose too many geometric limitations, such as the bounds on dihedral angles between adjacent triangles or internal angles of triangles.

1.2 Surface Reconstruction: Overview

In the literature, we find several surface reconstruction methods, which can be grouped in three main categories [GVJ+09]:

- Interpolation methods
- Approximation methods
- Hybrid methods

Interpolation methods. These methods aim to build a simplicial surface that interpolates the input point samples. In turn, this category of methods divides into two families. The first family comprises algorithms that build upon Delaunay triangulations (and Voronoi diagrams) of the input point samples. The crust algorithm of Amenta et al. [ABK98] is an emblematic algorithm of this family because it was the first to guarantee a correct surface reconstruction from a sufficiently dense point set. The cocone algorithms (see [ACDL00] or [CSD04]) are also representatives of this family. However, these algorithms have difficulties in dealing with either sparse point sets (i.e., they are rather sensitive to point sampling) or massive point sets (i.e., they are very time-consuming). The second family includes the mesh growing algorithms, which make the mesh grow progressively under some geometric conditions. The ball pivoting algorithm [BMR+99] is a well-known mesh growing algorithm, as well as the one due to Xumin *et al.* [XLC14]. In general, mesh growing algorithms have limited success in generating a mesh that interpolates a given point cloud, largely because they use bounds for the admissible dihedral angle to decide on the next triangle to be attached to the mesh front, as well as for the internal angles of each triangle. Moreover, interpolation methods have difficulties in dealing with non-uniform density or low density of such point clouds.

Approximation methods. They aim at fitting a smooth surface to the input point samples [BTS+17]. These methods also are very time-consuming because they need to solve large systems of equations. Besides, they have difficulties in dealing with sharp features as those of mechanical parts, as rounding effects may occur. Interestingly, they can handle noisy point sets because they mostly follow an approximation approach. Some of these approximation methods define implicit functions at samples, which jointly form a zero level set of the surface to be built up. Several representations of implicit surfaces are used such as, Radial Basis Functions, Moving Least Squares or Multiple Partition Unity, which can be

found in [HDD+92], [CBC+01], [SMG10], [KBSS01], [OBA+03], [DS05], [KBH06], [DMSL11]. For further details see Gomes et al. [GVJ+09]. Other approximation methods use parametric surfaces, defined by a function, that closely approximates a point cloud, [FH05], [PL03].

Hybrid methods. They combine the previous techniques concerning interpolation and approximation methods. For that purpose, we define the implicit functions through the distance to samples, since the Delaunay/Voronoi diagram of such samples has a natural connection with the distance function. Interestingly, one can prove that the original surface can be reconstructed with theoretical guarantees through the union of the stable manifolds (i.e., surface patches) of the index-2 critical points that are close to the surface [Gro93]. In the literature, we find other hybrid algorithms that combine Delaunay/Voronoi diagram and region-growing approaches. The algorithms of Kuo and Yau [KY03] or [KY05] are representative of hybrid algorithms. They later improved their algorithms to reconstruct surfaces with sharp features [KY06].

All these methods usually ensure the correct reconstruction of the surface since the point cloud is dense enough. Otherwise, the reconstruction of the surface is very likely to be incorrect. That is, the sampling density of the point cloud determines the correctness of the reconstructed surface. The problems that may arise in the reconstruction of the surface are as follows: mesh drifting, incomplete reconstruction of the surface (mesh holes), rounding and also trimming of sharp features (i.e., creases and apices).

This dissertation aims to research new algorithms for surface reconstruction that overcome the problems mentioned above. Therefore, the main challenges are to design and implement new surface reconstruction algorithms capable of dealing with non-uniform point density and sharp features. These capabilities would avoid problems like mesh drifting, mesh holes, and trimming/rounding of sharp features, without using angle bounds and, at the same time, generating meshes with triangles that tend to be regular.

1.3 Research Hypothesis

The focus of this research work is on the investigation of new interpolation-based surface reconstruction methods. Their novelty stems from the new geometric machinery used to solve the aforementioned problems. In this sense, the new algorithms should be less sensitive to the point sampling density and the shape

variations of the surface.

Therefore, the research behind this work aims to explore a base of concepts (or criteria) from geometry and topology that would allow the correct reconstruction of surfaces, regardless of whether using simplicial or implicit methods. Thus, the thesis statement (which can be reformulated in terms of a research hypothesis) that gave rise to the present dissertation reads as follows:

> *It is possible to correctly reconstruct surfaces from point clouds, regardless of their variations in point sampling density and shape.*

More specifically, we will show that this statement is true using the following geometric and topological criteria: proximity, planarity, regularity, and manifoldness. The first two surface reconstruction algorithms proposed in this dissertation take full advantage of these four concepts. In turn, the third algorithm does not use the regularity criterion because it uses the marching-cubes triangulation for implicitly-defined surfaces [LC87].

1.4 Research Path

To positively demonstrate the hypothesis above, the course of the research work leading to the present dissertation encompassed the following main steps:

- *PCR, a mesh growing algorithm*: The starting point for developing this algorithm, called PCR algorithm (an acronym for Proximity, Coplanarity, and Regularity), resulted from the objective of eliminating the bounds imposed on the dihedral angles (i.e., angle bounds) between adjacent triangles. Note that these angle bounds are typical in mesh growing algorithms, as those due to Wongwaen [WTS12], and Wang et al. [WZZW13] and Xumin et al. [XLC14]. In the PCR algorithm, there are no bounds imposed on the dihedral angles, and the triangulation of the surface grows, giving priority to the most coplanar or lower curvature regions. This algorithm mostly builds upon three geometric criteria: proximity, coplanarity, and regularity. As will be seen, proximity makes the algorithm less sensitive to the lack of uniformity in the point sampling density. Coplanarity avoids the phenomenon of mesh drifting and makes it unnecessary to impose bounds on the dihedral angles between adjacent triangles. The innovative regularity function allows producing meshes with triangles that tend to be regular, without the need for a regularization step after constructing the

mesh.

- *CTC, a mesh atlas algorithm*: The main idea of this algorithm, called CTC (Compatible Triangle Charts), is to build an atlas of compatible stars in the triangulation, with each triangle star representing a neighborhood (or chart) centered at each cloud point. Therefore, the algorithm is parallelizable using one thread per triangle star. In other words, the CTC algorithm is not a mesh growing algorithm. PCR algorithm takes advantage of the triangle star concept, but only to build a seed triangle star at the beginning of the triangulation. On the contrary, in the CTC algorithm, we use a point star (i.e., a sequence of points neighboring each point) for each point in the point cloud. Then we check the compatibility of overlapping stars in terms of triangles. Therefore, the objective was to find criteria that would allow selecting the right triangles. The main criterion used in the selection of triangles was that of manifoldness. The attachment of any triangle to the mesh under construction cannot break the mesh manifoldness. Note that each star's construction must satisfy the conditions of proximity, planarity, regularity, and manifoldness. Planarity avoids the phenomenon of mesh drifting (i.e., shortcuts), also allowing to deal with sharp features (i.e., creases and apices). Manifoldness guarantees its topological consistency. Regularity makes it possible to build stars and, consequently, meshes with triangles that tend to be regular.

- *LIS, an implicit algorithm*: The idea of generating an implicit surface through a global function resulting from local functions' blending is not new. One of the most popular algorithms using this idea to create an implicit surface from a point cloud is the MPU-based algorithm (Multi-level Partition-of-Unit), introduced by Ohtake et al. [OBA+03]. In the MPU-based algorithm, each weighting function applies to the center of the respective subdomain (leaf cell of the octree) with spherical radius support that overlaps its subdomain's limits. In this way, the center of the weighting function takes into account the center of the subdomain. Still, it does not consider the distribution of sampling points within the subdomain, which defines the local approximation function. Therefore, the center of a subdomain may not represent the set of points located within that subdomain. In the algorithm proposed here, which is called an algorithm based on LIS (Linear Implicit Surface), each sampling point is the center of a sub-domain defined by its point star, which determines a local linear function given by its plane tangent. Thus, the LIS-based

algorithm encodes an interpolation method. In other words, the LIS interpolates all sampling points in the input cloud. As far as we know, LIS is the first algorithm that produces an implicit surface that interpolates all the sampling points without the need to use time-consuming matrix computations. Besides, LIS overcomes the problems of mesh drifting, rounding, and trimming of sharp features (i.e., creases and apices).

1.5 Contributions

Regarding the research hypothesis (thesis statement) above, the research work underlying this dissertation has the following main contribution:

- It is possible to correctly reconstruct surfaces from point clouds, regardless of the point sampling and shape variations, using a setup of criteria based on the following concepts: proximity, planarity, regularity, and manifoldness.

As a result of this research work, we must highlight other contributions as follows:

- The innovative regularity function introduced in Chapter 3 allows us to produce meshes that tend to be formed by regular triangles. Thus, unlike the overwhelming majority of surface reconstruction algorithms, no post-processing step is necessary to regularize the mesh.

- The concept of planarity applied transversely to the three developed algorithms, even though realized by different geometric properties, allows us to avoid the problem of shape drifting and discard bounds imposed on the dihedral angles between adjacent triangles, so common in the direct triangulation algorithms.

- The star compatibility of manifold triangles gave rise to a new interpolation algorithm that does not fit either in Delaunay triangulation algorithms or in mesh growing algorithms. In the CTC algorithm, the triangulation occurs by making the triangles of the overlapping stars compatible and not by advancing the expansion front. Therefore, we are in the presence of a new family of interpolation algorithms, here called atlas-of-charts based algorithms, in which each chart is a triangle star.

- The LIS-based algorithm is an interpolation method because local linear

functions interpolate the sampling points. From what we think we know from the literature, it is the first algorithm that reconstructs an implicit surface that interpolates the input cloud points without using time-consuming matrix computations. Note that this algorithm also takes advantage of point stars to define the zone of influence of local shape functions.

1.6 Thesis Organization

Considering that the dissertation intends to develop and implement new algorithms for the reconstruction of surfaces from point clouds, it is then structured as follows:

- *Chapter 1*: The first chapter briefly describes the research work which led to the preparation of this doctoral thesis. Furthermore, it approaches the research topic of surface reconstruction in the context of computer graphics and geometric computing and the motivation behind this work.

- *Chapter 2*: This chapter revisits the literature regarding methods for reconstructing surfaces from point clouds, indicating its principles, as well as its virtues and limitations.

- *Chapter 3*: This chapter describes a new interpolation algorithm for reconstructing surfaces (PCR), which fits into the family of mesh growing algorithms. This algorithm builds upon three main geometric properties (say, proximity, coplanarity, and regularity) that allow us to overcome the problems inherent to this type of algorithms.

- *Chapter 4*: This chapter describes a new algorithm for reconstructing surfaces, called CTC algorithm. CTC is also an interpolation method, but it belongs to a new family of algorithms. Essentially, this algorithm produces an atlas of compatible triangular stars that completely covers the surface.

- *Chapter 5*: This chapter proposes a new interpolation algorithm that fits the family of implicit surface reconstruction methods (LIS). Therefore, this algorithm produces an implicit surface from a global function that results from the weighted blending of the local functions representing the tangent planes at the input cloud points.

- *Chapter 6*: This last chapter presents the main conclusions of the research work that has led to this dissertation's write-up and puts forward some

questions for future work.

1.7 Summary

Throughout the research work leading to this dissertation, we developed the following algorithms:

- *PCR*: a mesh growing algorithm.
- *CTC*: triangle star matching algorithm (i.e., atlas of charts).
- *LIS*: reconstruction algorithm of implicit linear surfaces.

What these algorithms have in common is that they all fall into the category of interpolation methods. The first two are direct triangulation algorithms; that is, the input cloud points become vertices of the reconstructed surface triangulation. In turn, the third is an indirect triangulation algorithm, as it requires determining a parametric or implicit function before moving on to its triangulation.

The first is a simplicial algorithm that is based on mesh growing. The second is also a simplicial algorithm, but it builds upon the compatibility or matching of triangular stars. The third is an implicit algorithm because it determines a global implicit function representing the surface before carrying out its triangulation. This implicit function results from the blending of local linear functions representing tangent planes at the sampling points.

As known, the surface reconstruction algorithms are sensitive to variations in density and shape of the point cloud resulting from sampling generated by any 3D scanner. Therefore, our three algorithms intend to overcome such problems of point density and shape. For this purpose, our algorithms take advantage of the same geometric and topological concepts: proximity, planarity, regularity, and manifoldness. The exception is the third algorithm, which does not use the concept of regularity because one discretizes the implicit surface through the marching cubes algorithm.

For example, the coplanarity criterion used in the PCR algorithm finds its parallel in the LIS algorithm's linear local functions, with each local function representing a tangent plane at each sampling point. This fact allows us to avoid the phenomenon of shape drifting (i.e., undesirable shortcuts between surface re-

gions that pass close to each other) and prevent the smoothing of sharp features of the surface.

Recall that reconstructing surfaces from point clouds is an ill-posed problem since there is an endless number of surfaces that approximate a point cloud. Furthermore, a point cloud *per se* does not define a surface. Therefore, this inverse problem requires the definition of a set of assumptions and constraints which determine the reconstruction of a specific surface. That is, different assumptions and constraints generate different surfaces for the same point cloud. Hence, it is not strange that our choice for interpolation methods, as they constitute an attempt to reduce the number of surfaces generated from a point cloud.

Chapter 2

Interpolation Methods for Surface Reconstruction: A Survey

2.1 Introduction

The main focus of this research work is on interpolation methods for surface reconstruction from a point cloud. Therefore, we survey here such methods. The interpolation methods for surface reconstruction divide into Voronoi/Delaunay based methods, region-growing methods, and implicit surface-based methods. The first type of algorithm computes the Voronoi diagram and its dual graph, which corresponds to the Delaunay triangulation. The reconstructed surface consists of triangles resulting from the Delaunay triangulation. In the second type of algorithms, the reconstruction process starts with a triangle seed, and then the mesh grows under some geometric criteria. Finally, implicit methods build upon triangulation of cloud point-interpolatiing isosurfaces locally defined by kernel functions.

2.2 Voronoi/Delaunay Based Methods

Surface reconstruction algorithms based on the Delaunay triangulation/Voronoi diagram work well when the sample points are dense enough. If the point cloud is dense enough, there is no need for any hole-filling step to guarantee the reconstruction correctness. However, these algorithms produce holes on the reconstructed surface if the point clouds are not dense enough (i.e., downsampling). Thereby, the reconstructed surface becomes incomplete. In case of downsampling, errors at sharp features may also occur. Another drawback of these algorithms is they are very time-consuming when the number of sampling points increases because the Delaunay triangulation/Voronoi diagram's computation slows down rapidly.

As described below, crust algorithms and co-cone algorithms are well-known algorithms based on the Voronoi diagram and Delaunay triangulation to produce a triangle mesh from a point cloud.

2.2.1 Crust Algorithms

Amenta et al. in [ABK98] proposed the first Voronoi diagram based surface reconstruction algorithm with theoretical guarantees. Such an algorithm was called the crust algorithm, which has worked as a basis for developing other algorithms.

2.2.1.1 The Crust Algorithm

The crust algorithm is described by Amenta et al. in [ABK98], and its theoretical guarantees appear in [AB99]. This algorithm computes the Voronoi diagram of the points belonging to the point cloud. After that, one determines Delaunay triangulation from the Voronoi diagram.

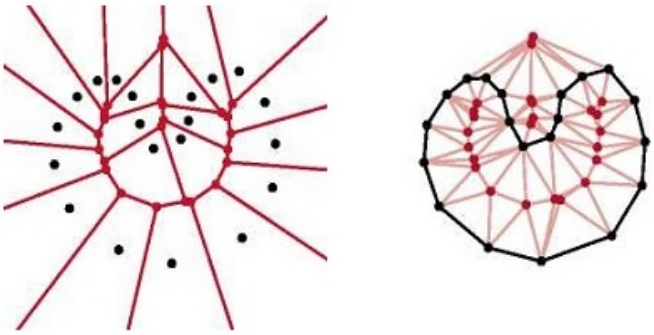


Figure 2.1: 2D crust algorithm: On the left, the Voronoi diagram of a point set S sampled from a curve. The Voronoi vertices V in red approximate the medial axis of the curve; On the right, the Delaunay triangulation of $S \cup V$, with the crust edges in black. (abusively taken from [ABK98])

In 2D Euclidean space, we define the crust as the graph of Delaunay edges that connects all sample points (e.g., the black polygon in Figure 2.1). Note that the crust is a subset of the Delaunay triangulation of the input points. Moreover, the Voronoi vertices (in red) allow us to filter out the unwanted edges from the Delaunay triangulation, a procedure we call *Voronoi filtering*. Voronoi vertices also approximate the medial axis if the points sampled are dense enough.

In 3D Euclidean space, not all Voronoi vertices are near to the medial axis. Some are closed to the surface. As shown in [AB99] when the sample points are dense enough, the solution is to use only two Voronoi vertices, called poles, per sample point. These two poles of a sample point are the farthest vertices of the sample point's cell below and beyond the surface. As shown in Fig. 2.2), those poles well approximate to the surface's medial axis. Doing so, we generalize the crust algorithm to 3D. The Delaunay triangles with circumspheres rid of poles

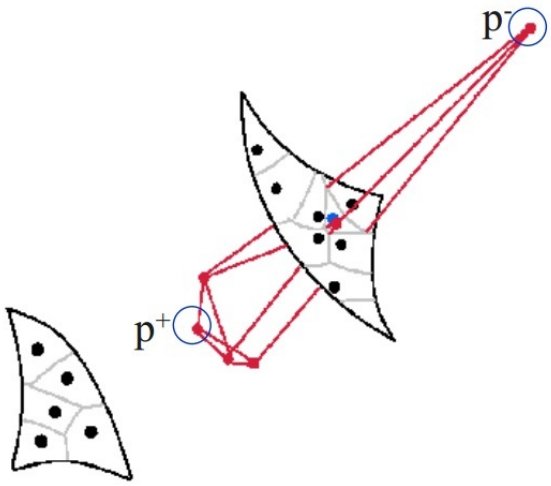


Figure 2.2: Voronoi vertices called poles: the two farthest vertices of the Voronoi cell one on each side of the surface, p^+ and p^-. (abusively taken from [ABK98])

result in a piecewise-linear surface that converges to the original surface. That is, the crust algorithm only keeps those triangles whose three vertices are sample points (*Voronoi filtering*). Poles also allow for further filtering of unwanted triangles using surface normals (*normal filtering*). This second filtering removes any triangle whenever at least one of its vertices satisfies the following condition: the angle between its surface normal and the vector to one of its poles is "too large". The definition of "too large" means "greater than" an input parameter given by the user. This parameter is connected with the sampling density and has to be estimated, backing off when holes appear in the reconstructed surface.

The crust algorithm's theoretical guarantees come with the notion of r-sampling, which we define relative to the medial axis of the sampled surface. In fact, the crust algorithm uses a positive function, called the *Local Feature Size* (LFS). LFS at the point $\mathbf{x}$ defines itself as the distance from $\mathbf{x}$ to its nearest point in the medial axis. Therefore, the LFS function describes the sampling density. A sample is said to be sufficiently dense if the distance from any sample point $\mathbf{x}$ to its nearest neighbor is smaller than the constant r ($0 < r < 1$) times the LFS function. The expression $r \times LFS$ is called sampling condition. The constant r is an input parameter that the user needs to estimate, determining the r-sampling of the point cloud. For a small r, it means the sample is dense enough, and the algorithm will produce a good reconstructed surface. When r is too big, the algorithm may fail.

2.2.1.2 Power Crust Algorithm

The power crust algorithm proposed by Amenta et al. in [ACK01a] also produces a piecewise-linear approximation of the original surface. The algorithm first produces a MAT (Medial Axis Transform) approximation from the point cloud. It then applies an inverse transform to the MAT to generate a piecewise-linear surface that approximates the original surface. Details about the theoretical guarantees of this algorithm are in [ACK01b].

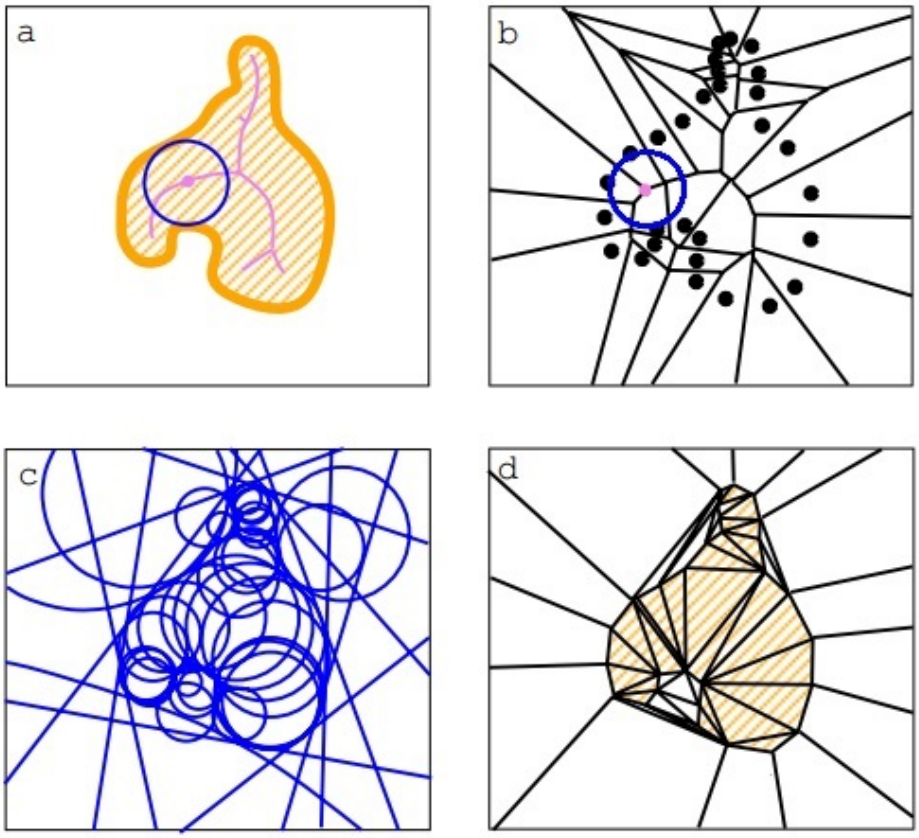


Figure 2.3: 2D example of power crust construction: a) an object with its medial axis. One maximal interior ball is shown; b) the Voronoi diagram of the sample points, with the Voronoi ball surrounding a single pole. In 2D, all Voronoi vertices can be selected as poles, but not in 3D; c) the inner and outer polar balls. Outer polar balls with centers at infinity degenerate to halfspaces on the convex hull; d) the Power diagram cells of the poles, labeled inner and outer. (abusively taken from [ACK01a])

Note that one approximates the MAT by a subset of the Voronoi vertices of the input point cloud, called the poles. As seen above, these poles are near the medial axis. The balls centered at the poles and touching the nearest sample points are the polar balls. These polar balls approximate maximal balls contained either in the interior or else in the surface's exterior. Similarly, the power diagram is the Voronoi diagram of polar balls and divides space into poly-

hedral cells. Labeling power diagram cells inside or outside, the power crust is the boundary (2-dimensional faces) that separates the power diagram cells belonging to inner poles from power diagram cells belonging to outer poles (see Fig. 2.3). The power crust is the piecewise-linear surface that approximates the original object's surface.

Although the power crust algorithm is an interpolation method, not all input sample points are power crust vertices; conversely, not all power crust vertices are input sample points. Besides, power crust faces are not triangles because they result from intersection points between balls in 3D space. Therefore, the power crust algorithm produces a piecewise-linear surface with more points and faces than other comparable triangulated surfaces. Under the theoretical sampling assumptions, power crust is a robust algorithm, always bounding a solid. No hole-filling step is required, and sharp features are correctly polygonized. However, the expensive computing costs of the MAT constitute a drawback of the power crust algorithm. Furthermore, it fails to reconstruct surfaces when the input point cloud owns noise because it might estimate a wrong medial axis. Another drawback is the need to change user input parameters due to the input data's sample spacing variability.

2.2.1.3 The Eigen Crust Algorithm

The eigen crust algorithm is due to Kolluri et al. [KSO04]. This algorithm computes the Delaunay tetrahedralization of the input point cloud, and then it labels each tetrahedron as inside or outside the surface. The result is a piecewise-linear surface called eigen crust. This eigen crust surface is composed of all triangles where each inside tetrahedron meets an outside tetrahedron. This procedure guarantees that the output surface bounds a volume. Also, this procedure ensures that the eigen crust surface is watertight and closed.

The eigen crust algorithm's main novelty is the introduction of spectral partitioning techniques into surface reconstruction. We usually find spectral partitioning techniques in image segmentation, circuit layout, document clustering, and sparse matrix arithmetic on parallel computers. Here, and following Hall [Hal70], Fiedler [Fie73], and Pothen et al. [PSL90], spectral methods were used for partitioning graphs connecting tetrahedra.

Based on a spectral partitioner, the algorithm creates a graph representing the tetrahedralization and subdivides it into two subgraphs, an inside subgraph and an outside subgraph. The advantage has to do with the spectral partitioner hav-

ing a global view of the point set. It effectively identifies the triangles that are most likely to lie at the interface between the object's interior and exterior. This way for labeling the tetrahedral is more robust than in previous algorithms. Besides, spectral surface reconstruction is robust against noise or outliers. However, that makes the eigen crust algorithm slower than competitors. Like its predecessors, this algorithm approximates the original surface well if the cloud points are sampled densely enough from a closed surface.

2.2.2 Co-cone Algorithms

This family of algorithms arose with the basic co-cone algorithm proposed by Amenta et al. in [ACDL00]. The co-cone algorithm simplifies and improves the crust algorithm. After that, we can find in the literature several enhancements to the basic co-cone algorithm.

2.2.2.1 The Basic Co-cone Algorithm

The basic co-cone algorithm [ACDL00] computes the restricted Voronoi diagram of the input point cloud, from which one determines the dual Delaunay triangulation. The Delaunay triangulation contains the piecewise-linear surface one intends to output. Designing this algorithm assumed the sampled surface is a smooth manifold without boundary and was adopted the definition of sampling density from [AB99]. This algorithm provides topological guarantees and geometric accuracy guarantees when the input points are appropriate samples from a smooth surface.

In each restricted Voronoi cell V_p, the farthest Voronoi vertex p^+ is called the positive pole of the sample point p, and the vector pp^+ is the pole vector for p. Pole vectors approximate the normals of the surface S at sample points. The co-cone of point p is the set C_p that is defined as the complement of a double cone centered at p clipped within a Voronoi cell V_p. This double cone has p as the apex, the pole vector pp^+ as the axis, and an opening angle of $3\pi/8$ with the axis (see Fig. 2.4). The resulting piecewise-linear surface requires computing all its triangles from the Delaunay triangulation, whose dual Voronoi edges intersect the co-cones.

The co-cone algorithm simplified the crust algorithm because it only requires one Voronoi diagram computation instead of two such computations in the crust algorithm. Besides, the co-cone algorithm makes unnecessary the normal trimming step (or normal filtering).

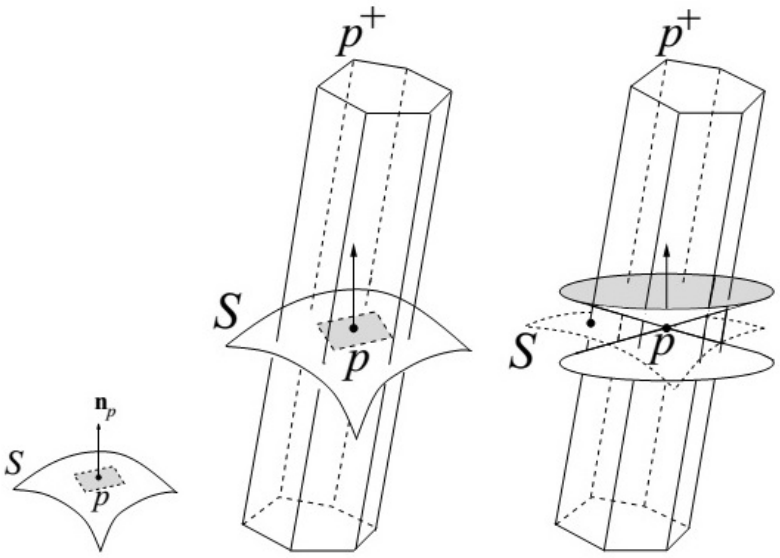


Figure 2.4: A Voronoi cell V_p is elongated along the normal n_p. The pole vector pp^+ approximates n_p. The co-cone C_p is the region in V_p between the two cones at p. (abusively taken from [DG03])

2.2.2.2 The Super Co-cone Algorithm

For unorganized point sets, surface reconstruction methods based on the Voronoi diagram (and its dual) are too slow and cannot handle large data sets; for example, data sets with about a million points. The super co-cone algorithm was proposed by Dey et al. [DGH01] and extends the applicability of the basic co-cone algorithm [ACDL00] to large data sets by using the divide-and-conquer approach. Moreover, the theoretical guarantees assume, globally, the sample points lie with sufficient density on a smooth surface, in line with the r-sampling assumptions presented in [ACDL00] and [AB99].

The super co-cone algorithm builds upon an octree subdivision to accelerate neighbor points' computation and point clustering. This way, it processes small subsets from the entire input point set separately, allowing the computation of local Voronoi diagrams for fewer sample points. Each node divides further if it holds more sample points than the predefined threshold. This procedure ends with the lowest leaf node containing a subset of sample points, which is smaller than the predefined value.

After the octree subdivision step, one computes the local Voronoi diagram computation for each leaf node, considering its lowest adjacent leaf nodes. The local reconstruction is similar to the co-cone algorithm steps described above. However, the super co-cone algorithm gathers a fraction of points belonging to adjacent leaf nodes, but there is no guarantee that a leaf node includes all

their restricted Voronoi neighbors. As a consequence, their co-cones may not be computed correctly. Another consequence is that the correct co-cone triangles may not be computed across the node boundaries, and the reconstructed patches are not stitched seamlessly together. The super co-cone algorithm can handle large sample sets, with the advantage of reconstructing the 3D surface without computing the entire Voronoi diagram that consumes too much time and memory and can be parallelizable.

2.2.2.3 The Tight Co-cone Algorithm

The co-cone algorithm works well in case of the input point set samples the surface densely. However, when the input set undersamples the surface, holes are left on the reconstructed surface, producing an incomplete reconstructed surface. Moreover, the co-cone algorithm computes many undesirable triangles near undersampling regions. The tight co-cone algorithm was proposed by Dey and Goswami in [DG03] and is a modified implementation of the co-cone algorithm that produces a surface that is free of holes.

This version combines several algorithms performing in several phases. First, the co-cone algorithm computes the set of co-cone candidate triangles. Next, it comes the time of marking all input points into well-sampled and poorly-sampled regions applying the detection of undersampling presented in [DG01]. The removal of those triangles with vertices falling into under poorly-sampled regions takes place. In the end, applying the umbrella filter found in [DZ02], holes are detected and filled up.

The umbrella definition of a point resembles a topological disk, and all triangles in the umbrella have that point as the common vertex. For a well-sampled surface, every point in 3D reconstruction owns its umbrella, contrary to when holes show up. This topological structure resembling an umbrella is essential for detecting the surface boundaries, i.e., holes in undersampled regions.

This process to fill up the holes introduces no additional points, and the piecewise-linear surface is generated by interpolating input sample points. Note that the tight co-cone algorithm relies on the principle of locality of undersampling. However, if this assumption is not satisfied, the algorithm faces trouble in computing the surface. Therefore, the output surface may not be close to the original or even produce holes. This algorithm copes with some noise for noisy samples, but it cannot output any surface if the noise is beyond its tolerance limit.

2.2.2.4 The Robust Co-cone Algorithm

In [DG06], Dey and Goswami proposed the robust co-cone algorithm to reconstruct surfaces from noisy point sets. Recalling some of the principles of the power crust algorithm in [ACK01b] and [ACK01a], the union of polar balls approximates the solid bounded by the sampled surface. However, this property does not hold in the presence of noise because of the small and big Delaunay balls' appearance.

The robust co-cone algorithm applies a filtering process to separate big Delaunay balls from small ones. The algorithm will use big Delaunay balls to approximate the solid surface bounded by a point cloud. As shown, some of the big Delaunay balls remain relatively big under some reasonable noise conditions and can play the role of the polar ball. So, we need to separate outer and inner big Delaunay balls, keep the points that reside on the outer (or inner), and then delete the rest of the points. Next, the surface reconstruction performs from the retained points.

Thus, the algorithm computes a surface interpolating the collected points using the restricted Delaunay triangulation of the filtered point set concerning the boundary of the union of the outer (or inner) Delaunay balls. However, for this second phase, the algorithm does not compute the restricted Delaunay triangulation. Instead, the tight co-cone is used to reconstruct the surface from the filtered point set.

A drawback is this algorithm requires two Delaunay triangulation computations, one for the filtering phase and another for the surface reconstruction phase. The theoretical guarantees of the robust co-cone algorithm come under the notion of reasonable noise conditions. So, some assumptions have to be made in the algorithm, such as sampling density or proximity of the points. Therefore, predefined or threshold values for some parameters are assumed.

2.2.2.5 Localized Co-cone Algorithm

The localized co-cone algorithm [DDW11] is a modification of the super co-cone algorithm previously introduced by Dey e al. [DGH01]. As seen above, the super co-cone algorithm is an octree-based version of the co-cone algorithm, such that the restricted Delaunay tetrahedralization only performs upon small clusters of the entire point set. However, the super co-cone algorithm does not guarantee topological correctness and geometric accuracy as the original co-cone algo-

rithm. Therefore, the localized co-cone algorithm results from reformulating the super co-cone algorithm to recover those guarantees.

Like its predecessor, the localized co-cone algorithm also uses an octree for spatial partition. The input point set is divided into small manageable subsets, performing the co-cone algorithm on each subset so that all the pieces match up to one consistent whole. Processing small portions of the surface means determining patches with boundaries, but the co-cone algorithm is not prepared to provide guarantees for surfaces with boundaries. On the other hand, such patches must be consistent across those boundaries to produce a manifold surface without boundary. Besides that, the pole vector used to estimate the normal vector at each sample point may not be reliable if we only consider a portion of the samples.

To face such difficulties and ensure that each node has the correct set of co-cone triangles, one includes those triangles that have vertices in more than one node into the set of the co-cone triangles of each such node. This way, all the properties of the traditional co-cone algorithm are maintained while processing the nodes individually. Note that the pole vector is not used to estimate the normal vector at the sample point p. Instead, one selects the normal to a particular Delaunay triangle bounded by p that is guaranteed to lie close to the surface's tangent plane at p. As demonstrated in [DDW11], any triangle with a vertex at sample point p that have a circumradius bounded by the sampling radius (the distance from sample point p to its nearest neighbor) will suffice for that purpose. Different from its predecessor, the localized co-cone algorithm does not use a fixed size of buffer around each node to create an overlap between adjacent nodes. Such an option has a drawback because the buffer's width might not be sufficiently large concerning the local feature size. The localized co-cone algorithm adaptively builds a buffer around each node, and Dey et al. demonstrated that it is possible to ensure that it is sufficiently broad. The buffer's size depends on the maximum local density requirement, which depends on the sampling condition.

When the sample points lie with sufficient density on a smooth surface, the localized co-cone algorithm produces piecewise-linear surfaces with topological and geometric guarantees. To accommodate input point sets that do not meet the algorithm's theoretical sampling requirements, one uses heuristics to prevent catastrophic failure. When these heuristics are employed, there is no theoretical guarantee on the output. However, the algorithm maintains the theoretical guarantees for point sets whose local sampling density is bounded

by a constant. Specifically, the maximum local density requirement demands for any sample point x a ball with a radius equal to a constant multiplied by the local feature size containing no more than c sample points; by default, $c = 50$ points. Besides, this algorithm has limitations that are inherent to its roots as a co-cone algorithm. Problems may occur if the local curvature does not match a sufficient local sampling density, such as holes in the mesh. Similar problems occur in noisy point sets.

2.2.2.6 Singular Co-cone Algorithm

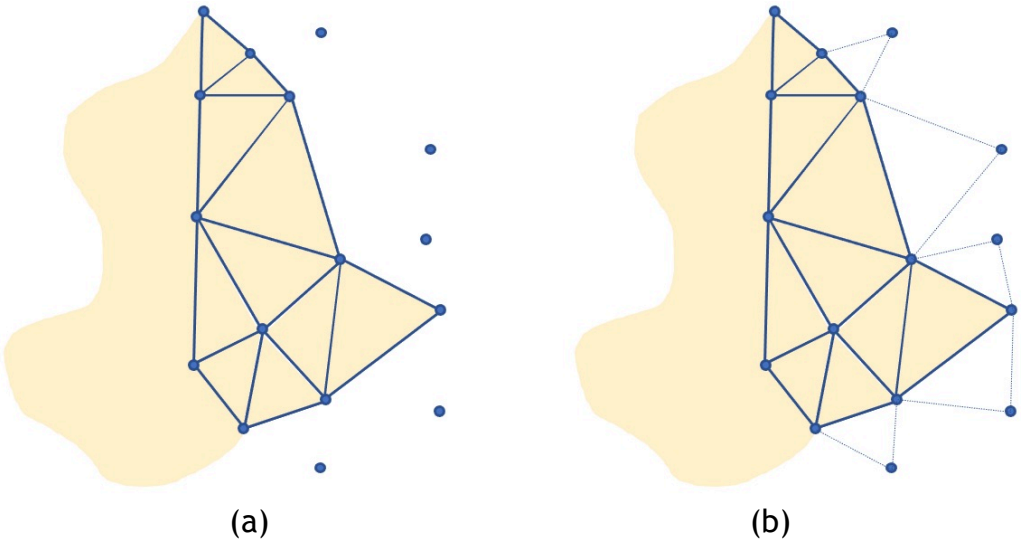
This algorithm is the latest co-cone family algorithm proposed by Dey e al. [DW13]. The singular co-cone algorithm handles singular features, as sharp feature curves or corners, and reconstructs the surface using the weight co-cone method [DGQ$^+$12]. The weight co-cone is a variation of the co-cone algorithm [ACDL00] that uses a weighted Delaunay triangulation.

The singular co-cone algorithm takes advantage of a novel feature recovery method as its first step. For that, a provable Voronoi-based method for feature point detection [DGGZ02] was extended, which can isolate points near the feature curves from a possibly noisy sample of a singular surface with no orientation. Next, curve direction is estimated applying principal component analysis (PCA), and these points are filtered to rebuild feature curves using NNCrust algorithm [DK99]. For the second step, surface reconstruction uses the weight co-cone algorithm described in [DGQ$^+$12] designed to reconstruct surfaces in the presence of feature curves. The singular co-cone algorithm is more robust than other co-cone algorithms in dealing with noise due to the local Voronoi diagram computations for each point and the Voronoi diagram's weighted version. However, these modifications increase the computational cost, but with better accuracy.

The major drawback of this algorithm is the use and setting of multiple parameters. For most models, setting such parameters can be fixed in a short time. However, in a poorly sampled model or a model that contains noise, outliers, and small features close to each other, it can be challenging to find the right set of parameters that aid in capturing all the features. Another drawback has to do with the weights used for feature detection at the noise scale. A too-small weight may not detect the feature point; on the other hand, a too-large weight may compromise the feature detection step's locality and increase the computation cost.

2.3 Mesh-Growing Methods

The leading idea of mesh-growing algorithms is to grow the surface mesh starting from a seed triangle, so that new triangles are attached to the mesh front while there are cloud points to triangulate (see Fig. 2.5)



(a) (b)

Figure 2.5: Illustrating the mesh growing procedure: (a) partial triangulation of a point cloud; (b) attaching new triangles (dashed triangles) to mesh front.

2.3.1 Ball Pivoting Algorithm

One of the most known mesh-growing algorithms is the ball pivoting algorithm (BPA), introduced by Bernardini et al. [BMR+99]. This algorithm augments the data points with approximate surface normals to decide surface orientation.

BPA algorithm uses a sphere of a predefined radius that inscribes each edge of the mesh front or boundary, searching for the next cloud point on the sphere, with the restriction that no other point is inside the sphere. So, the three points on the sphere form a putative new triangle, but, for that, the dot product of its normal vector with the surface normal must be positive; otherwise, one rejects the triangle. Therefore, this algorithm is very dependent on the choice of the ball radius; consequently, it is not robust for noisy point clouds unless the point cloud is dense enough, nor when the original object presents ridges or sharp features. Moreover, this algorithm outputs the surface with holes when the sample points are low density. For dealing with point clouds of non-uniform density, the BPA allows for multiple ball radii.

The BPA takes much time to complete and is memory intensive. Hence, its multi-threading implementation proposed by Digne [Dig14], using multiple ball

radii. Compared with Delaunay/Voronoi-based algorithms, the BPA algorithm is time-efficient because it does not need to compute the Delaunay triangulation or Voronoi diagram to produce the mesh surface.

2.3.2 Advanced-Front Algorithm

Cohen-Steiner et al. in [CSD04] present a surface reconstruction algorithm based on the 3D Delaunay triangulation from unorganized point sets that uses a greedy approach, called the advanced-front algorithm. The leading idea is to extract a 2D triangle mesh (or piecewise-linear surface) from the 3D Delaunay triangulation. The advanced-front algorithm sequentially selects the triangles that are attached to the mesh front one by one. At each advancing step and following a specific criterion, the algorithm selects the most plausible triangle, striking on the generation of an orientable manifold triangulated surface.

The first step of this algorithm is the 3D Delaunay triangulation of the point set. After that, the Delaunay triangle with the smallest radius is the seed triangle. This triangle is the initial triangulated surface, and its three boundary edges are the initial advancing front. The radius of a triangle is the smallest sphere's radius passing through that triangle's vertices and enclosing no other sample point.

This algorithm maintains a priority queue of candidate (valid) triangles incident to the current surface's boundary edges. That priority is the plausibility grade. The algorithm pops from the queue the most plausible candidate triangle and adds it to the surface, while the priority queue is not empty.

New candidate triangles are pushed to the priority queue whenever new boundary edges appear on the advancing front. As this algorithm strikes on the generation of a manifold surface, some candidate triangles cannot be selected due to topological constraints. Those topological constraints correspond to four configurations, and a triangle is valid whether one of those configurations applies.

Besides, valid triangles for an edge on the front mesh are subjected to geometric criteria. Such criteria impose angle bounds, namely the dihedral angle between adjacent triangles, the angle between the triangles' normals, and the internal angles of the triangle. The user can specify these bounds. Thus, the candidate triangle of an edge on the front mesh is valid if it owns the smallest radius and satisfies the angle bounds criteria. Consequently, the most plausible candidate triangle among those in the priority queue has the smallest radius.

Results indicate this algorithm should be competitive. However, it was not proved the topological correctness of the output under reasonable assumptions on the sampling. Although this algorithm correctly reconstructs the surface for most cases, angle bounds are a drawback. Moreover, the algorithm may not work well for regions where the sampling is too sparse or non-uniform concerning curvature.

2.3.3 Scale-Space Surface Meshing Algorithm

Digne et al. [DMSL11] developed a scale-space strategy for orienting and meshing a raw input point set. Later, a parallel implementation of the scale-space meshing algorithm was proposed by Digne [Dig15]. The scale-space algorithm applies the mean curvature motion (MCM) to a set of points. The MCM moves all points toward the shape's concavity at a rate equal to the surface mean curvature. The use of the mean curvature motion, forward and backward, is a direct 3D extension of the scale-space paradigm in image processing introduced by Witkin in [Wit83]. A scale-space represents a shape at different geometric scales, i.e., at different degrees of smoothness.

The principle behind the scale-space meshing method is to use a standard surface mesh reconstruction algorithm to interpolate the point set once we get the shape smoothed. Then, we can find the mesh for the original scale from the smoothed scale mesh. The scale-space surface meshing algorithm builds upon the scale-space framework for reconstructing a mesh from an oriented input point set. This algorithm first smooths the point set, producing a singularity-free shape. Then the ball pivoting algorithm is used to build a mesh from the smoothed point set. The final step consists of back projecting the mesh built on the smoothed positions onto the original point set. The result of this process is a surface mesh whose vertices are almost all raw input points.

As shown in [DMSL11], this algorithm approximates the mean curvature motion through the iterative process of projecting each point of the data set onto the local regression plane of its radial neighborhood. This iterative projection process allows for the computation of robust geometric information, which can be backtracked to the initial scale. The projection algorithm makes no use of the normal; the orientation choice is irrelevant for the mean curvature motion, but having that information is useful for performing the ball pivoting algorithm.

The use of a scale-space overcomes a few limitations. If data has to be smoothed before building a mesh, then the interpolating property may be lost. Moreover,

although the ball pivoting algorithm works well for noiseless data or data with details at a scale coherent with the ball's chosen radius, it fails when data contains small details or noise. So, scale-space allows for a better interpolation of the original raw points and deals better with noise and small details.

The scale-space surface meshing algorithm has ideal conditions. A few are inherent to the ball pivoting algorithm. The surface should be reasonably regularly sampled and dense enough, and it should not contain overly concavities. A drawback of this algorithm is that it depends on parameters whose values we need to set each other. We need two parameters for scale-space: the radius of the projection filter and the number of iterations. The ball pivoting algorithm needs one parameter, the radius of the pivoting ball, which is equal to half of the projection filter radius. Consequently, when using not appropriate parameter settings or lacking sample points, we observe that concave regions, sharp features, and sparse regions may not be reconstructed correctly.

2.3.4 Other Mesh Growing Algorithms

The mesh growing algorithm proposed by Li et al. [LHW09] uses a priority-driven function for the mesh growing so that low curvature regions triangulate before high curvature regions. In this regard, this algorithm resembles our PCR Cocktail, but unlike ours, it uses angle bounds for internal angles of each triangle and the dihedral angle between adjacent triangles. The sum of three quantities gives its priority-driven function. The first is the value of the cosine of the largest internal angle of the new triangle; the second quantity is the cosine of the dihedral angle between the border triangle and the new triangle; the third corresponds to the distance between the border edge and the evaluation point. Among all candidate points beyond the growing mesh front, one selects one with the highest priority to generate the next triangle. In addition to angle bounds, this algorithm does not produce regular high-quality triangulations, being necessary to use some triangle optimization procedure to locally re-triangulate some triangles after surface mesh reconstruction. In some circumstances, in particular, when a region is very noisy, the algorithm becomes noise-sensitive.

Angelo et al. [ASG11] introduced a mesh growing algorithm based on Gabriel 2-Simplex criterion (G2S), which states that a triangle is a G2S if the smallest ball that circumscribes it is empty [AGJ00]; this is the 2D counterpart in 3D space for the Delaunay' empty circle criterion in 2D space. However, given a point cloud, and unlike Delaunay triangulations, the G2S triangulation is not unique, as it depends on the seed triangle. Besides, there is no guarantee that G2S

triangles all belong to the mesh surface, so some additional work to distinguish wrong from right triangles; this involves a non-manifoldness repair because some triangles are transverse to each other. In a way, the G2S algorithm is similar to the ball pivoting method, but it has the advantage of adapting the ball radius to the spacing of neighbor points. Like other algorithms, one of the main downsides is that some of its parameters are empirically set, not entirely automated. Even worse, it is a fact the G2S may fail for very non-uniform point clouds.

Wongwaen et al. [WTS12] proposed another mesh-growing algorithm, which subdivides the bounding box that encloses the point cloud into small equally-sized cubes. Then, triangulation occurs inside each cube, resulting in a set of triangle sub-meshes requiring some interstitial triangulation. However, this interstitial triangulation may go wrong if the sub-meshes of adjacent cubes are not supposed to be connected, which happens when the surface gets close to itself without touching, resulting in an undesirable mesh drifting effect. This fact also means that this algorithm cannot cope with sharp features and noise. Besides, it is sensitive to point density, and finding the right size of the cubes is challenging, not to say unfeasible. Also, similar to other mesh growing algorithms, it imposes bounds to the internal angles of each triangle and the dihedral angle between adjacent triangles.

The mesh-growing algorithm due to Wang et al. [WZZW13] tries to avoid drastic transitions in the mesh growing, so imposing angle bounds to the dihedral angle between adjacent triangles, as usual in other algorithms; it also avoids too narrow triangles by imposing angles bounds to internal angles of each triangle. As usual, this triangulation method assumes that the input data enjoy a uniform sampling rate, concerns a closed surface, and possesses noise-free sampling points.

Xumin et al. [XLC14] proposed another mesh-growing surface reconstruction algorithm that depends on angle bounds. Interestingly, it takes advantage of an octree space subdivision so that a local triangulation takes place in each octant containing points of the original cloud, with a maximum of 48 points. The problems resulting from the surface mesh reconstruction are related to the difficulty in finding adequate reconstruction criteria, hence using empirical parameters. Consequently, sharp or high curvature regions, and oscillatory or noisy regions, create difficulties in triangulating the point cloud. The algorithm also produces triangulation holes when the point sampling is not uniform.

2.4 RBF Implicit Surfaces Interpolation

The use of radial basis functions (RBFs) in computer graphics was introduced by Nielson [Nie93] to build interpolants for 3D data where the interpolation centers do not form a regular grid. After that, Savchenko et al. [SPOK95], Carr et al. [CFB97], and Turk and O'Brien [TO99] developed the first surface reconstruction algorithms based on RBF interpolants.

Radial basis functions are used for interpolating a multivariate scattered data. Let focus on the problem of interpolating a multivariate function f from a set of sample values $f(x_i)$ ($i = 1, \cdots, N$) on a scattered point set x_i ($i = 1, \cdots, N$) in $\mathbb{R}^3$. In order to reconstruct f, this function is locally approximated by a real-valued function (called radial basis function) ϕ at each center x_i. That real-valued function depends on the Euclidean distance from each center x_i and is radially symmetric. The characteristic property of a radial basis function is its monotone decreasing (or increasing) with the distance from its center.

Thin-plate functions are a class of radial basis functions and are a particular case of polyharmonic functions. The success of RBF interpolation has to do with an appropriate choice of the radial basis functions. Turk and O'Brien [TO99] used the triharmonic thin-plate RBF $\phi(\|x - x_i\|) = \|x - x_i\|^3$. The uniharmonic thin-plate RBF $\phi(\|x - x_i\|) = \|x - x_i\|$ was used by Carr et al. [CFB97]. After the choice for the radial basis function ϕ, function f can be approximated by the interpolant $F(x)$ that results from the sum of N radial basis functions, each of which is associated with a distinct center x_i, and weighted by an adequate coefficient w_i as follows:

$$F(x) = \sum_{i=1}^{N} w_i \phi(\|x - x_i\|) \quad \textbf{(2.1)}$$

However, it is necessary to add a low-degree polynomial $P(x)$ to the interpolant $F(x)$ to guarantee the positive-definiteness, which is one condition to ensure the uniqueness of the solution. Thus, the RBF interpolant results as follows:

$$F(x) = P(x) + \sum_{i=1}^{N} w_i \phi(\|x - x_i\|) \quad \textbf{(2.2)}$$

As Duchon [Duc77] showed, the smoothness of the interpolant can be guaranteed by a linear polynomial P.

Surface reconstruction through RBF interpolation requires to approximate a real-valued function $f(x)$ by the RBF interpolant $F(x)$ given the scalar values f_i at the distinct points x_i ($i = 1, \cdots, N$). The computation of the interpolant $F(x)$ requires determining the weights w_i and the vector of unknown coefficients of the polynomial $P(x)$ in terms of its basis. The weights w_i are such that $F(x)$ satisfies the interpolation conditions, $F(x_i) = f_i$, $i = 1, \cdots, N$. Besides, we need to impose orthogonality conditions on the weights w_i because this system of equations has more parameters than data, such as

$$\sum_{i=1}^{N} w_i P(x_i) = 0 \quad (2.3)$$

All the above conditions determine a system of equations which may be written in matrix form to determine the coefficients of P, the weights w_i, and after all, the interpolant $F(x)$. The time consumption for solving this equation system is the main drawback of this interpolation process, mainly when the number of data points goes above a few thousands.

The RBF interpolation allows the reconstruction of complex and smooth implicit surfaces with arbitrary topological shapes from a set of scattered data points. RBFs impose no restrictions on data points, namely, concerning those where data points lie in, the type of regular grid, and space decomposition of the points.

2.4.1 Fast RBF Interpolation

As referred before, the standard RBF interpolation has a significant drawback. It only copes with small scattered data sets, having up to two thousand points approximately. Carr et al. [CBC+01], and [CBM+03] overcame this problem by introducing a kind of fast RBFs to reconstruct surfaces from large point sets. We can speed up the RBF interpolation thanks to two mechanisms: the fast multipole method (FMM) and RBF center reduction.

The FMM method allows the computation of the matrix-vector product using lower order operations than $O(n^2)$ operations performed on the direct product. The center reduction technique used in the fast RBF interpolation makes possible

the surface reconstruction from large point data sets. The fast RBFs do not use all the input scattered points x_i as RBF centers. Instead, we use a significantly smaller point set as the centers. This technique also allows removing redundant detail or noise.

2.4.2 CS-RBF Interpolation

Standard RBF interpolation and fast RBF interpolation have a global nature that comes from the use of noncompact supported basis functions. That is a drawback of the RBFs because even a small change of one center may affect the entire interpolated surface. Instead of that, the use of radial basis functions with compact support (CS-RBFs) preserve the principle of locality, which means changing the position of a given center xi causes only a local change of the interpolant and the corresponding surface. Thus, the use of RBFs with compact support gives a better control because of each radial function's local influence, which depends on the radial basis function's radius at each center.

A table with a list of radial basis functions with compact support, which can be used to interpolate an implicit surface from scattered point data, can be found in Wendland [Wen95]. Those functions' support has a radius equal to 1, although scaling is allowed any radius. Therefore, a piecewise polynomial interpolant of minimal degree is built from summing up the contributions of these polynomial, positive-definite, and compactly supported RBFs associated with the centers.

2.5 Approximation Methods

We are not carrying out here a comprehensive review of approximation methods for surface reconstruction. Indeed, we will only address the algorithm we used in our tests, the Poisson surface reconstruction algorithm, which comes wrapped in CGAL Library (`http://www.cgal.org`).

2.5.1 Poisson Surface Reconstruction Algorithm

Introduced by Kazhdan et al. [KBH06], the Poisson surface reconstruction algorithm is a two-step process. It consists of computing an implicit function that approximates a solid surface, outputting a surface mesh as an isosurface of that function. Given a set of 3D points with oriented normals gotten as samples from the surface of a 3D solid, the Poisson surface reconstruction algorithm computes an approximate indicator function of the solid, whose gradient best matches the input normals. The output scalar function, represented in an adaptive octree, is

then iso-contoured using an adaptive marching cubes algorithm. The problem of computing the indicator function is to perform a least-square fit to minimize the difference between the gradient of the scalar function f and the vector field V of the input normals defined by the sample points. The divergence operator is applied to transform this problem into a standard Poisson problem; hence, the Poisson equation $\Delta f = \nabla \bullet V$. That means computing the scalar function f whose Laplacian (divergence of the gradient) equals the vector field's divergence V.

CGAL library comes with a variant of the Poisson algorithm. This variant computes a piecewise linear function on a 3D Delaunay triangulation instead of an adaptive octree. The algorithm takes input from a set of 3D-oriented points and builds the 3D Delaunay triangulation from these points. The Delaunay refinement presented in [RY07] is applied to remove all poorly shaped (non-isotropic) tetrahedra. Thus, the number of input points reduces as well as noise in the input data. Then, one computes the scalar indicator function f, represented by a piecewise-linear function over the refined Delaunay triangulation. The Poisson equation is solved at each vertex of the triangulation using a sparse linear solver. In the end, using the CGAL surface mesh generator, an isosurface is extracted with function value set by default as the median value of f at all input sample points.

Ideally, this implementation of the Poisson surface reconstruction method expects a sufficiently dense 3D oriented point set and sampled over a closed smooth surface. The algorithm solves an implicit function, which is an approximate indicator function of an inferred solid. For this reason, the algorithm always extracts a closed surface mesh, hence can fill the small holes where data are missing. Moreover, it is reasonably robust to noise. However, there is a drawback; the algorithm may not recover the sharp features and corners present in the inferred solid because of the smooth effect of approximation local functions on sharp features and corners. That is a common characteristic of implicit methods. Another limitation has to do with the reconstruction of thin features at undersampled regions. Such reconstruction of thin features depends on the local spacing, and it is related to the local feature size (the distance to the medial axis, which captures curvature, thickness, and separation altogether). There is no formal proof of correctness reconstruction by the algorithm under certain density conditions. When density is not enough, the algorithm does not reconstruct the thin features.

2.6 Summary

We have surveyed the most relevant surface reconstruction algorithms, particularly those that take advantage of the interpolation approach. In this way, we intend to highlight the three algorithms' contributions introduced in the next chapters. We have also described the only approximation algorithm we used in benchmarking for dissertation self-containedness sake.

Chapter 3

Surface Reconstruction: PCR Algorithm

3.1 Introduction

Reconstructing a surface from an unorganized point set (or point cloud) acquired by a laser scanner or a similar device is an important research topic in computer graphics, geometric modeling, virtual reality, computer animation, medicine, and reverse engineering, to mention a few. However, surface reconstruction algorithms have difficulties dealing with non-uniform density (including holes), sharp features (e.g., creases and apices), shape drifting, and noise of such unorganized point clouds. The method proposed —called PCR Cocktail— directly triangulates the entire raw point set without normal vectors and allows for an accurate meshing of the scanning holes and sharp features, even when the point sampling is not uniform. Furthermore, contrary to other meshing methods, our method does not generate new points; that is, only raw data points are vertices of the final mesh.

PCR Cocktail is a mesh growing method for surface reconstruction. It generates a triangle mesh from a seed triangle so that every new triangle is attached to a mesh front's triangle in conformity with one or more geometric criteria. More specifically, PCR Cocktail uses three geometric criteria, namely *proximity* (P), *coplanarity* (C), and *regularity* (R). As shown further ahead, the proximity relation is used in two circumstances and makes PCR Cocktail not sensitive to non-uniform point density of the input point cloud. Coplanarity allows for avoiding the shape drifting phenomenon that shows up in other methods and discarding dihedral angle bounds between adjacent triangles, which are typical in mesh growing methods. Note that PCR Cocktail triangulation flows from low curvature regions to high curvature regions. It gives priority to coplanar or low curvature regions in the triangulation of a given point cloud. Finally, an innovative regularity function allows us to produce meshes with triangles that tend to be regular (or equilateral). Thus, there is no need for a regularization step after the mesh construction. In short, combining the three criteria above allows us to solve most surface reconstruction problems identified in the literature and thus control the correctness of the final mesh.

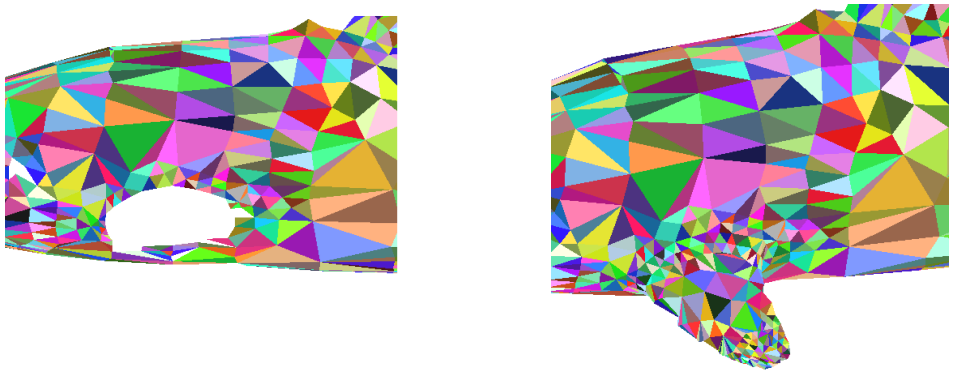
3.2 Background

The input data P consists of a set of points $\mathbf{p} \in P$, which we assume to lie on the surface $S = \partial O$ of an unknown object O. Besides, we assume that S is a manifold; that is, it is locally homeomorphic to $\mathbb{R}^2$. Our goal is to reconstruct a watertight triangle mesh M that linearly approximates to the surface S so that the points of P are vertices of M.

As mentioned above, our surface reconstruction algorithm builds upon three fundamental geometric criteria: *proximity*, *regularity*, and *coplanarity*.

3.2.1 Coplanarity

Coplanarity is the dominant geometric criterion of our mesh growing method. This criterion imposes a local geometric constraint to the manifoldness of the surface S. That is, the neighborhood of each point $\mathbf{p} \in P$ in S must be not only homeomorphic to $\mathbb{R}^2$ (or an open disc) but also isometric to $\mathbb{R}^2$ as much as possible. Thus, we prioritize the triangles around each point $\mathbf{p} \in P$ in S according to their coplanarity. The decision to attach either a candidate triangle or another to a mesh front edge depends on which one is the most coplanar with the adjacent triangle of the growing mesh.



(a) (b)

Figure 3.1: Reconstruction of flat and low-curvature regions come first.

In practice, in each step of the algorithm, among all candidate triangles to be attached to the edges of the mesh growing front, we select the one that is the most coplanar with its nearest front triangle. Hence, our argument of not using angular bounds. In other words, the triangulation of S takes place firstly on planar regions and lastly in high curvature regions (e.g., sharp features). Equivalently, the mesh growing works by decreasing the dihedral angle from π to 0, that is, the most coplanar triangles first, the fewer coplanar triangles

afterward. Fig. 3.1 shows this mesh growing process, where the mesh front in the dolphin's body only expands to the left pectoral fin when the body gets fully meshed.

3.2.2 Regularity

The *regularity* of the mesh has to do with the generation of approximately regular triangles, i.e., approximately equilateral triangles. PCR Cocktail tends to generate regular triangulations without using any post-processing step. We say that the triangle T_1 is more regular than the triangle T_2 if the sum of the cosines of the three internal angles of T_1 is greater than the one of T_2. This sum is here called *regularity function*, and expresses itself as follows:

$$r_T(\alpha, \beta, \gamma) = \cos(\alpha) + \cos(\beta) + \cos(\gamma) \quad (3.1)$$

where α, β, and γ stand for the internal angles of a given triangle T. As it proved further, the *regularity function* has a maximum when the three angles are identical and equal to 60°. That maximum is $r_T(60°, 60°, 60°) = 1.5$, and it is the maximum regularity of the given triangle.

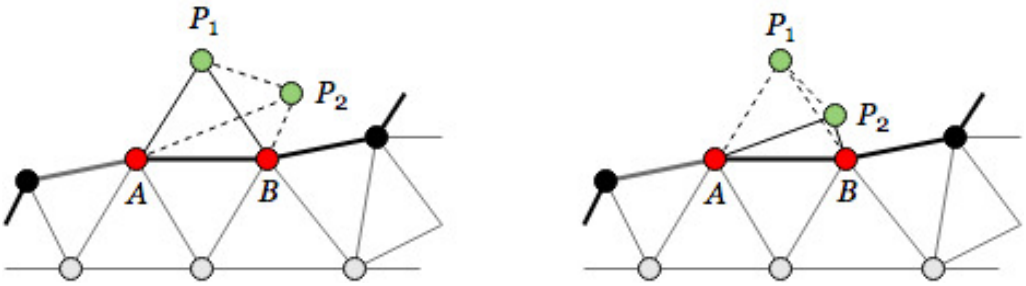


Figure 3.2: Four triangulations for two sets of four points when $P_1 \notin S_2$ and $P_2 \notin S_1$: (a) for this set of four points, the correct triangulation choose the point P_1 to form a triangle with $\overline{\mathbf{AB}}$; (b) for this set of four points, the correct triangulation choose the point P_2 to form a triangle with $\overline{\mathbf{AB}}$.

Therefore, given two triangulations of four points, it is feasible to decide which one is the most regular triangulation, as illustrated in Fig. 3.2. For example, in Fig. 3.2(a), we have two triangulations of the same set of four points, A, B, P_1, and P_2; the first triangulation $\mathcal{T}_1$ consists of two triangles, $T_1 = (A, P_2, B)$ and $T_2 = (A, P_2, P_1)$, whereas the triangles of the second triangulation $\mathcal{T}_2$ are $T_3 = (B, P_1, A)$ and $T_4 = (B, P_1, P_2)$. We see $\mathcal{T}_2$ is more regular than $\mathcal{T}_1$, as can

be determined through the following *relative regularity function*:

$$r = \max\{\min\{r_T(T_1), r_T(T_2)\}, \{\min\{r_T(T_3), r_T(T_4)\}\} \quad (3.2)$$

Consequently, we choose P_1 as the point that together with $\overline{\mathbf{AB}}$ will form the next triangle of the triangulation. Also, applying the above regularity function to the triangulations in Fig. 3.2(b) allows us to know that the triangulation consisting of the triangles (A, P_2, P_1) and (A, P_2, B) is more regular than the triangulation comprising the triangles (B, P_1, A) and (B, P_1, P_2); consequently, (A, P_2, B) will be next triangle of the triangulation. Note that we do not impose angular restrictions over the amplitude of the mesh triangles' internal angles.

3.2.3 Maximum regularity

Let consider the function which is called *regularity function* and is given by the following expression,

$$r_T(\alpha, \beta, \gamma) = \cos(\alpha) + \cos(\beta) + \cos(\gamma). \quad (3.3)$$

The variables, α, β, and γ are the internal angles of a given triangle T and they are under the next constraint,

$$\alpha + \beta + \gamma = 180^\circ \text{ with } 0^\circ < \alpha, \beta, \gamma < 180^\circ. \quad (3.4)$$

Applying the above constraint, the *regularity function* is equivalent to the bivariate expression,

$$r_T(\alpha, \beta) = \cos(\alpha) + \cos(\beta) + \cos(180^\circ - \alpha - \beta). \quad (3.5)$$

Let's find the maximum of this function. For that, we begin to take the gradient (∇) of the *regularity function*,

$$\nabla r_T(\alpha, \beta) = (\sin(180^\circ - \alpha - \beta) - \sin(\alpha), \sin(180^\circ - \alpha - \beta) - \sin(\beta))$$

Thereafter, we have to find the critical points of the *regularity function*, because the extreme values of this function will be find in such set of points. So, we need to solve the following equation,

$$\nabla r_T(\alpha, \beta) = (0, 0) \quad (3.6)$$

equivalent to,

$$\sin(180^\circ - \alpha - \beta) - \sin(\alpha) = 0 \wedge \sin(180^\circ - \alpha - \beta) - \sin(\beta) = 0$$

or yet,

$$\sin(180^\circ - \alpha - \beta) = \sin(\alpha) \wedge \sin(180^\circ - \alpha - \beta) = \sin(\beta). \quad (3.7)$$

Consequently,

$$\sin(\alpha) = \sin(\beta) \quad (3.8)$$

where two solutions are possible,

$$\alpha = \beta \vee \alpha = 180^\circ - \beta. \quad (3.9)$$

However, according to the previous constraint at Eq. 3.4, only the first solution have to be considered. So, by substitution in Eq. 3.7, just in one of the two conditions is needed, it comes,

$$\sin(\alpha) = \sin(180^\circ - 2\alpha) \Leftrightarrow \alpha = 180^\circ - 2\alpha \Leftrightarrow \alpha = 60^\circ \quad (3.10)$$

and also,

$$\beta = 60^\circ. \quad (3.11)$$

By the above demonstration, *regularity function* has only one critical point at $(\alpha, \beta) = (60^\circ, 60^\circ)$. Next, we will apply the Hessian matrix to that critical point to determine whether at this point is a maximum. The Hessian matrix is as follows:

$$H(\alpha, \beta) =$$

$$\begin{bmatrix} -\cos(\alpha) - \cos(180^\circ - \alpha - \beta) & -\cos(180^\circ - \alpha - \beta) \\ -\cos(180^\circ - \alpha - \beta) & -\cos(\beta) - \cos(180^\circ - \alpha - \beta) \end{bmatrix}$$

then,

$$H(60^\circ, 60^\circ) = \begin{bmatrix} -1 & -1/2 \\ -1/2 & -1 \end{bmatrix} \quad (3.12)$$

where the principal minors in ascending order are, respectively, $d_1 = -1$ and $d_2 = 3/4$. Hence, the Hessian matrix at $(60°, 60°)$ is negative definite, thus, *regularity function* attains a maximum, moreover, that maximum is $r_T(60°, 60°) = 1.5$.

As demonstrated, the maximum regularity of a triangle occurs when all internal angles are 60°of amplitude, which means it is an equilateral triangle.

3.2.4 Proximity

Proximity is the geometric concept underlying the computation of the nearest neighbors of a cloud point and the computation of the closest point to a given edge. In the first case, to decrease the complexity and searching time to find the point neighborhoods, we use the octree subdivision of the bounding box containing the point cloud. The objective is to reduce, for each point, the search domain of the closest points instead of considering all sample points, but such search space's reduction cannot be too much. The subdivision performs until the leaf nodes contain fewer points than a pre-established threshold. This threshold considers that each edge of a triangle determines four perpendicular half-spaces holding such an edge. Therefore, around each vertex of the triangle, we have eight perpendicular half-spaces. So, by selecting the closest point in each of those eight half-spaces per vertex, we end up having 24 points nearby to a triangle. Thus, 24 points make the threshold below which we stop the octree subdivision of each populated octant. This way, the search for the closest points of a given point is limited to its leaf node's points and its adjacent nodes, which all together form the point neighborhood. Therefore, PCR neither pre-establish a maximum radius nor pre-establish a minimum number of points for the closest points surrounding a given point. So, PCR Cocktail is less sensitive to the non-uniform point density of the input point cloud.

In the second case, when attaching a new triangle to the mesh front, we first need to find the appropriate closest point to a given front edge. Let be point $\mathbf{A}$ and point $\mathbf{B}$ the endpoints of a given edge. The nearest point $\mathbf{P}$ of both $\mathbf{A}$ and $\mathbf{B}$ among their neighbor points $N(\mathbf{A})$ and $N(\mathbf{B})$, respectively, is the one given by the minimum distance d that combines the distances from $\mathbf{A}$ and $\mathbf{B}$ to $\mathbf{P}$, that is,

$$d = \min_{\mathbf{x} \in N(\mathbf{A}) \cup N(\mathbf{B})} (||\mathbf{A} - \mathbf{x}|| + ||\mathbf{B} - \mathbf{x}||) \quad (3.13)$$

3.3 PCR Cocktail Triangulation

The PCR Cocktail algorithm essentially consists of the following steps:

1. Find the seed triangle and build the smooth and manifold *star* of triangles.
2. Find candidate triangles on new edges of mesh front;
3. Select the new triangle among all candidate triangles to be attached to mesh front;
4. Repeat the last two steps while there are points to triangulate.

3.3.1 Finding the seed triangle and the *star*

The seed (or initial) triangle choice is crucial in reconstructing the surface; otherwise, one may undermine the correctness or quality of the resulting triangulation. So, this first step of the PCR algorithm aims to find a smooth curvature region to start the surface reconstruction. Given a cloud point and taking it as the center point, we build an initial triangle. Afterward, we construct a smooth manifold *star* approximately isomorphic to a disc by attaching new triangles to the initial triangle and around that fixed center point, as illustrated in Fig. 3.4. Note that no neighbor point can project to any star triangle orthogonally. Otherwise, we cannot ensure the smoothness and manifoldness of the *star*.

The procedure to find the point where to start the surface reconstruction is the following:

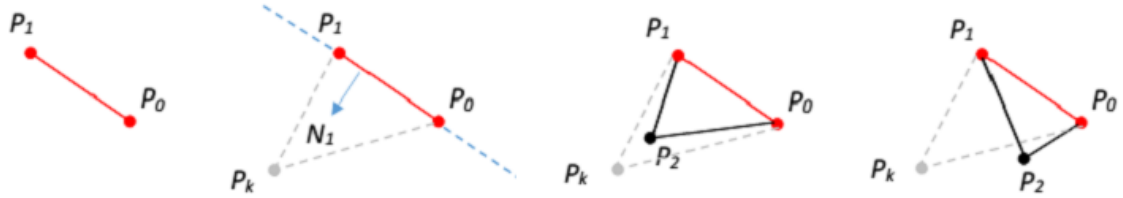
- Find the initial triangle;
- Build the smooth and manifold *star*;

The previous procedure is applied to each cloud point until one point succeeds. Let us detail the two sub-steps outlined above.

3.3.1.1 Finding the initial triangle

Given a point P_0, consider the point P_1 as the closest point to the first one. Both points make the initial edge, Fig. 3.3(a). Then, find the point P_k that belongs to the P_0 neighbor points, making the most regular triangle with the initial edge $\overline{P_0P_1}$. Next, determine the normal vector $\vec{N_1}$ with the initial edge and coplanar with the triangle $\Delta(P_0P_1P_k)$ and also toward the point P_k. The

vector $\vec{N_1}$ determines the half-space where the initial triangle lies. Besides this, the first candidate to the initial triangle is $\Delta(P_0P_1P_k)$. Moreover, if there is no closer point to the initial edge among the P_0 neighbor points, then the initial triangle is found, as shown in Fig. 3.3(b). However, suppose there is another point P_2 closer to the initial edge. In that case, we look at three cases, and we have to decide among the triangles $\Delta(P_0P_1P_k)$ or $\Delta(P_0P_1P_2)$, which of them will be the initial triangle or none. In the first, we orthogonally project both third vertices of such triangles to the other triangle. This fact could not guarantee smoothness conditions or low curvature around point P_0; in this case, the initial triangle's computation does not occur. In the second, the closer point P_2 to the edge $\overline{P_0P_1}$ is the only projected point to within the triangle $\Delta(P_0P_1P_k)$, so, we select the triangle $\Delta(P_0P_1P_2)$ as the initial triangle candidate, as depicted in Fig. 3.3(c). Finally, in the third, there is not any triangle projecting it to each other; hence, the choice among those two triangles occurs by applying the *relative regularity function* (3.2) over those four points, as illustrated in Fig. 3.3(d). This task ends when there are no more points closer to the initial edge to analyze; thus, the initial triangle is found.



(a) (b) (c)

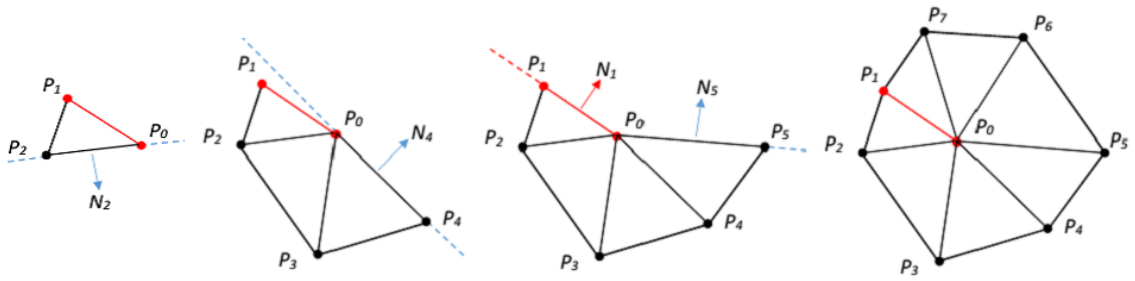
(d)

Figure 3.3: Finding the seed triangle: (a) P_1 is the closest point to P_0 and both together define the initial edge; (b) P_k makes the most regular triangle with the edge $\overline{P_0P_1}$; $\vec{N_1}$ is the normal vector with the edge $\overline{P_0P_1}$ and coplanar with the triangle $\Delta(P_0P_1P_k)$; (c) P_2 is closer to the edge $\overline{P_0P_1}$ and is projected to within the triangle $\Delta(P_0P_1P_k)$, the initial triangle candidate has changed to $\Delta(P_0P_1P_2)$; (d) P_2 is closer to the edge $\overline{P_0P_1}$ and is not projected to within the triangle $\Delta(P_0P_1P_k)$, the initial triangle is $\Delta(P_0P_1P_2)$ as the result by the application of the *relative regularity function* (3.2).

3.3.1.2 Building the smooth and manifold star

After the seed triangle has been found, Fig. 3.4(a), the next task is to build the *star* by attaching new triangles around the center point P_0 until the *star* is closed, Fig. 3.4(d). In each iteration, finding a new triangle and attaching it to the last built edge of the *star* is similar to the procedure described above to find the initial triangle from the initial edge. Before proceeding any further, we should consider a few details on determining the half-space where the next at-

taching triangle lies. Thus, taking the last built triangle, we compute the normal vector to the last built edge, which is coplanar and outward with the last built triangle, as illustrated in Fig. 3.4(a or b) by the vectors $\vec{N}_2$ or $\vec{N}_4$. However, after a few iterations, the initial edge of the seed triangle is not behind the plane defined by the normal outward vector with the last built edge, Fig. 3.4(c). In this case, we have to consider the normal outward vector to the seed triangle's initial edge to confine the half-space where we will find the next triangle to be attached. Another detail we should note is the smoothness and manifoldness guarantees. Accordingly, one must check the orthogonal projection test to validate any new attaching triangle. Thus, no point of the *star* cannot orthogonally project to within the new triangle nor vice-versa. Otherwise, one discards the new triangle; also, one abandons the computation of the in-progress *star* because one cannot ensure the star's low curvature. In this case, one proceeds to the triangle star computation for another point.



(a) (b) (c) (d)

Figure 3.4: Building the *star*: (a) $\vec{N}_2$ is the normal outward vector with the edge $\overline{P_0P_2}$; (b) the initial edge is still behind the plane defined by the vector $\vec{N}_4$; (c) the initial edge is not behind the plane defined by $\vec{N}_5$, both vectors, $\vec{N}_4$ and $\vec{N}_5$, determine the half-space where the next triangle to be attached lies; (d) the *star* is completed.

3.3.2 Finding candidate triangles

The mesh grows when one adds a new triangle to the mesh front. When a new triangle is attached to the mesh front, it adds either one, two, or three (for the seed triangle) new border edges to the mesh front. For each new border edge, one has to find the candidate triangle that supposedly will attach to it.

The procedure to identify the candidate triangle T that supposedly attaches a new border edge $\overline{\mathbf{AB}}$ comprises the following steps:

- Find the subspace $S \in \mathbb{R}^3$ beyond $\overline{\mathbf{AB}}$ where the third vertex of T lies in, as illustrated in Fig. 3.5.

- Find two points P_1 and P_2 (at maximum) in S that are the nearest points to the border edge $\overline{\mathbf{AB}}$ bounded by the endpoints $\mathbf{A}$ and $\mathbf{B}$ (see Fig. 3.6).

- If two nearest points P_1 and P_2 are found in S (see Fig. 3.6(a)-(b)), we have to decide which one is the right one; if only a single nearest point P_1 is found in S, the candidate triangle $\Delta(\mathbf{AB}P_1)$ is achieved; if no nearest point is found in S (see Fig. 3.6(c)), P_1 and P_2 are set to opposite endpoints of the border edges incident at $\mathbf{A}$ and $\mathbf{B}$, respectively, so we have to decide which one is the right one once again.

Let us now detail each of the three sub-steps outlined above.

3.3.2.1 Finding subspace S in the neighborhood of border edge $\overline{\mathbf{AB}}$

Let us now see how to form the subspace $S \in \mathbb{R}^3$ beyond the border edge $\overline{\mathbf{AB}}$. We have to determine the outwards vector $\vec{n}$ that is perpendicular to $\overline{\mathbf{AB}}$, but parallel to plane defined by the mesh triangle bounded by $\overline{\mathbf{AB}}$. Then, we determine the plane π that contains $\overline{\mathbf{AB}}$ and is defined by $\vec{n}$. As shown in Fig. 3.5, we have then to consider four possible cases. In the first case, both vertices $\mathbf{A}'$ and $\mathbf{B}'$ bounding the border edges $\overline{\mathbf{A}'\mathbf{A}}$ and $\overline{\mathbf{B}\mathbf{B}'}$ are behind π (Fig. 3.5(a)); in the second case, only one of those two vertices, either $\mathbf{A}'$ or $\mathbf{B}'$, is behind π (Fig. 3.5(b)); in the third and fourth cases, both vertices $\mathbf{A}'$ and $\mathbf{B}'$ are beyond π (Fig. 3.5(c)-(d)). Thus, S is defined by the intersection of one, two or three planar half-spaces defined by outwards vectors that are perpendicular to border edges.

3.3.2.2 Finding at most two nearest points P_1 and P_2 to border edge $\overline{\mathbf{AB}}$

These nearest points (if any) must be in the subspace S (see Fig. 3.6), and belong to $N(\mathbf{A})$ and $N(\mathbf{B})$. The nearest points are found using the distance function given by Eq. (3.13).

3.3.2.3 Finding the third vertex P of the candidate triangle $\Delta(\mathbf{AB}P)$

Let us assume that we have already found the two nearest points P_1 and P_2 relative to $\overline{\mathbf{AB}}$, as shown in Fig. 3.6. We first determine the unbounded subspace S_1 generated by the triangle $\Delta(\mathbf{AB}P_1)$ along the direction defined by its normal vector. Such subspace results from the intersection between three half-spaces, each containing an edge of the candidate triangle. Analogously, we determine the unbounded subspace S_2 generated by the triangle $\Delta(\mathbf{AB}P_2)$.

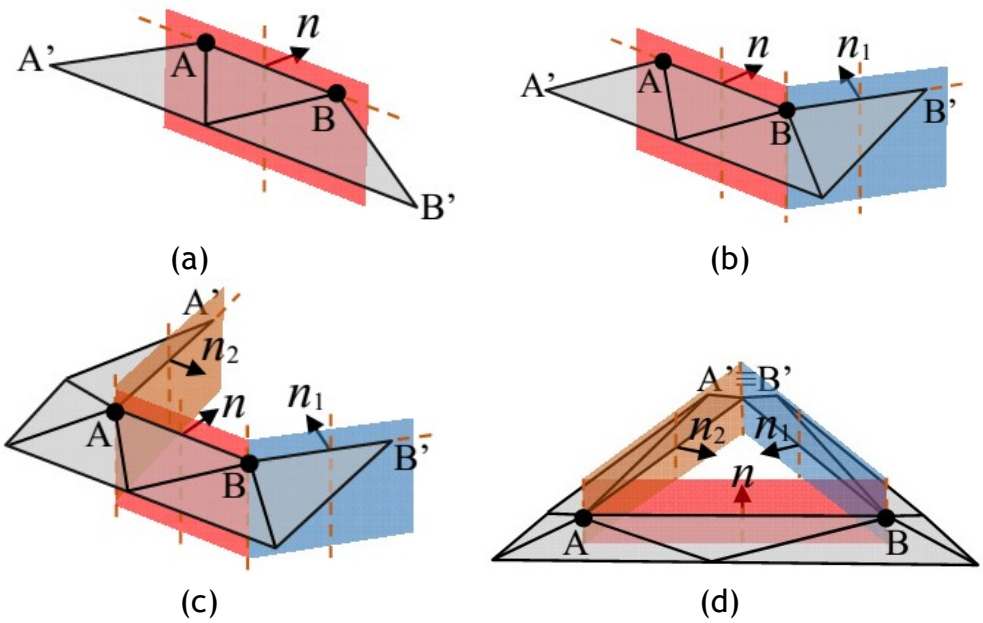


Figure 3.5: The subspace S beyond the border edge $\overline{\mathbf{AB}}$: (a) with $\mathbf{A}'$ and $\mathbf{B}'$ behind the plane π defined by $\vec{n}$; (b) with $\mathbf{A}'$ behind π and $\mathbf{B}'$ beyond π; (c) and (d) with both $\mathbf{A}'$ and $\mathbf{B}'$ beyond π.

To determine the third vertex $P \in \{P_1, P_2\}$ of the candidate triangle $\Delta(\mathbf{AB}P)$ in the neighborhood of the border edge $\overline{\mathbf{AB}}$, we consider the three cases depicted in Fig. 3.6. If $P_1 \in S_2$ but $P_2 \notin S_1$ (see Fig. 3.6(a)), then the right candidate triangle is $\Delta(\mathbf{AB}P_1)$. Otherwise, if $P_1 \in S_2$ and $P_2 \in S_1$ (see Fig. 3.6(b)), then the right candidate triangle is $\Delta(\mathbf{AB}P)$, where $P \in \{P_1, P_2\}$ is the point that is more close to $\overline{\mathbf{AB}}$. Finally, if $P_1 \notin S_2$ nor $P_2 \notin S_1$ (Fig. 3.6(c)), we choose the point $P \in \{P_1, P_2\}$ that makes the candidate triangle $\Delta(\mathbf{AB}P)$ the most regular.

In searching the third vertex of the candidate triangle, we have assumed that we had two or more points in the subspace S delimited by $\overline{\mathbf{AB}}$ and its incident neighbor border edges, $\overline{\mathbf{A}'\mathbf{A}}$ and $\overline{\mathbf{B}\mathbf{B}'}$ (Fig. 3.5). However, it may happen to have only one point P_1 in S; in this case, the candidate triangle is $\Delta(\mathbf{AB}P_1)$. But, if there is no neighbor point remaining in the subspace S, we also consider the particular cases shown in Fig. 3.7, where the third point P of the triangle (A, P, B) is a boundary vertex of the mesh front.

3.3.3 Attaching the new triangle

We have to select a triangle —and attach it to the mesh— among all candidate triangles around the mesh front (i.e., a ring of border edges). Such a triangle is the most coplanar with the triangle plane that upholds the growing front's border edge. Thus, the mesh grows preferably in regions with less curvature.

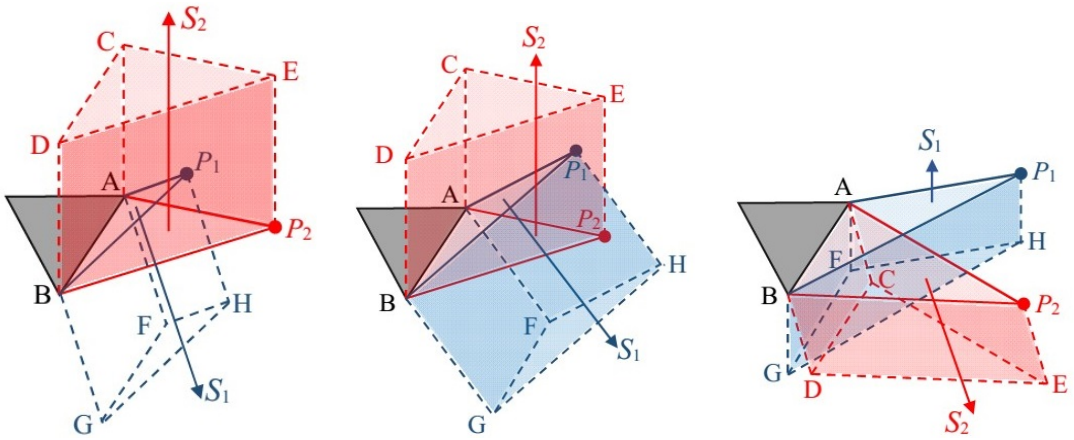


Figure 3.6: Choosing the third vertex $P \in \{P_1, P_2\}$ of the candidate triangle $\Delta(\mathbf{AB}P)$ in the neighborhood of the border edge $\overline{\mathbf{AB}}$.

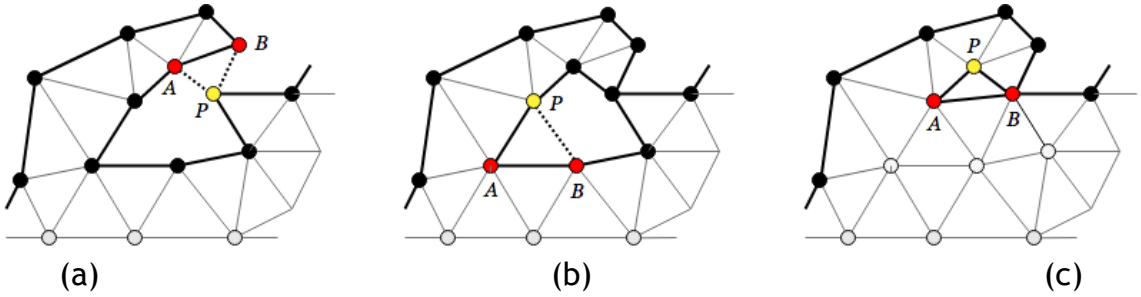


Figure 3.7: Choosing the third vertex P when there is no neighbor point in the subspace S beyond the border edge $\overline{\mathbf{AB}}$.

For example, in Fig. 3.1, we can see the reconstruction of the dolphin's fin took place after the main body because there is a fast-changing curvature of the main body to the dolphin's fin. Note that, unlike most mesh growing algorithms (see, for example, [LHW09], [WTS12] and [XLC14]), we do not use bounds to the dihedral angle between the adjacent triangles along with the mesh growing process.

The mesh front consists of one or more rings of border edges together with their triangles (see Fig. 3.7); for example, the initial mesh front consists of three edges and a single triangle, the so-called initial or seed triangle. The second triangle of the mesh is attached to one of the border edges of the seed triangle. The mesh growing algorithm terminates when no border edge of the mesh front is left out, and the resulting triangulated surface is a manifold.

3.4 Experimental Results

3.4.1 Hardware and software setup

We implemented the proposed algorithm in C++ and GLUT 3.7 (Microsoft Visual Studio 2017) on a desktop computer equipped with an Intel processor Core(TM) i3-4005U, 1.73 GHz, 6 GB RAM, running Windows 8.1. Furthermore, we used MeshLab v1.3.3 (`http://www.meshlab.net/`) to evaluate the mesh reconstruction quality of the benchmarking algorithms (including the PCR Cocktail algorithm) through the Hausdorff distance between original meshes and their corresponding reconstructed meshes.

3.4.2 Benchmarking surface reconstruction methods

The surface reconstruction method, PCR Cocktail, was compared to four methods. The first is the power crust meshing algorithm due to Amenta et al. [ACK01a] (`http://web.cs.ucdavis.edu/~amenta/powercrust.html`). In contrast, the other three are the advancing front meshing algorithm due to Cohen-Steiner and Da [CSD04], the Poisson surface reconstruction algorithm due to Kazhdan et al. [KBH06], and the scale-space reconstruction algorithm due to Digne et al. [DMSL11]. The latter three methods are part of CGAL (`http://www.cgal.org`).

3.4.3 Testing mesh models

Our testing point cloud models were obtained from triangle meshes retrieved from Princeton Shape Benchmark (`http://shape.cs.princeton.edu/benchmark/`), 3D Segmentation Benchmark (`http://segeval.cs.princeton.edu/`), tf3dm (`http://tf3dm.com/`), John Burkardt Home Page (`https://people.sc.fsu.edu/~jburkardt/data` and MeshLab Samples (`https://people.sc.fsu.edu/~jburkardt/data/meshlab/meshlab.html`), some of which are listed in Tables 3.1 and 3.2. Such triangle meshes form our ground-truth dataset for benchmarking surface reconstruction methods. Several mesh models are depicted in Figs. 3.9 and 3.10 after reconstruction through PCR Cocktail algorithm.

3.4.4 Mesh reconstruction quality

Intuitively, Hausdorff distance measures “how similar” two point sets are in the metric sense. We used the Hausdorff distance between two surface meshes as

Table 3.1: Original meshes and reconstructed meshes generated by PCR and advancing-front algorithms.

Model	Label	Original Mesh		PCR Mesh			Advancing Front Mesh		
		#Vertices	#Triangles	#Vertices	#Triangles	H	#Vertices	#Triangles	H
1	Duck	710	1,416	710	1,414	4.639	710	1,416	3.098
2	Knot	1,280	2,560	1,280	2,560	0.380	1,280	2,560	0.281
3	Loop	1,440	2,880	1,440	2,880	2.691	1,440	2,874	7.502
4	Dolphin	1,867	3,694	1,867	3,734	4.413	1,867	3,658	23.400
5	Doubletorus	4,352	8,708	4,352	8,708	0.000	4,352	8,708	3.612
6	MaxPlanck	7,399	14,749	7,399	14,796	22.446	7,399	14,794	22.261
7	Egea	8,268	16,532	8,268	16,532	1,889	8,268	16,531	1.889
8	Pear	10,754	21,504	10,754	21,504	0.153	10,754	21,504	0.127
9	Fish	14,000	27,996	14,000	28,005	3,624	14,000	27,998	5.064
10	Horse	20,000	39,996	20,000	40,004	2.721	20,000	39,994	2.591
11	Armadillo	26,002	52,000	26,002	52,013	1.936	26,002	51,995	1.467
12	Bunny	34,834	69,451	34,834	69,663	16.010	34,834	69,656	16.253

Abbreviations:
H: Hausdorff distance

Table 3.2: Reconstructed meshes generated by power-crust, scale-space, and Poisson algorithms.

Model	Label	PowerCrust Mesh			Scale-Space Mesh			Poisson Mesh		
		#Vertices	#Triangles	H	#Vertices	#Triangles	H	#Vertices	#Triangles	H
1	Duck	6,642	13,279	7.705	710	1,988	27.355	1,489	2,974	27.746
2	Knot	7,744	15,492	23.791	1,280	4,356	29.430	2,490	5,004	34.903
3	Loop	14,368	28,613	14.869	1,440	4,498	32.077	2,223	4,454	38.958
4	Dolphin	16,009	32,006	3.614	1,867	5,436	25.066	3,281	6,562	24.008
5	Doubletorus	35,064	70,132	10.997	4,352	16,752	18.413	3,176	6,356	14.921
6	MaxPlanck	66,282	132,560	22.779	7,399	28,732	8.932	11,108	22,212	22.515
7	Egea	71,882	155,970	2.065	8,268	32,818	6.827	3,568	7,132	5.778
8	Pear	104,840	209,675	1.068	10,754	37,840	25.895	2,994	5,984	28.969
9	Fish	–	–	–	14,000	44,932	21.303	12,030	24,080	21.299
10	Horse	209,879	419,744	3.641	20,000	78,936	7.549	10,264	20,524	9.200
11	Armadillo	245,683	491,364	2.418	26,002	99,346	4.908	16,032	32,060	8.432
12	Bunny	349,013	698,011	17.382	34,834	138,050	4.470	12,927	25,850	16.998

Abbreviations:
H: Hausdorff distance

our mesh reconstruction quality metric. It is defined as follows:

$$H(S, S') = \max\Big(\max_{\mathbf{x}_i \in S} \delta(\mathbf{x}_i, S'), \max_{\mathbf{x}_j \in S'} \delta(\mathbf{x}_j, S)\Big) \quad (3.14)$$

where S, S' denote the two surface meshes, the ground-truth mesh and testing mesh, with

$$\delta(\mathbf{x}_i, S') = \min_{\mathbf{x}_j \in S'} d(\mathbf{x}_i, \mathbf{x}_j) \quad \text{and} \quad \delta(\mathbf{x}_j, S) = \min_{\mathbf{x}_i \in S} d(\mathbf{x}_j, \mathbf{x}_i) \quad (3.15)$$

where $d(\mathbf{x}, \mathbf{x}')$ is the Euclidean distance between points $\mathbf{x}$ and $\mathbf{x}'$.

During the tests of our PCR algorithm, we used the MeshLab to compute the

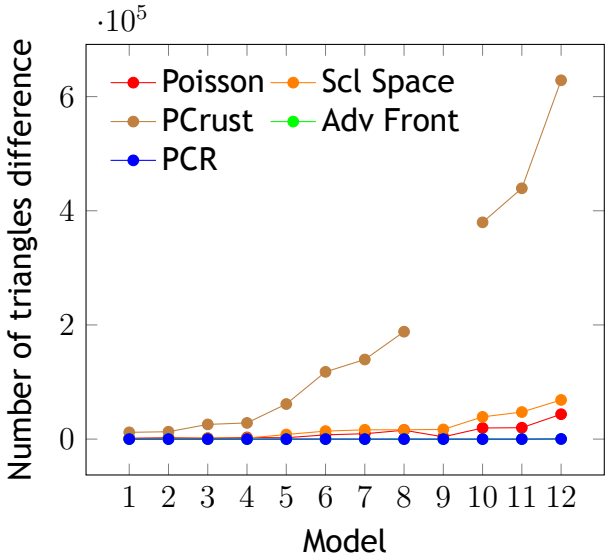


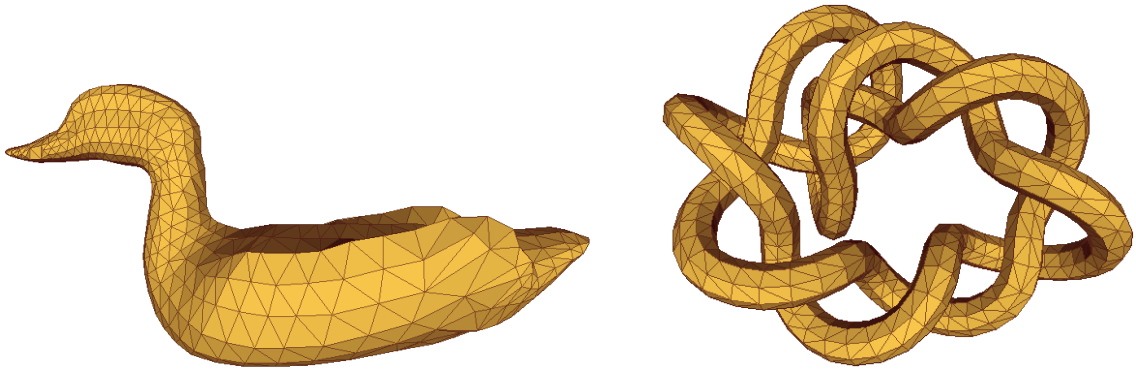
Figure 3.8: Number of triangles differs between original meshes and reconstructed meshes.

Hausdorff distance. The closer the reconstructed mesh is to the original mesh, the higher quality is of such reconstruction. According to Eq. (3.14), Hausdorff distance applies to sets of points. Therefore, the simplest way to compute the distance between two meshes consists of calculating the Hausdorff distance considering only vertices. However, the Hausdorff distance would be zero because the vertices of PCR, advancing front, and scale-space meshes are the same as those of the original mesh. We also considered barycenters of edges and triangles in computing the Hausdorff distance between meshes to overcome this problem.

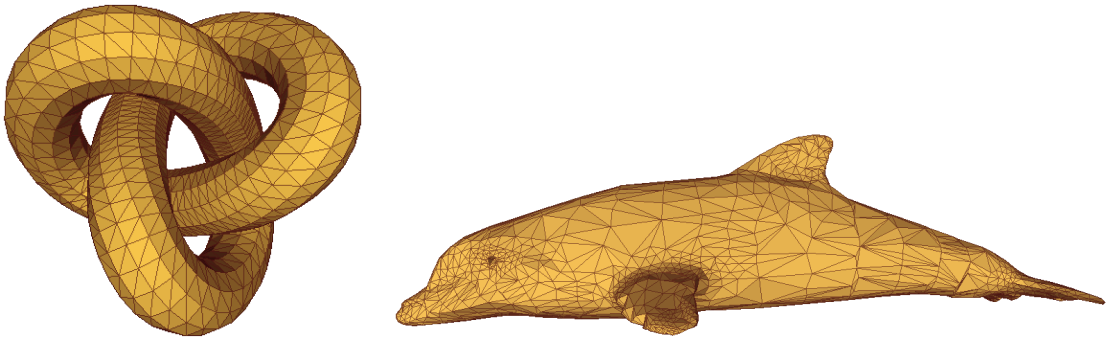
Looking at Tables 3.1 and 3.2, we observe the following:

- *Number of vertices*. The number of vertices remains unchanged for PCR Cocktail, advancing front, and scale-space algorithms, but not for Power crust and Poisson algorithms. Power crust meshing adds much more vertices to the mesh as a way of getting a uniform density of points in the cloud (cf. [ACK01a]). Poisson algorithm produces a mesh of an implicit surface that approximates the point cloud. The triangulation of the implicit surface explains why the number of vertices (and their locations) varies relative to the original ones.

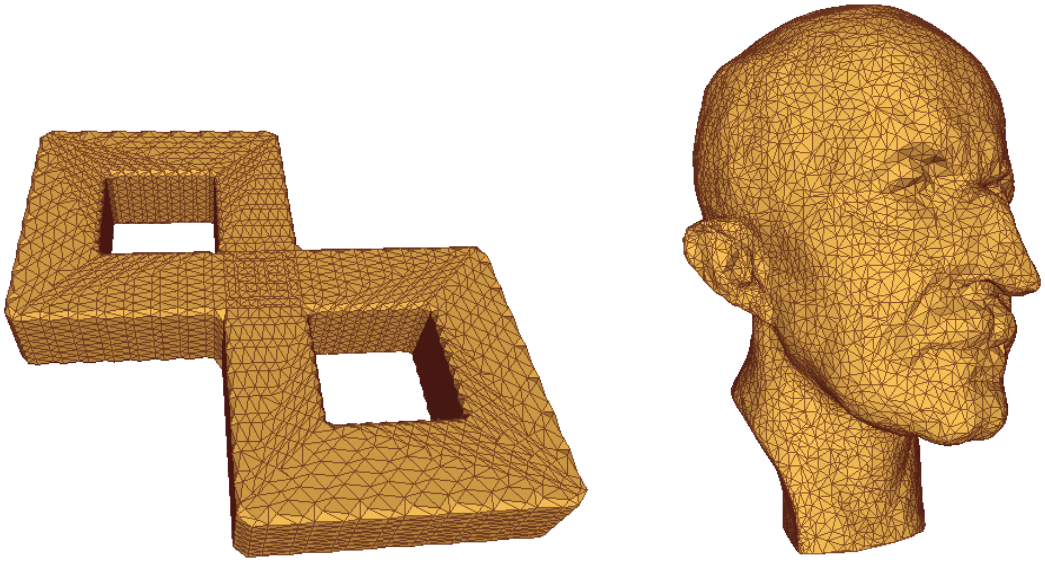
- *Number of triangles*. The number of triangles tends to remain unchanged for simplicial methods like PCR Cocktail and advancing-front methods (see Table 3.1). This fact does not apply to the power-crust method because it adds more points to the input point set. On the other hand, Poisson and



(a) Duck (b) Knot



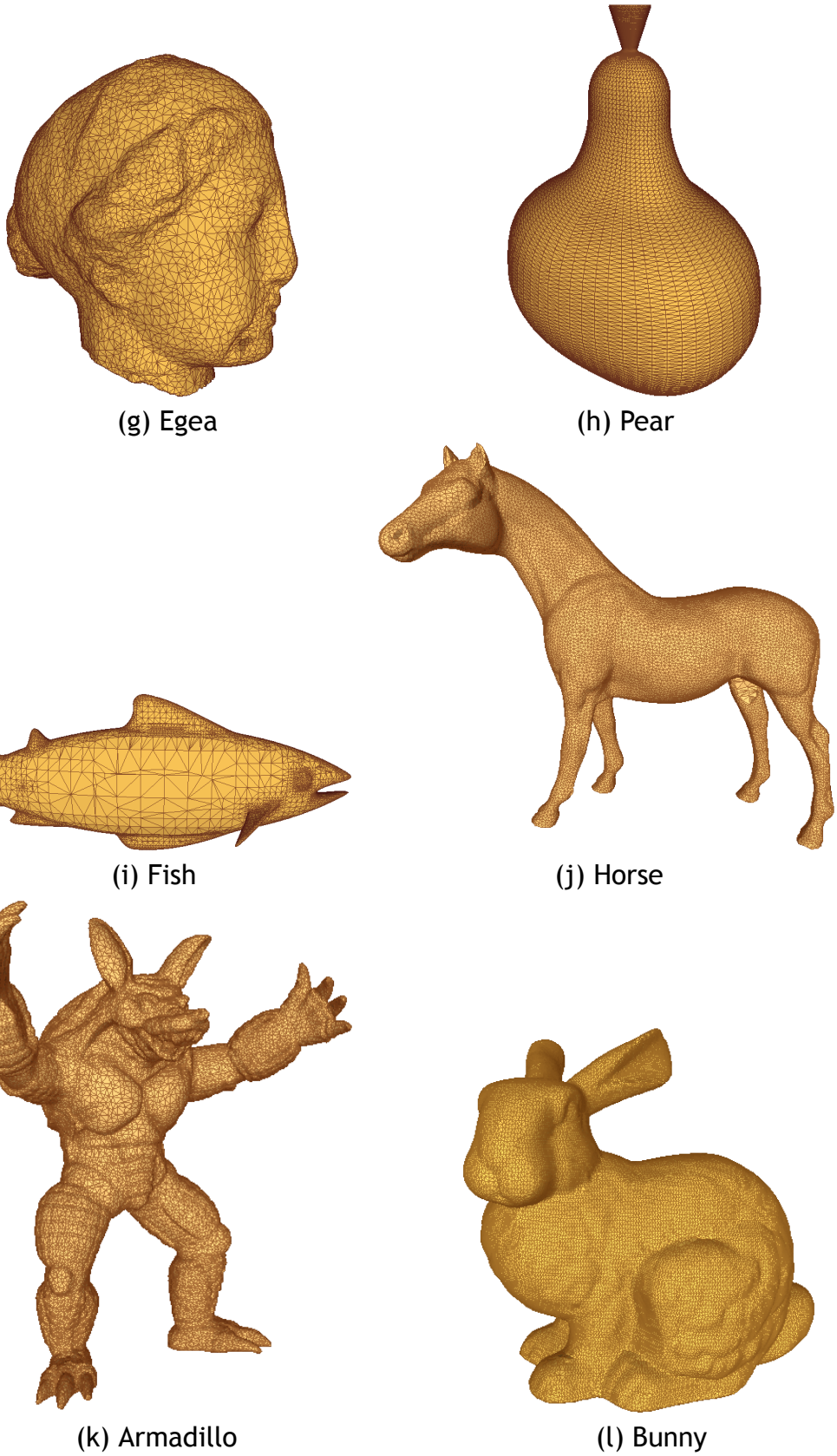
(c) Loop (d) Dolphin



(e) Doubletorus (f) MaxPlanck

Figure 3.9: Reconstructed surface meshes using the PCR Cocktail.

scale-space reconstruction methods produce implicit surfaces that approximate the input point cloud, after which one proceeds to the triangulation of such surfaces. In other words, implicit methods generate triangles that do not necessarily match those of the original mesh, as shown in Table 3.2. Besides, Fig. 3.8 indicates the number of triangles difference between the original meshes and the reconstructed meshes, PCR Cocktail and Advanc-



(g) Egea (h) Pear

(i) Fish (j) Horse

(k) Armadillo (l) Bunny

Figure 3.10: More reconstructed surface meshes using the PCR Cocktail.

ing front present the lowest differences.

- *Hausdorff distance*. As shown in Tables 3.1 and 3.2, the Hausdorff distance between the original meshes and the meshes generated by the PCR Cocktail algorithm is, in general, smaller than for the benchmarking surface reconstruction methods (cf. Fig. 3.11). Therefore, our algorithm produces more reliable or high-quality meshes than the other ones, as can be observed in Figs. 3.9 and 3.10. Recall that the Hausdorff distance is the maximum of the minimum distance between two sets of points (see Eq. (3.14)).

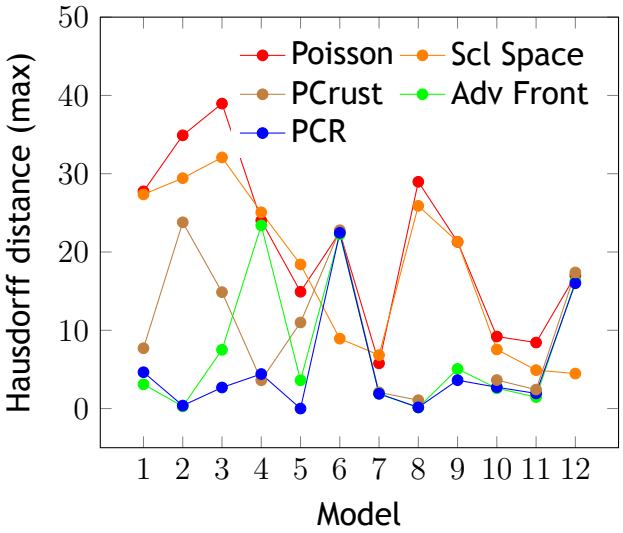


Figure 3.11: Hausdorff distance for reconstructed meshes, comparing PCR Cocktail, Power Crust, Advancing Front, Scale-Space and Poisson algorithms.

Overall, a glance at Figs. 3.11-3.14 allows us to observe implicit methods (i.e., Poisson and scale-space surface reconstruction algorithms) generate meshes that are further away from the original meshes than simplicial methods (i.e., power crust, advancing front, and PCR Cocktail algorithms). This deviation is due to the approximating approach of implicit methods rather than the simplicial methods' interpolation approach. Moreover, the blending of kernel functions of implicit methods causes rounding of sharp features (e.g., apices and creases) of the original object.

It is worth noting both mesh growing algorithms, say PCR Cocktail and advancing front algorithms, perform in a similar way regarding the Hausdorff distance (see Fig. 3.11). To make sure about which one of these two methods ranks first, we carried out a finer-grain analysis of results in terms of other Hausdorff metrics as follows:

- *Hausdorff distance's mean*. This measure represents the mean distance

between two sets of points, that is,

$$d_M = \frac{1}{n+m}\Big(\sum_{i=1}^{n}\delta(\mathbf{x}_i, S') + \sum_{j=1}^{m}\delta(\mathbf{x}_j, S)\Big) \quad (3.16)$$

where $\delta(\mathbf{x}_i, S')$ is the minimum distance from the point $\mathbf{x}_i \in S$ to the point set S' and $\delta(\mathbf{x}_j, S)$ is the minimum distance from the point $\mathbf{x}_j \in S'$ to the point set S, as given by Eq. (3.15). Fig. 3.12 shows that simplicial algorithms outperform implicit algorithms in terms of Hausdorff mean distance. Besides, PCR Cocktail and advancing-front algorithms rank first and are indistinguishable concerning Hausdorff's mean distance.

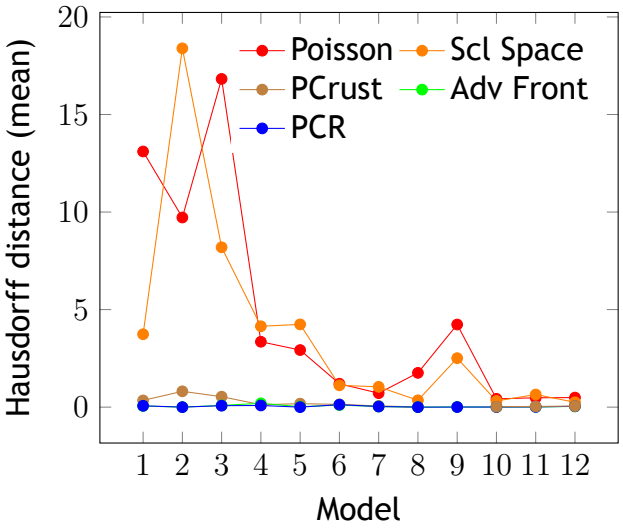


Figure 3.12: Hausdorff distance's mean for reconstructed meshes, comparing PCR Cocktail, Power Crust, Advancing Front, Scale-Space and Poisson algorithms.

- *Hausdorff distance's root mean square* (RMS). This metric measures the arithmetic mean of the squares of the set of minimum distances between two point sets S and S', that is,

$$d_{RMS} = \sqrt{\frac{1}{n+m}\Big(\sum_{i=1}^{n}(\delta(\mathbf{x}_i, S'))^2 + \sum_{j=1}^{m}(\delta(\mathbf{x}_j, S))^2\Big)}. \quad (3.17)$$

As shown in Fig. 3.13, we see that PCR Cocktail ranks first and produces triangulations more similar to the original meshes than the advancing front algorithm, which agrees with the results shown in Fig. 3.11. Note that the most different triangulations relative to the original are the fourth and sixth models (i.e., Dolphin and Max Planck models). Still, the lack of similarity is even more striking in the triangulations generated by the

advancing front algorithm.

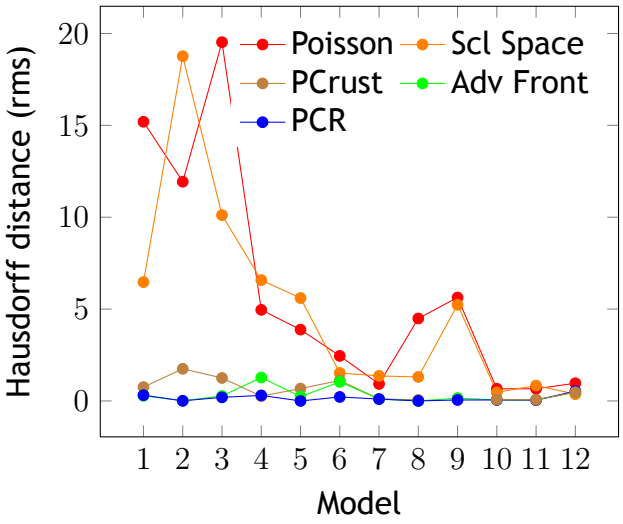


Figure 3.13: Hausdorff distance's root mean square for reconstructed meshes, comparing PCR Cocktail, Power Crust, Advancing Front, Scale-Space and Poisson algorithms.

- *Hausdorff distance's standard deviation*. This metric allows us to measure the dispersion of the minimum distances in relation to the Hausdorff's mean distance (see Eq. (3.16)), that is,

$$d_{SD} = \sqrt{\frac{1}{n+m}\Big(\sum_{i=1}^{n}(\delta(\mathbf{x}_i, S') - d_M)^2 + \sum_{j=1}^{m}(\delta(\mathbf{x}_j, S) - d_M)^2\Big)}. \quad (3.18)$$

  As shown in Fig. 3.14, PCR Cocktail has the lower dispersion of the distances for all models.

In short, PCR Cocktail produces the most similar triangulations to the original meshes. In general, simplicial methods generate triangulations that tend to be identical to the original meshes because of their interpolating approach. On the other hand, the approximating approach behind implicit methods generates triangulations that are further away from the original meshes, mainly when the implicit surface results from local functions' blending.

3.4.5 Surface reconstruction times

Another comparison we have observed was the time performance of surface reconstruction algorithms. Table 3.3 shows the times taken to the mesh reconstruction by each algorithm; see also Fig. 3.15. Besides the reconstructed surfaces' quality, we note the PCR algorithm also presents the shortest times for each mesh reconstruction. As we observe in Tables 3.1 and 3.2, the power-crust

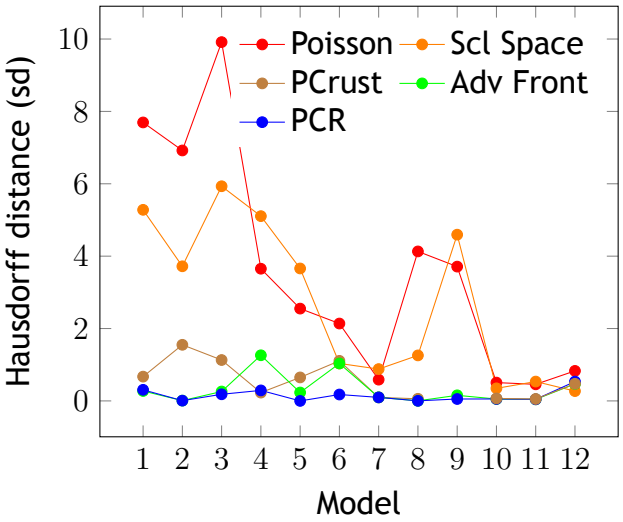


Figure 3.14: Hausdorff distance's standard deviation for reconstructed meshes, comparing PCR Cocktail, Power Crust, Advancing Front, Scale-Space and Poisson algorithms.

algorithm uses a linear approximation method to increase the density of points of the point cloud, so new vertices are part of the mesh. This fact explains why the power-crust algorithm reconstruction takes a long time in the surface reconstruction compared to PCR. In turn, the Poisson algorithm's implicit nature explains the poor time performance results, particularly in the larger point clouds.

Table 3.3: Time performance for surface reconstruction by every algorithms.

		Original Mesh		Time (seconds)				
Model	Label	#Vertices	#Triangles	PCR	Adv Front	Scl Space	PCrust	Poisson
1	Duck	710	1,416	0.024	0.051	0.045	1.259	0.863
2	Knot	1,280	2,560	0.041	0.067	0.072	2.172	1.358
3	Loop	1,440	2,880	0.049	0.086	0.093	3.200	1.285
4	Dolphin	1,867	3,694	0.075	0.124	0.134	2.809	2.379
5	Doubletorus	4,352	8,708	0.175	0.297	0.372	10.955	2.627
6	MaxPlanck	7,399	14,749	0.334	0.441	0.504	11.050	3.619
7	Egea	8,268	16,532	0.368	0.520	0.595	18.554	3.863
8	Pear	10,754	21,504	0.372	0.663	1.086	41.161	3.663
9	Fish	14,000	27,996	0.658	0.976	1.044	–	13.848
10	Horse	20,000	39,996	1.176	1.328	1.605	31.500	9.435
11	Armadillo	26,002	52,000	1.423	1.733	2.018	45.957	15.115
12	Bunny	34,834	69,451	2.180	2.240	2.635	71.754	14.774

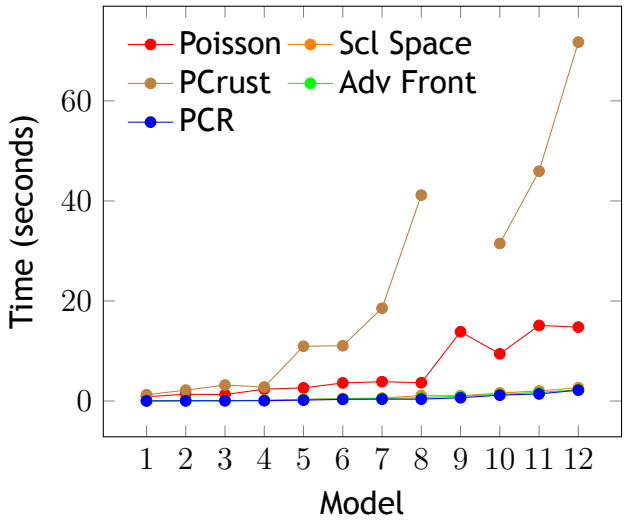


Figure 3.15: Times for reconstructed meshes using PCR Cocktail, Power Crust, Advancing Front, Scale-Space and Poisson algorithms.

3.5 Discussion

We observed from our experiments that all benchmarked surface reconstruction algorithms work well for uniform, high-density point clouds. Therefore, non-uniform, low-density point sampling is the principal problem faced by surface reconstruction algorithms and the source for most mesh reconstruction quality issues: mesh drifting, holes, incomplete meshing, and trimming and rounding off sharp features (e.g., apices and creases).

We also noted that the benchmarked implicit algorithms are more sensitive to sampling non-uniformity than simplicial methods. As shown in Figs. 3.16(c)-(d), implicit methods may misbehave when the input point cloud does not possess a uniform distribution and high density of the points, resulting in incomplete or weird triangulations. As shown in Figs. 3.16(a)-(b), simplicial methods are more robust because the meshing tends to have fewer holes and less prone to unexpected triangulations than implicit methods. However, we noted the power-crust algorithm could not reconstruct the fish model (Table 3.2) at all, possibly because in high-density zones, the distance between points is less than its pre-defined minimum distance. In practice, the power-crust algorithm does not work for point clouds with overdense regions of points either.

PCR algorithm successfully copes with these extreme situations because it gracefully combines the coplanarity criterion and clustering of points via octree subdivision of the point cloud. The octree subdivision ensures the triangulation of low-density points to those in surroundings, even when the original surface is

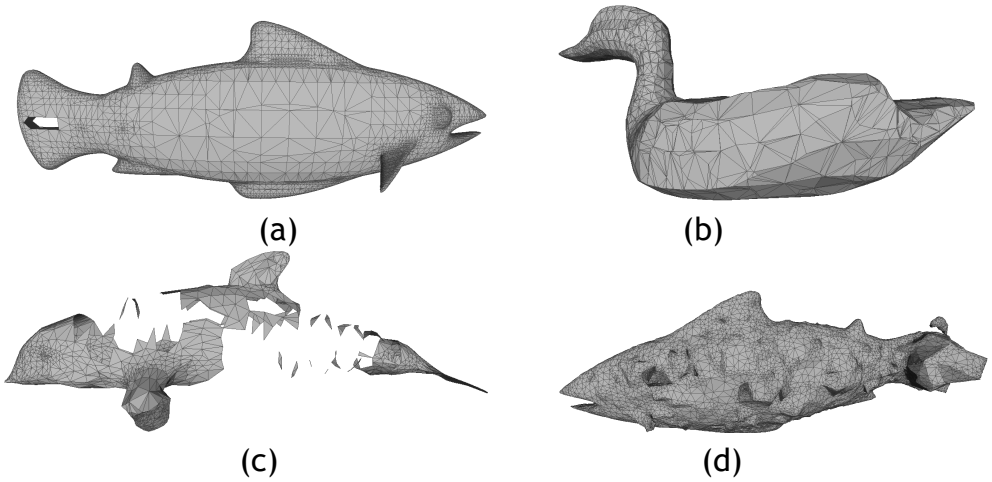


Figure 3.16: Non-uniform point sampling issues: (a) fish's meshing using advancing front algorithm originates a hole through the mesh; (b) duck's meshing using power crust algorithm increases its volume as a result of more points added to the initial point cloud; (c) incomplete dolphin's meshing using scale-space algorithm; (d) weird fish's meshing using Poisson algorithm.

not closed (i.e., a surface with a boundary).

Taking into consideration the Hausdorff distance results shown in Fig. 3.11, we see the worst surface mesh reconstructions using PCR Cocktail are those concerning Max Planck and bunny models (see Tables 3.1-3.2 and Figs. 3.9(f) and 3.10(l)). These results are so because their original meshes are not closed meshes underneath. Therefore, the algorithm attaches new triangles to the mesh to close any mesh border to the outside. In other words, new edges and facets that act as hole caps increase the Hausdorff distance between the original mesh and the reconstructed mesh.

3.5.1 Mesh drifting

As shown in Fig. 3.17, the four competitor benchmarking algorithms suffer from the mesh drifting problem, which translates itself into internal shortcuts (or holes), external shortcuts, and components (separate parts). This problem occurs when two or more non-adjacent regions of the surface get nearby to each other. For example, in Fig. 3.17(a), we see the advancing front reconstruction algorithm makes an internal shortcut or hole when the dolphin's surface passes a short distance from itself, as is the case of the hole through the left pectoral fin. Additionally, the shortcut in the tail fin (or fluke) put it apart into two components. But, as suggested by triangulations in Fig. 3.17, implicit methods are more prone to mesh drifting than simplicial methods. This drifting phenomenon explains itself by the inflating (and blending) effect of implicit methods in ap-

proximating the input cloud's points, making non-adjacent, nearby mesh regions even closer to each other. This fact agrees with the Hausdorff distance's results obtained in the previous section. In contrast, PCR Cocktail's coplanarity criterion prevents mesh drifting effects during the reconstruction process, as shown in Figs. 3.9 and 3.10.

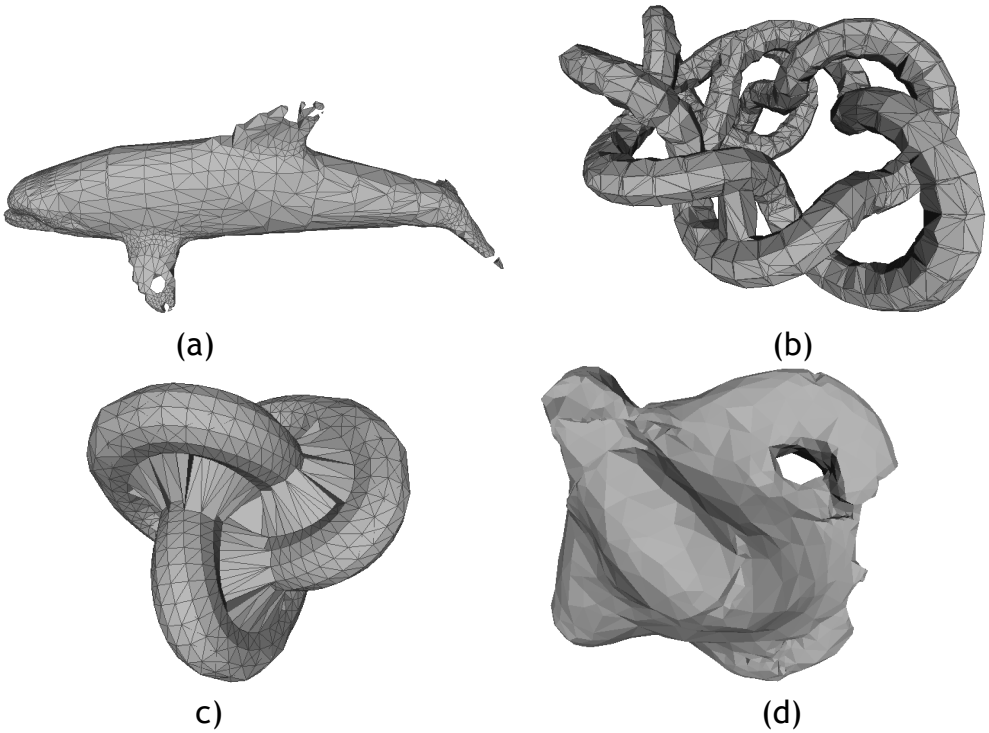


Figure 3.17: Mesh drifting issues: (a) dolphin's meshing using advancing front algorithm; (b) knot's meshing using power crust algorithm; (c) loop's meshing using scale space algorithm; and (d) loop's meshing using Poisson algorithm.

3.5.2 Trimming and rounding of sharp features

We observed that simplicial algorithms tend to trim sharp features, whereas implicit algorithms tend to round them, as illustrated in Fig. 3.18. Regarding PCR Cocktail, we can say it deals very well with creases and corners because the triangulation grows from flat regions to sharp features due to applying the coplanarity criterion. In fact, as illustrated in Figs. 3.9 and 3.10, where we see the creases of duck's wings and dolphin's back fin (Fig. 3.9(a) and (d)), as well as the corner of fish's right fin (Fig. 3.10(i)), are correctly triangulated. Thus, in principle, reconstructing sharp features of mechanical parts is also feasible.

3.5.3 Dihedral angle and internal angles bounds

Typically, mesh-growing algorithms suffer from the problem of dihedral angle bounds. Such algorithms impose minimum and maximum internal angles of each

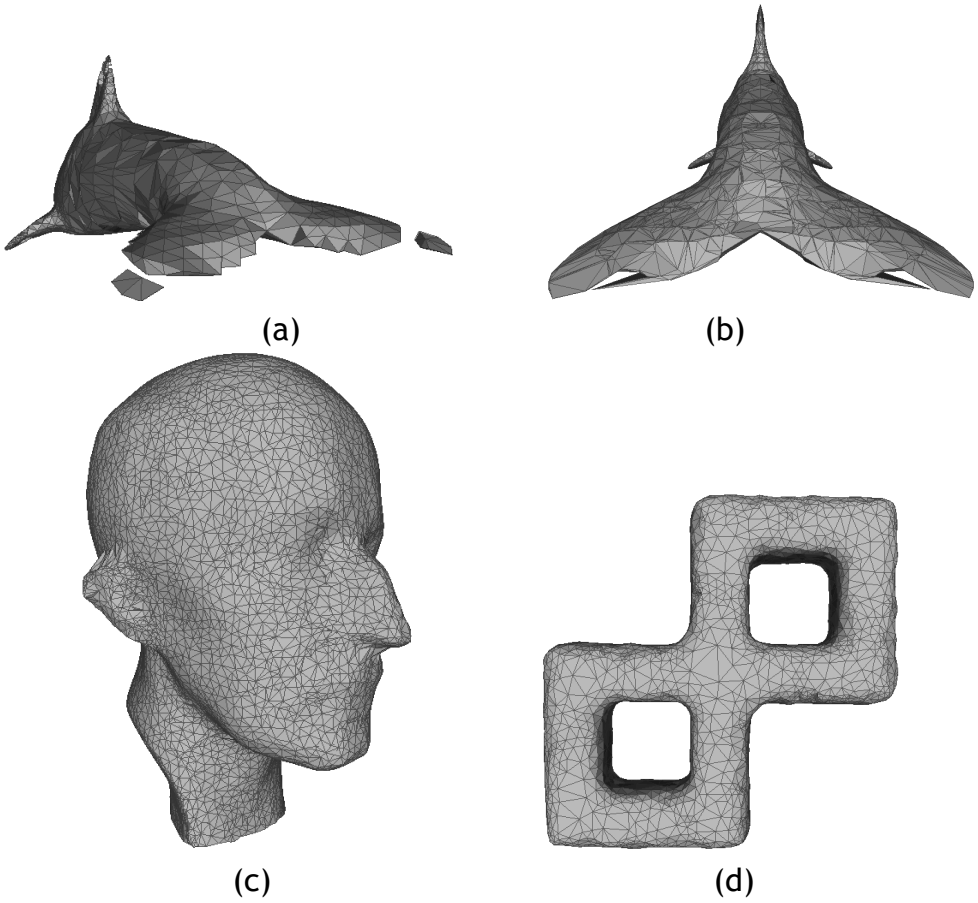


Figure 3.18: Sharp features issues: (a) dolphin's meshing using advancing front algorithm originates a total trimming of its fluke; (b) dolphin's meshing using power crust algorithm originates a partial trimming of its fluke; (c) Max Planck's meshing using scale space algorithm; and (d) double torus's meshing using Poisson algorithm.

new triangle attached to the mesh front. Moreover, such algorithms also set a maximum dihedral angle between the new triangle and its corresponding triangle in the mesh front. This problem does not exist in our algorithm because PCR Cocktail prioritizes attaching triangles with the lowest dihedral angle with mesh front triangles; hence, the coplanarity criterium. Besides, the coplanarity and regularity criteria ensure that our triangulation consists of triangles that tend to be regular. Therefore, PCR Cocktail does not impose any bounds on triangles' internal angles and the adjacent triangles' dihedral angle.

3.6 Summary

We have introduced the PCR Cocktail algorithm, a surface mesh reconstruction algorithm that belongs to interpolation methods. Our algorithm combines three geometric properties: proximity, regularity, and coplanarity. These properties

allow us to decide the next triangle to be attached to the growing mesh front.

PCR algorithm gives priority to smoother regions during mesh growing process, which means lower curvature regions or even planar regions are triangulated before higher curvature regions, as illustrated in Fig. 3.1. As a consequence of the coplanarity criterion, the triangulation of sharp features takes place after lower curvature regions. Another result of that priority is ridding of eventual mesh drifting effects.

This algorithm constructs a tendentiously regular triangle mesh; that is, triangles tend to be equilateral. This quasi-regular triangulation makes unnecessary any regularization step after the mesh construction. The use of regularity criteria avoids imposing any bounds on the internal angles of triangles. Besides combining regularity and coplanarity criteria, one does not need to use any dihedral angle bounds between adjacent triangles.

The proximity criteria and that meshing priority given to flat regions make the PCR Cocktail algorithm capable of successfully dealing with point clouds with variable point density. Moreover, this algorithm neither pre-determines a maximum radius nor pre-determines a minimum number of points for the point neighborhoods.

To the best of our knowledge, no other mesh growing algorithm reunites these properties in an integrated manner. Even algorithms belonging to different categories hardly enjoy these properties simultaneously.

Chapter 4

Surface Reconstruction: CTC Algorithm

4.1 Introduction

This chapter proposes another algorithm to reconstruct a manifold mesh from a point cloud, called *Compatible Triangle Charts* (CTC). This algorithm builds upon the concept of the manifold star, already used in the PCR algorithm. But, unlike PCR algorithm, the CTC algorithm determines a manifold star of triangles (or triangle chart) for every cloud point, making such stars compatible with one another, much like in the spirit of the compatible charts to build an atlas in topology and differential geometry [Laf15]. Therefore, the CTC algorithm is not a mesh growing algorithm because it does not grow the mesh from any mesh front. Indeed, it is a new type of meshing algorithm, an atlas-based surface reconstruction algorithm.

CTC algorithm triangulates the point cloud without normal vectors, and the vertices of the final mesh are only input cloud points. A *curvedness criterion* guides the triangulation flow, giving priority to low curved charts (or more planar charts). Note that each triangle chart is topologically equivalent to $\mathbb{R}^2$. This priority-based strategy is similar to the one used in the PCR algorithm, as it avoids the shape drifting phenomenon and helps in dealing with sharp features. This priority-based strategy discards dihedral angle bounds between adjacent triangles and does not impose bounds on the generated triangles' internal angles. The *manifoldness criterion* is necessary for the chart compatibility stage to ensure that the final triangle mesh corresponds to a manifold surface. Furthermore, the CTC method produces meshes whose triangles tend to be regular; that is, there is no need for a regularization step after the mesh construction.

4.2 Background

Before proceeding any further, let us introduce a few fundamental concepts to understand the CTC algorithm better. Those concepts have to do with the manifoldness and curvedness of triangle charts.

4.2.1 Manifoldness

Let us begin by classifying edges and triangles concerning manifoldness. Any candidate triangle (and its bounding edges) of a chart must satisfy the manifoldness criterion. A 2-dimensional mesh is topologically equivalent to a 2-manifold if all its triangles meet the manifoldness condition. A mesh triangle is manifold if and only if each of its points has a mesh neighborhood topologically equivalent to the 2-dimensional Euclidean space; the same applies to mesh edges.

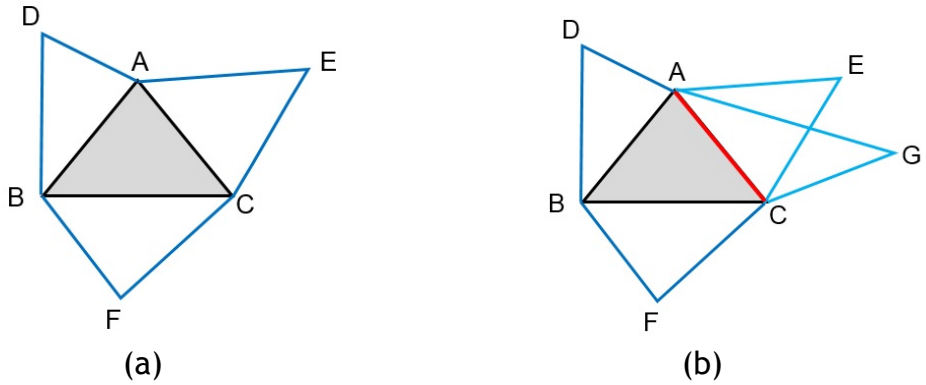


Figure 4.1: Triangle manifoldness: (a) the triangle ABC is manifold; (b) the triangle ABC is not manifold.

Manifold edge: A manifold edge has precisely two incident triangles. For instance, in Fig. 4.1, the edge AC on the left-hand side is manifold because it has two adjacent triangles, whereas the edge AC on the left-hand side has three incident triangles so that it is not a manifold edge.

Manifold triangle: A triangle is manifold if its edges are all manifold. As shown in Fig. 4.1, ABC on the left-hand side is a manifold triangle because its edges are all manifold. Interestingly, this also means that this triangle ABC belongs to the triangle chart of each vertex, A, B, and C. However, the triangle ABC on the right-hand side is not manifold because the edge AC is not manifold.

Note that the manifoldness criterion is paramount to remove redundant triangles incident on edges, as needed to comply with triangle charts' compatibility. For example, we must discard one of the triangles incident on the edge AC on the right-hand side in Fig. 4.1 to preserve the manifoldness condition.

4.2.2 Planarity degree of a triangle chart

Considering that surface reconstruction is performed prioritizing more planar charts, we need to come across a planarity metric. In a way, planarity is the antonym of curvedness because a more planar triangle chart means a less curved

chart. We might also use the classical surface curvature measures (e.g., the Gaussian curvature and the mean curvature) at a surface point or even their discrete forms. However, often they are not indicative of the local shape of a surface [KvD92].

A triangle star's planarity is determined by the coplanarity degree of its adjacent triangles. Thus, the supplementary angle (i.e., the one that adds up to 180°) to the dihedral angle has to be computed (see Fig. 4.2). In practice, one calculates the cosine of the supplementary angle to evaluate the planarity degree. This calculation is so because the cosine is a normalized function and the maximum coplanarity also corresponds to the maximum cosine value of the supplementary angle (i.e., 0°). Therefore, we can define the planarity degree of a triangle star or chart as the minimum cosine value of the supplementary angle for any two adjacent triangles of such a star. For example, in Fig. 4.3, the triangle star on the left-hand side has a higher planarity degree than the star on the right-hand side.

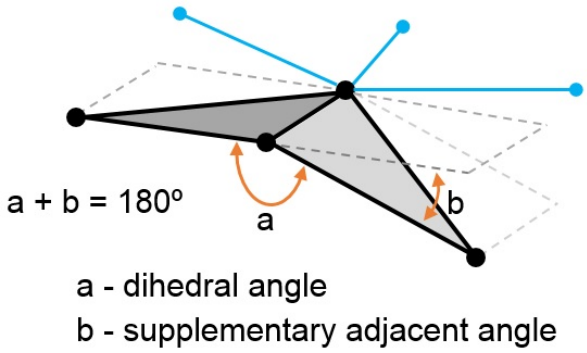


Figure 4.2: Dihedral angle between two adjacent triangles and the supplementary angle.

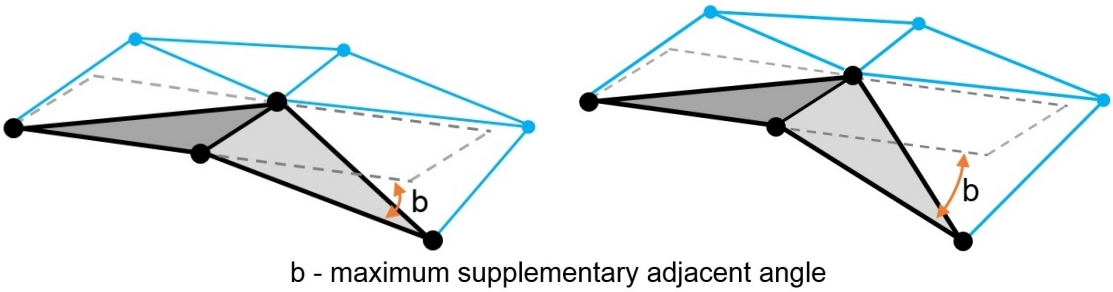


Figure 4.3: The maximum supplementary angle on the left-hand side is less than the one on the right-hand side.

4.2.3 Planarity degree of a triangle neighborhood

Before proceeding any further, let us say that the triangle charts mentioned above are not indeed triangle stars but vertex stars to speed up the algorithm

and use less memory space. A vertex star has a central vertex and a sequence of vertices around it; in other words, a vertex star is a triangle star without its triangles, as its vertices remain.

The triangulation or meshing process starts effectively after building all the stars. Therefore, we need to pick up which triangles will be attached to the triangle mesh. As seen further ahead, we will use a few criteria to select a triangle and attach it to the mesh. Such criteria are the manifoldness and planarity of each triangle and its neighbor triangles.

Let us then see the criterion concerning the planarity of the neighborhood of each triangle. To better illustrate how to determine the planarity degree of a triangle neighborhood, we first look at this problem's 2-dimensional setting. So, the word "triangle" must be replaced by the word "edge", and the expression "triangle neighborhood of a triangle" gives place to "edge neighborhood of an edge." An edge neighborhood's planarity in the 2-dimensional setting is determined using the inscribed angle in a circle and a half-circle.

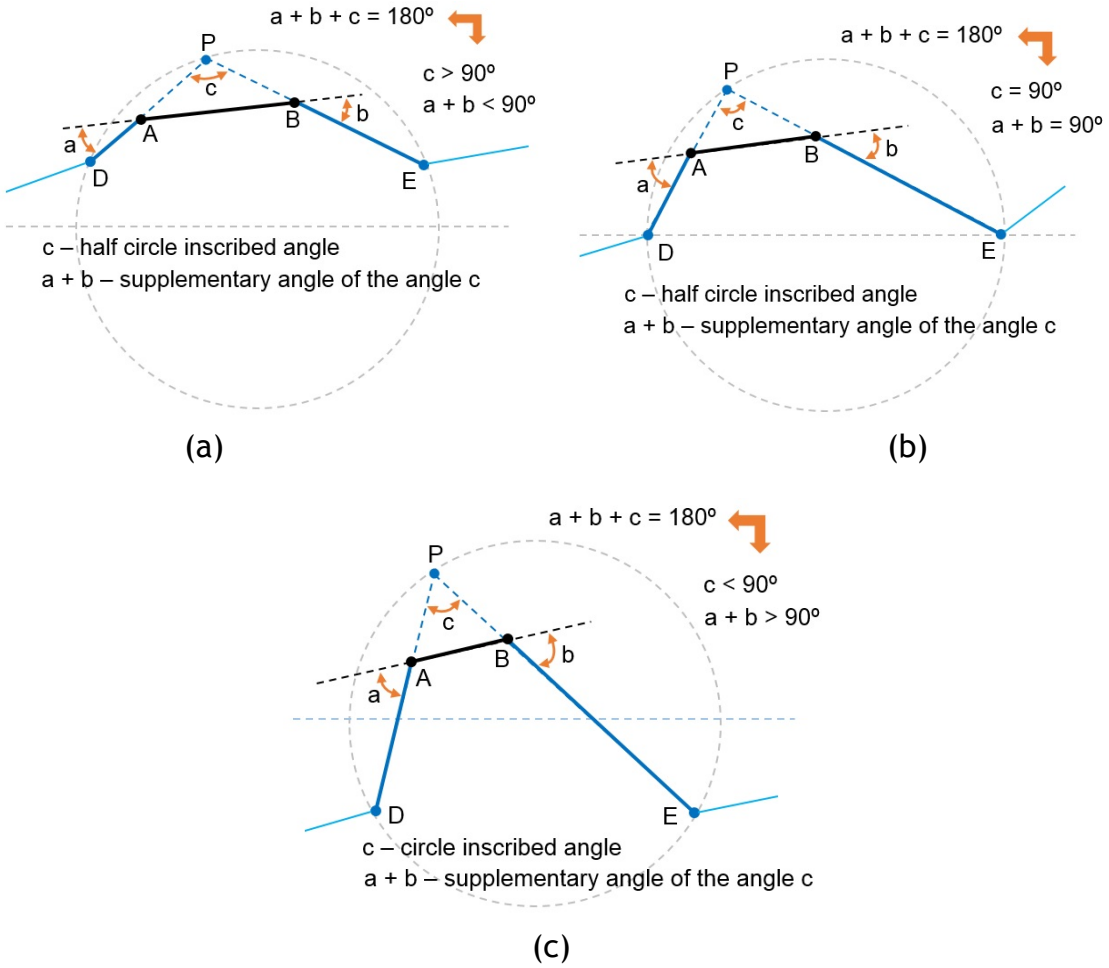


Figure 4.4: Inscribed angle of a polygonal chain within the circle.

In Fig. 4.4(a)-(c), the edge AB and its adjacent edges AD and BE constitute an edge chain. The straight lines supporting AD and BE meet at point P with the angle c. Recall that the inscribed angle in a semicircle is exactly $90°$ (see angle c in Fig. 4.4(b)). Also, the inscribed angle in an arc smaller than semicircle is always greater than $90°$ (see angle c in Fig. 4.4(a)). Otherwise, the angle inscribed in an arc bigger than a semicircle is always less than $90°$ (see angle c in Fig. 4.4(c)). In these circumstances, the sum of the two external adjacent angles, a and b, is supplementary to the inscribed angle. Consequently, if the sum of the two external adjacent angles is $90°$ at maximum, then such edge chain is laid in a half circle (Fig. 4.4(a) and (b)); otherwise, that edge chain is laid in a circle (Fig. 4.4(c)). Thereby, the inscribed angle defined by adjacent edges allows us to evaluate an edge chain's local planarity.

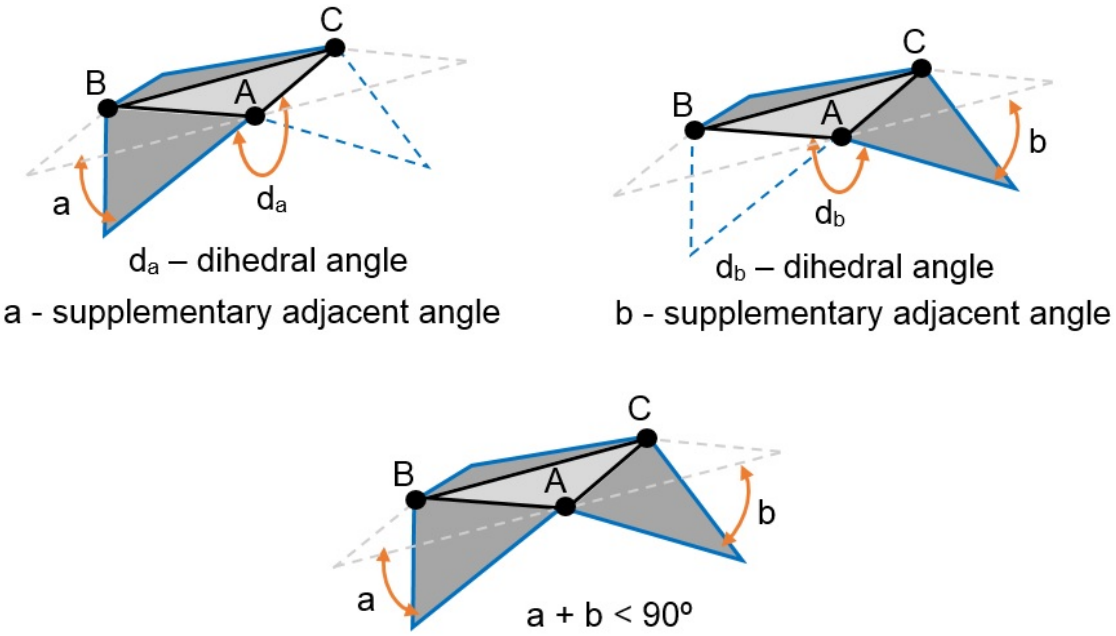


Figure 4.5: Minimum planarity criterion: the sum of both supplementary adjacent angles to the dihedral angles at edges AB and AC must be $90°$ in maximum.

Analogously, the planarity degree of each triangle depends on its three adjacent triangles in the 3D space. For that purpose, we determine the supplementary angle to the inscribed angle between two triangles that are adjacent to the reference triangle, as illustrated in Fig. 4.5. Let us consider the reference triangle ABC and two of its adjacent triangles at edges AB and AC. Thus, the sum of both supplementary adjacent angles to the dihedral angles at edges AB and AC must be $90°$ in maximum. Applying this criterion to each pair of adjacent triangles around the reference triangle, it comes up with the following conjunction of three conditions.

The minimum planarity criterion: Given a manifold triangle and its three adjacent triangles, and considering each pair of these adjacent triangles, the sum of both supplementary angles to the dihedral angles must be 90° at maximum. This fact leads to the conjunction of the following three conditions:

$$a + b < 90^\circ \wedge b + c < 90^\circ \wedge c + a < 90^\circ \quad (4.1)$$

where a, b, and c correspond to the supplementary angles to the dihedral angles at edges AB, AC and BC, respectively (Fig. 4.5).

As mentioned above, this conjunction given by Eq. 4.1 will be one of the criteria we will use to decide whether each candidate triangle will be part of the reconstructed surface.

4.3 CTC Triangulation

The CTC algorithm essentially consists of the following main steps, which are detailed in the next subsections.

1. Point cloud octree subdivision;
2. Find neighbor points for each point;
3. Build triangle stars;
4. Sort triangle stars by planarity degree;
5. Reconstruct the surface by attaching triangles to its mesh;
6. Fill in surface holes with triangles;

4.3.1 Point cloud octree subdivision

We use an octree as an acceleration data structure to determine each cloud point's neighborhood; that is, the points neighboring each point. The leading idea here is to determine the triangle star of each point in a subsequent step. After finding the bounding box enclosing the point cloud, one proceeds to its octree subdivision. We do not use a regular division of the bounding box into voxels (i.e., equally-sized cubes) because the cloud points do not equally distribute in the space; that is, the point cloud is not necessarily uniform in terms of point density. The octree subdivision of a cell stops whenever the cell con-

tains a number of points below a pre-defined threshold. In our experiments, we have used 24 points for such a threshold. This threshold is large enough to ensure that the leaf cells preserve surface connectivity from one cell to another, avoiding as much as possible getting isolated leaf cells containing points.

4.3.2 Finding neighbor points for each point

After terminating the octree subdivision, the leaf cells contain all the cloud points. Usually, the octree subdivision of the bounding box enclosing the cloud points is static because each cube cell's size remains unchanged. This fact implies that finding the neighbor points around each point must be performed inside its leaf cell and adjacent leaf cells.

To reduce the number of neighbor points for each point (i.e., reference point), we use the procedure illustrated in Fig. 4.6. First, we determine a local bounding box for the points inside its cell (i.e., reference cell), as well as a local bounding box for the points inside each one of its adjacent cells (see Fig. 4.6(a) and (c)). Next, for each cell, one computes the distance from the bounding box barycenter to its closest corner (see d_1, d_2, and d_3 in Fig. 4.6(a) and (c)), enlarging then each bounding box up to a square box whose size doubles the distance computed previously (see Fig. 4.6(b) and (d)). Finally, if any enlarged bounding box intersects the enlarged bounding box of the reference (or current) cell (Fig. 4.6(b)), its neighbor points remain as neighbor points of the reference point; otherwise, they are discarded (Fig. 4.6(d)).

For example, in Fig. 4.6, the reference cell is C_2, and any of its points is a reference point. The cells C_1 and C_3 are adjacent cells. But, C_3 will be discarded as an adjacent cell of C_2 because their enlarged boxes do not intersect. Consequently, the points of C_3 do not belong to the set of neighbor points of C_2. In the end, any point inside the reference cell C_2 has the same set of neighbor points.

Note that the number of points neighboring a reference point is not pre-defined. Moreover, the set of neighboring points for all points inside a cell is always the same. In a way, we can say that the local bounding boxes enlargement strategy allows the algorithm to adapt to shape changes and non-uniform point distribution (or point density) of the point cloud. For example, in Fig. 4.6(a)-(f), points inside the leaf cells C_1 and C_2 are neighbor points, whereas points inside the leaf cells, C_2 and C_3 cannot be considered as neighbors.

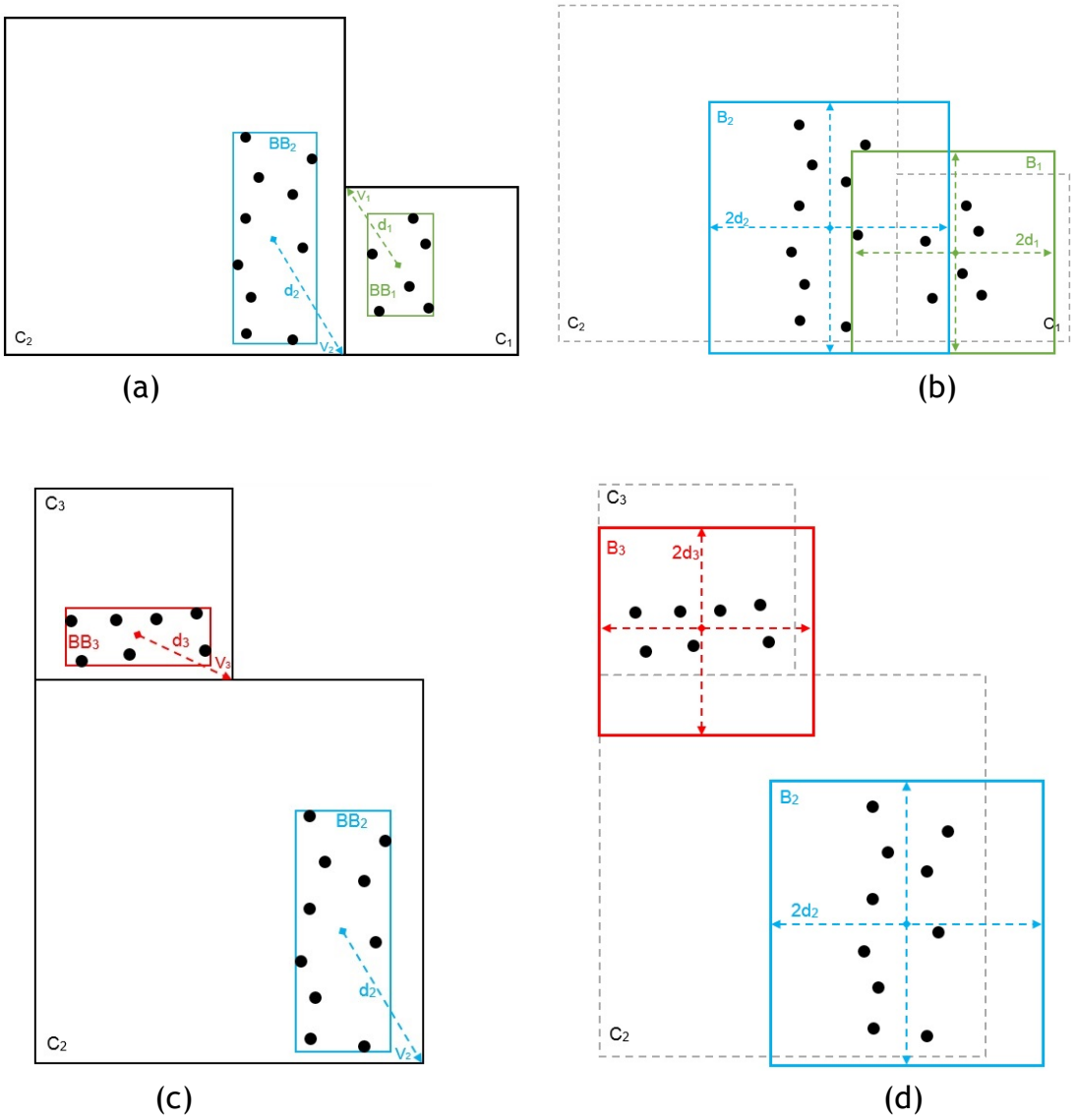
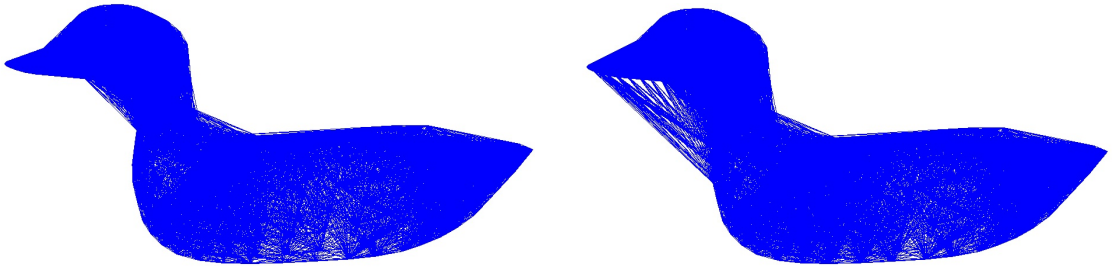


Figure 4.6: The enlargement strategy of local bounding boxes: C_2 is here the reference cell, and C_1 and C_3 are two of its adjacent cells; BB_1, BB_2 and BB_3 are local bounding boxes, while B_1, B_2 and B_3 are the corresponding enlarged bounding boxes; V_1, V_2 and V_3 are the closest cell corners from the bounding box centers; d_1, d_2 and d_3 are the distances from the bounding box centers to the closest cell corners; C_1 and C_2 remain adjacent cells, but not C_2 and C_3.

The effects of using the local bounding boxes enlargement strategy are particularly noticeable for the Duck's point cloud depicted in Fig. 4.7, where straight-line segments connect neighbor points. Fig. 4.7(a) shows the result of applying the local bounding boxes enlargement strategy, and Fig. 4.7(b) exhibits the result of not using the local bounding boxes enlargement strategy. We see that using the local bounding boxes' enlargement strategy results in point neighborhoods that better approximate the surface shape.



(a) (b)

Figure 4.7: Duck's point neighborhoods: (a) applying the local bounding boxes enlargement strategy; (b) not applying the local bounding boxes enlargement strategy.

4.3.3 Building triangle stars

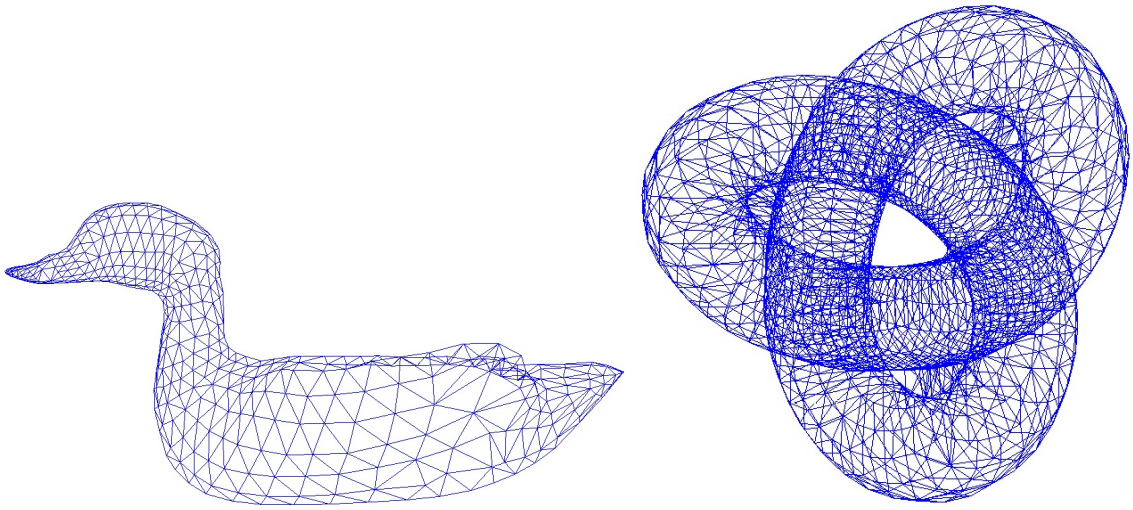
We build up the triangle star of each point from its set of neighbor points. For that purpose, we use the method described in Section 3.3.1 of the previous chapter. Recall that constructing a triangle star around a point (i.e., reference point) is based on two main geometric criteria: proximity and regularity. A point closer to the reference point has a higher probability of being picked up to form a triangle than a distant point from the reference point. We use the regularity criterion to resolve eventual ambiguities in selecting the next neighboring point around the reference point.

Fig. 4.8 shows the union of triangle stars for each of three cloud points. These stars cover each surface almost entirely. This fact is a consequence of building each triangle star, which ensures its manifoldness. However, the manifoldness of each star *per se* does not guarantee the manifoldness of the final mesh. For that purpose, we need to introduce a few rules to distinguish 'good' triangles from 'bad' triangles, discarding then those 'bad' triangles from the triangulation induced by the triangle stars. Such rules are detailed further below.

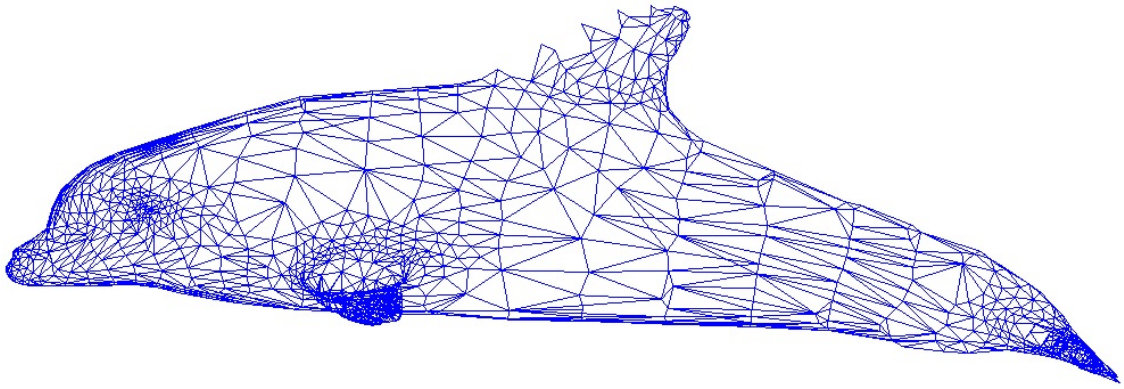
4.3.4 Sorting triangle stars by planarity degree

Before the surface reconstruction step, we need to sort the triangle stars using the planarity degree in increasing order (Section 4.2). That is, one sorts the stars from the most planar to the less planar ones. Thus, one performs the surface reconstruction prioritizing the most planar stars, as illustrated in Fig. 4.9, where we observe that the surface reconstruction of a dolphin is being performed by attaching triangles in the most planar regions first.

Then, for each star, we test each one of its triangles against a few conditions to decide whether one attaches such a triangle to the surface mesh or not. Let us



(a) Duck (b) Loop



(c) Dolphin

Figure 4.8: Combining smooth and manifold stars for all cloud points of three models: (a) Ducks's reconstructed mesh; (b) Loop's reconstructed mesh; (c) Dolphin's reconstructed mesh.

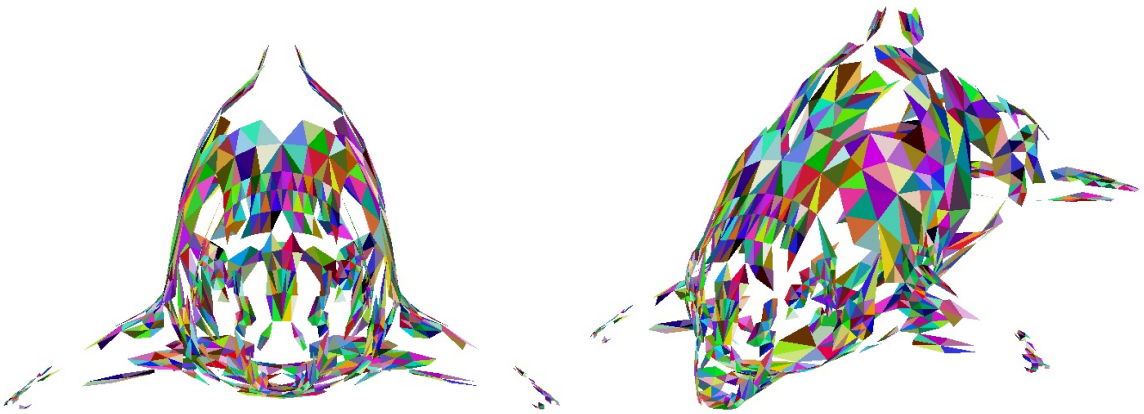


Figure 4.9: Attaching triangles to the mesh representing a dolphin takes place in most planar regions first.

then describe this testing procedure.

4.3.5 Reconstructing the surface by attaching triangles to its mesh

Attaching a triangle to the mesh under construction requires that such a triangle satisfies a few conditions simultaneously. These conditions are the following:

- (i) The triangle must be manifold in the reconstructed mesh (section 4.2);
- (ii) The minimum planarity criterion must be verified (Eq. (4.1));
- (iii) The attachment of the triangle into the mesh must be valid.

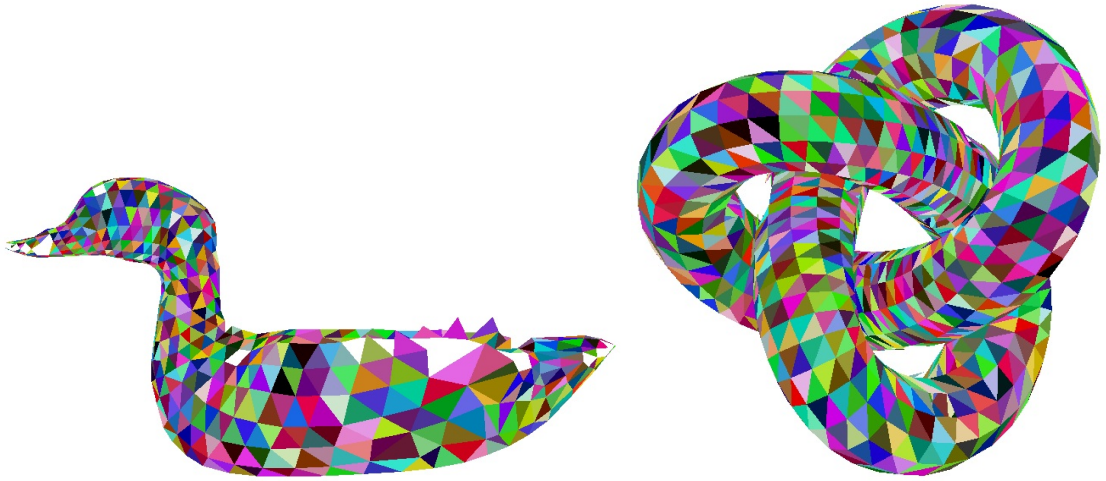
The first condition concerns a manifoldness criterion. It is necessary to bear in mind that there are two moments in the attachment of a triangle to the final mesh; they are the *before* and the *after*. One *a priori* manifold triangle turns into one *a posteriori* manifold triangle. Therefore, we first iterate on the list of triangle stars, only attaching manifold triangles to the mesh. Next, in a second iteration on the list of triangle stars, we attach those *a priori* non-manifold triangles that turn into *a posteriori* manifold triangles. Note that if a triangle is *a priori* non-manifold, it can be even so attached to the mesh since it does cause the appearance of a non-manifold edge after attaching it to the mesh. In other words, a triangle *a priori* classified as non-manifold can be attached to the mesh since it is a manifold triangle in the reconstructed mesh. Thus, we must block any attempt of attaching a non-manifold triangle to an edge with two already attached triangles.

The second condition concerns the minimum planarity conjunction (Eq. (4.1)). The attaching triangle must also satisfy this condition; otherwise, its attachment to the mesh does not occur.

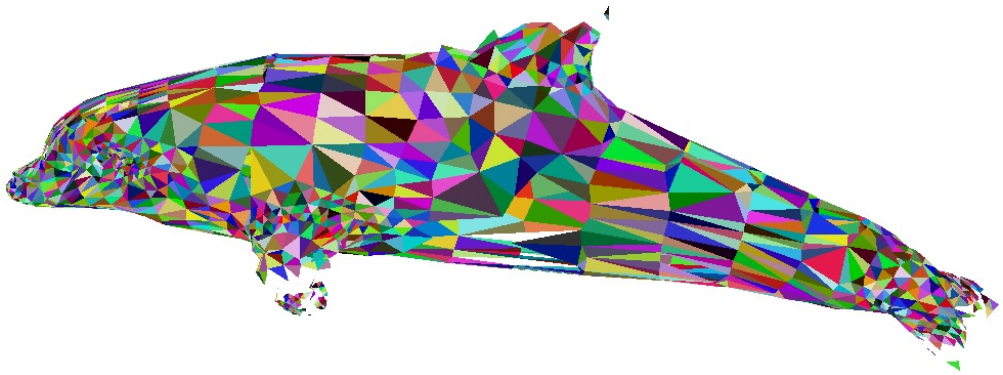
The third condition, say the validity condition, is necessary to check whether the attaching triangle already belongs to the mesh or not. The attaching triangle may already exist in the mesh because any triangle shares three stars, one star per vertex. Thus, if a triangle already exists in the mesh, there is no need to generate an identical triangle and attach it to the mesh.

Summing up, whenever all these three conditions are satisfied simultaneously, one generates a new triangle, which is then attached to the mesh. At the end of this stage, the stars and their triangles were all tested out. However, the reconstructed surface may not be complete yet, as a few holes may exist, as

shown in Fig. 4.10. In this case, we need a final step to fill in the gaps with new triangles.



(a) Duck (b) Loop



(c) Dolphin

Figure 4.10: Reconstructed meshes after every stars and its triangles have been tested. A few holes are left.

4.3.6 Filling in surface holes with triangles

At this point, it may happen that the mesh still has holes (Fig. 4.10). Each mesh hole results from blocking a triangle from being attached to the mesh because, as seen above, the triangle stars cover the entire surface. Thus, each blocked triangle is subject to new scrutiny to make sure its attachment is feasible. Attaching a triangle must satisfy the following three conditions: manifoldness, minimum planarity, and validity. Recall that the final mesh must be manifold; that is, all the *a posteriori* triangle edges must be manifold.

4.4 Experimental Results

4.4.1 Hardware and software setup

We implemented the CTC algorithm in C++ and GLUT 3.7 (Microsoft Visual Studio 2017) on a desktop computer equipped with an Intel processor Core(TM) i3-4005U, 1.73 GHz, 6 GB RAM, running Windows 8.1. Furthermore, we used MeshLab v1.3.3 (`http://www.meshlab.net/`) to evaluate the mesh reconstruction quality of the benchmarking algorithms (including the CTC algorithm) through the Hausdorff distance between original meshes and their corresponding reconstructed meshes.

4.4.2 Benchmarking surface reconstruction methods

In this section, we compare the CTC algorithm with other surface reconstruction algorithms. These latter algorithms are those we used in the previous chapter, including the PCR algorithm. More specifically, we used the Poisson surface reconstruction algorithm due to Kazhdan et al. [KBH06]; the advancing front meshing algorithm due to Cohen-Steiner and Da [CSD04]; the scale-space reconstruction algorithm due to Digne et al. [DMSL11]. The latter two come with CGAL (`http://www.cgal.org`). Furthermore, we used the power-crust meshing algorithm due to Amenta et al. [ACK01a] (`http://web.cs.ucdavis.edu/~amenta/powercrust.html`).

4.4.3 Testing mesh models

Our testing point cloud models were obtained from triangle meshes retrieved from Princeton Shape Benchmark (`http://shape.cs.princeton.edu/benchmark/`), 3D Segmentation Benchmark (`http://segeval.cs.princeton.edu/`), tf3dm (`http://tf3dm.com/`), John Burkardt Home Page (`https://people.sc.fsu.edu/~jburkardt/data` and MeshLab Samples (`https://people.sc.fsu.edu/~jburkardt/data/meshlab/meshlab.html`), some of which are listed in Tables 4.1, 4.2 and 4.3. Such triangle meshes form our ground-truth dataset for benchmarking surface reconstruction methods. Several mesh models are depicted in Figs. 4.12 and 4.13 after reconstruction through CTC algorithm.

4.4.4 Mesh reconstruction quality

The closer the reconstructed mesh is to the original mesh, the higher quality is of such reconstruction. Hausdorff distance measures how far two subsets of the

Table 4.1: Original and reconstructed meshes generated by CTC and PCR algorithms.

Model	Label	Original Mesh		CTC Mesh			PCR Mesh		
		#Vertices	#Triangles	#Vertices	#Triangles	H	#Vertices	#Triangles	H
1	Duck	710	1,416	710	1,414	3.062	710	1,414	4.639
2	Knot	1,280	2,560	1,280	2,560	0.281	1,280	2,560	0.380
3	Loop	1,440	2,880	1,440	2,880	1.441	1,440	2,880	2.691
4	Dolphin	1,867	3,694	1,867	3,724	4.413	1,867	3,734	4.413
5	Doubletorus	4,352	8,708	4,352	8,708	2.597	4,352	8,708	0.000
6	MaxPlanck	7,399	14,749	7,399	14,792	21.584	7,399	14,796	22.446
7	Egea	8,268	16,532	8,268	16,529	1,889	8,268	16,532	1.889
8	Pear	10,754	21,504	10,754	21,504	0.153	10,754	21,504	0.153
9	Fish	14,000	27,996	14,000	27,991	0,462	14,000	28,005	3.624
10	Horse	20,000	39,996	20,000	39,994	3.567	20,000	40,004	2.721
11	Armadillo	26,002	52,000	26,002	51,987	1.164	26,002	52.013	1.936
12	Bunny	34,834	69,451	34,834	69,657	15.240	34,834	69,663	16.010

Abbreviations:
H: Hausdorff distance

Table 4.2: Reconstructed meshes generated by advancing-front and power-crust algorithms.

Model	Label	Advancing Front Mesh			PowerCrust Mesh		
		#Vertices	#Triangles	H	#Vertices	#Triangles	H
1	Duck	710	1,416	3.098	6,642	13,279	7.705
2	Knot	1,280	2,560	0.281	7,744	15,492	23.791
3	Loop	1,440	2,874	7.502	14,368	28,613	14.869
4	Dolphin	1,867	3,658	23.400	16,009	32,006	3.614
5	Doubletorus	4,352	8,708	3.612	35,064	70,132	10.997
6	MaxPlanck	7,399	14,794	22.261	66,282	132,560	22.779
7	Egea	8,268	16,531	1.889	71,882	155,970	2.065
8	Pear	10,754	21,504	0.127	104,840	209,675	1.068
9	Fish	14,000	27,998	5.064	–	–	–
10	Horse	20,000	39,994	2.591	209,879	419,744	3.641
11	Armadillo	26,002	51,995	1.467	245,683	491,364	2.418
12	Bunny	34,834	69,656	16.253	349,013	698,011	17.382

Abbreviations:
H: Hausdorff distance

metrics space are from each other. Thus, we used the Hausdorff distance between two surface meshes as our mesh reconstruction quality metric. Hausdorff distance is defined as follows:

$$H(S, S') = \max\Big(\max_{\mathbf{x}_i \in S} \delta(\mathbf{x}_i, S'), \max_{\mathbf{x}_j \in S'} \delta(\mathbf{x}_j, S)\Big) \quad (4.2)$$

Table 4.3: Reconstructed meshes generated by scale-space and Poisson algorithms.

Model	Label	Scale-Space Mesh			Poisson Mesh		
		#Vertices	#Triangles	H	#Vertices	#Triangles	H
1	Duck	710	1,988	27.355	1,489	2,974	27.746
2	Knot	1,280	4,356	29.430	2,490	5,004	34.903
3	Loop	1,440	4,498	32.077	2,223	4,454	38.958
4	Dolphin	1,867	5,436	25.066	3,281	6,562	24.008
5	Doubletorus	4,352	16,752	18.413	3,176	6,356	14.921
6	MaxPlanck	7,399	28,732	8.932	11,108	22,212	22.515
7	Egea	8,268	32,818	6.827	3,568	7,132	5.778
8	Pear	10,754	37,840	25.895	2,994	5,984	28.969
9	Fish	14,000	27,998	5.064	12,030	24,080	21.299
10	Horse	20,000	78,936	7.549	10,264	20,524	9.200
11	Armadillo	26,002	99,346	4.908	16,032	32,060	8.432
12	Bunny	34,834	138,050	4.470	12,927	25,850	16.998

Abbreviations:
H: Hausdorff distance

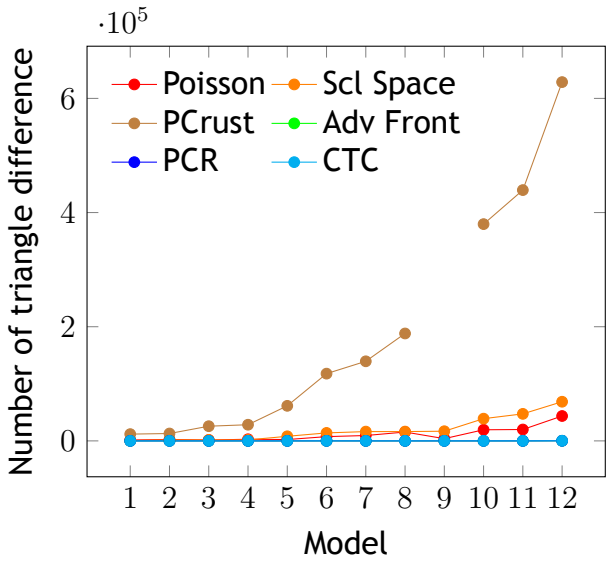


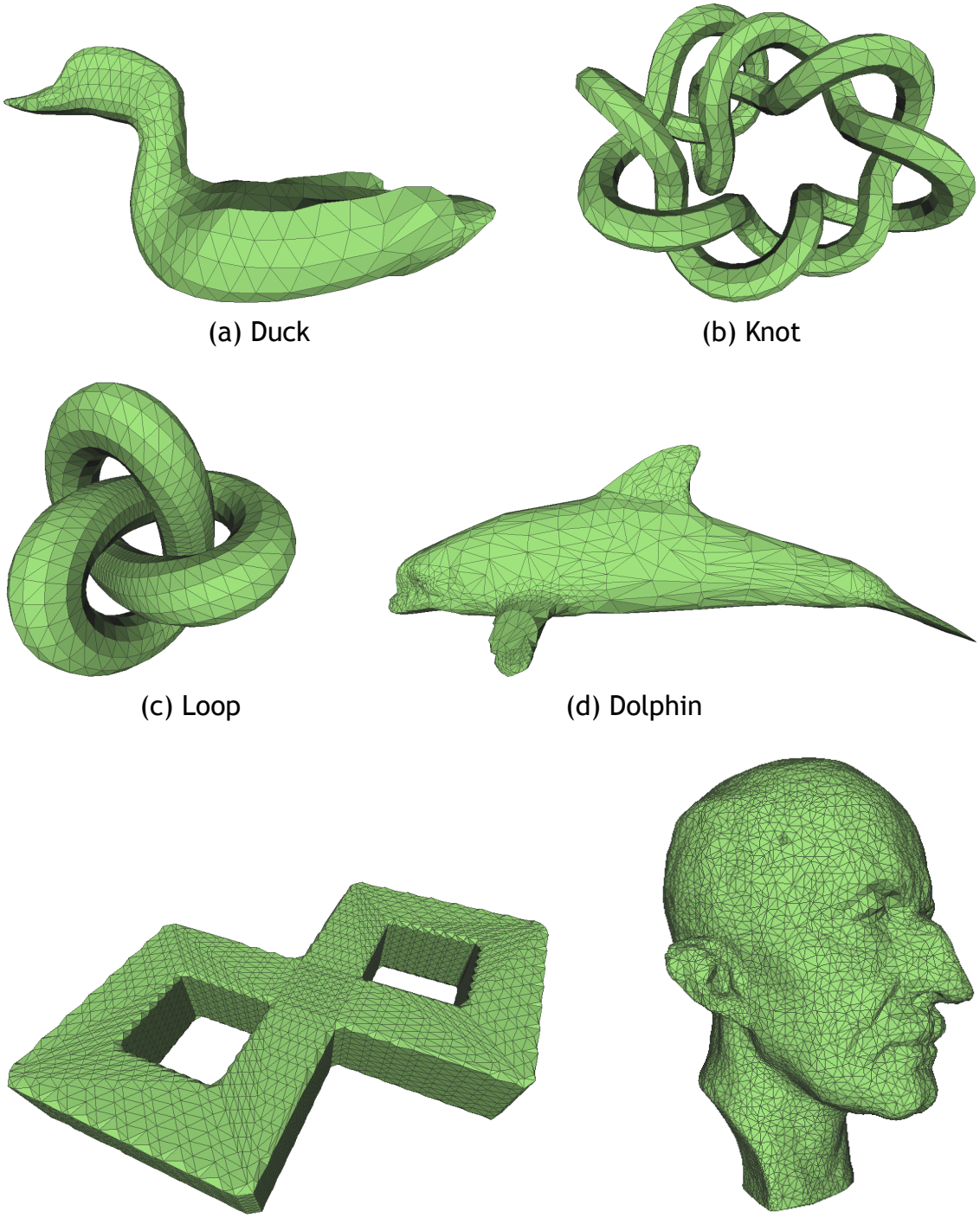
Figure 4.11: Differences in the number of triangles relative to original meshes.

where S, S' denote the two surface meshes, the ground-truth mesh and testing mesh, with

$$\delta(\mathbf{x}_i, S') = \min_{\mathbf{x}_j \in S'} d(\mathbf{x}_i, \mathbf{x}_j) \quad \text{and} \quad \delta(\mathbf{x}_j, S) = \min_{\mathbf{x}_i \in S} d(\mathbf{x}_j, \mathbf{x}_i) \quad (4.3)$$

where $d(\mathbf{x}, \mathbf{x}')$ is the Euclidean distance between points $\mathbf{x}$ and $\mathbf{x}'$.

During the tests of the CTC algorithm, the MeshLab was used to compute the Hausdorff distance. According to Eq. (4.2), Hausdorff distance applies to sets



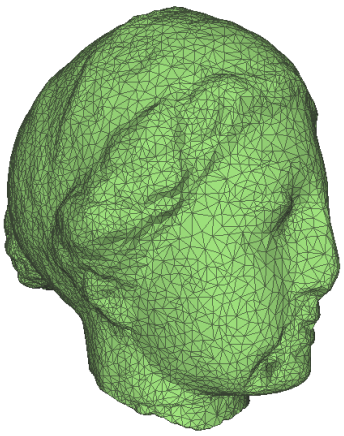
(a) Duck (b) Knot

(c) Loop (d) Dolphin

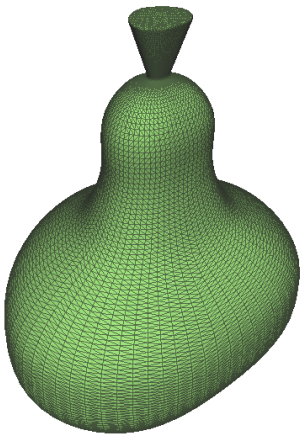
(e) Doubletorus (f) MaxPlanck

Figure 4.12: Some reconstructed surface meshes using the CTC algorithm.

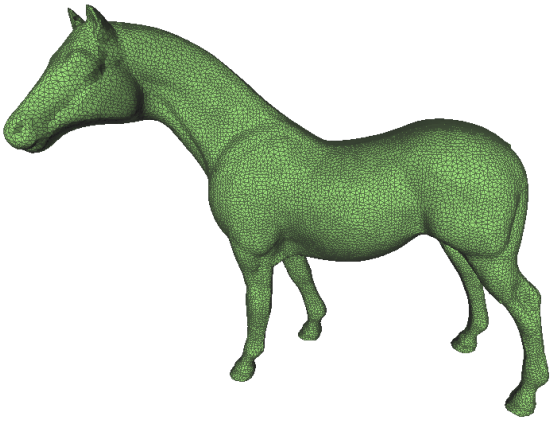
of points. Therefore, the simplest way to calculate the distance between two meshes is computing the Hausdorff distance considering only vertices. However, such a distance would be zero because the vertices of CTC, PCR, Advancing front, and scale-space meshes are the same as those of the original mesh. We also considered barycenters of edges and triangles in computing the Hausdorff distance between meshes to overcome this problem.

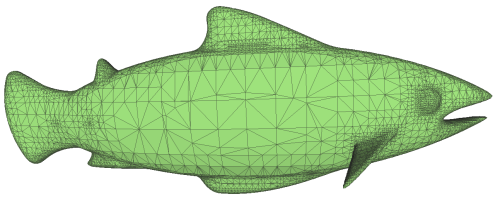


(g) Egea



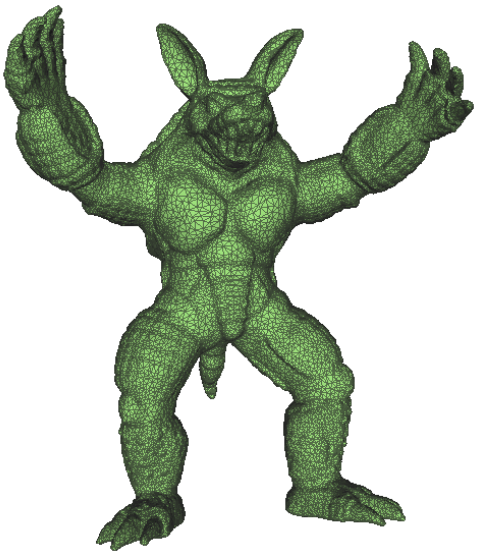
(h) Pear

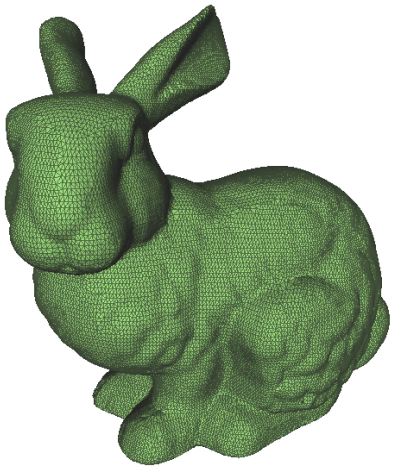




(i) Fish

(j) Horse





(k) Armadillo

(l) Bunny

Figure 4.13: More reconstructed surface meshes using the CTC algorithm.

Looking at Tables 4.1, 4.2 and 4.3, we observe the following:

- *Number of vertices*. The number of vertices remains unchanged for CTC, PCR, advancing front, and scale-space algorithms, but not for Power crust

and Poisson algorithms. Power crust meshing adds much more points into the point cloud to get a uniform density and increase the density of points (cf. [ACK01a]). Poisson algorithm produces a mesh of an implicit surface that approximates the point cloud. Consequently, the number of vertices (and their locations) varies relative to the original ones.

- *Number of triangles*. The number of triangles tends to remain unchanged for interpolated methods like CTC, PCR, and advancing-front methods (see Table 4.1 and 4.2). This principle does not apply to the power-crust algorithm because it adds many more points to the point cloud, producing a mesh with much more triangles. On the other hand, Poisson and scale-space reconstruction algorithms generate implicit surfaces that approximate the input point clouds; such surfaces get triangulated *a posteriori*. In other words, implicit methods produce triangles whose number does not necessarily match those of the original mesh, as shown in Table 4.3. Also, looking at Fig. 4.11, we note the number of triangles of the original meshes and corresponding reconstructed meshes is not the same, though the number of triangles differs less in the CTC, PCR Cocktail, and advancing-front algorithms.

- *Hausdorff distance*. Recall that the Hausdorff distance is the maximum of the minimum distance between two sets of points (see Eq. (4.2)). As shown in Tables 4.1, 4.2 and 4.3, the Hausdorff distance between the original meshes and the meshes generated by the CTC or PCR algorithms are, in general, smaller than for the benchmarking surface reconstruction methods (cf. Fig. 4.14). CTC algorithm presents a smaller Hausdorff distance for models 1, 3, and 9 than the PCR algorithm, whereas the PCR algorithm presents a shorter Hausdorff distance for model 5 than the CTC algorithm. Note that the two worst surface mesh reconstructions using CTC are those concerning model 6 (Max Planck) and model 12 (Bunny). These results are so because their original meshes are not closed meshes underneath. So, it is necessary to attach new triangles to the mesh to fill its gaps. Therefore, new edges and facets increase the Hausdorff distance between the original mesh and the reconstructed mesh. Even so, CTC or PCR algorithms produce more reliable or high-quality meshes than their competitors. The meshes generated by the CTC algorithm can be observed in Figs. 4.12 and 4.13, while meshes generated by the PCR algorithm are shown in Figs. 3.9 and 3.10.

A glance at Fig. 4.14 allows us to observe that the interpolated methods (CTC,

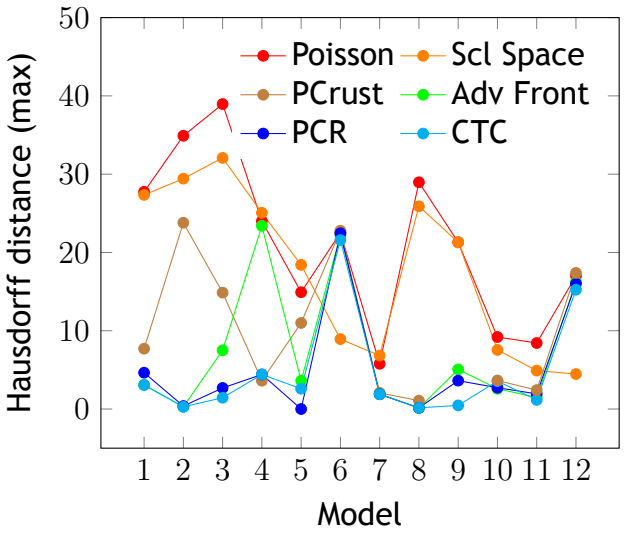


Figure 4.14: Hausdorff distance for reconstructed meshes, comparing CTC, PCR, power-crust, advancing-front, scale-space and Poisson algorithms.

PCR, advancing-front, and power-crust algorithms) generate meshes closer to the original meshes than methods that approximate the input point cloud (Poisson and Scale-space algorithms). This fact explains by the interpolation approach of simplicial methods rather than the approximating approach of implicit methods. Also, the rounding of sharp features results from the blending of implicit methods' kernel functions.

To consolidate the ranking of these algorithms, we carried out a more refined analysis of results in terms of other Hausdorff metrics as follows:

- *Hausdorff distance's mean*. This measure represents the mean distance between two sets of points, that is,

$$d_M = \frac{1}{n+m}\Big(\sum_{i=1}^{n}\delta(\mathbf{x}_i, S') + \sum_{j=1}^{m}\delta(\mathbf{x}_j, S)\Big) \quad (4.4)$$

  where $\delta(\mathbf{x}_i, S')$ is the minimum distance from the point $\mathbf{x}_i \in S$ to the point set S' and $\delta(\mathbf{x}_j, S)$ is the minimum distance from the point $\mathbf{x}_j \in S'$ to the point set S, as given by Eq. (4.3).

  Fig. 4.15 shows that interpolated algorithms outperform approximating algorithms in terms of Hausdorff mean distance. Besides, CTC, PCR, and advancing-front algorithms rank first and are indistinguishable concerning Hausdorff's mean distance.

- *Hausdorff distance's root mean square* (RMS). This metric measures the

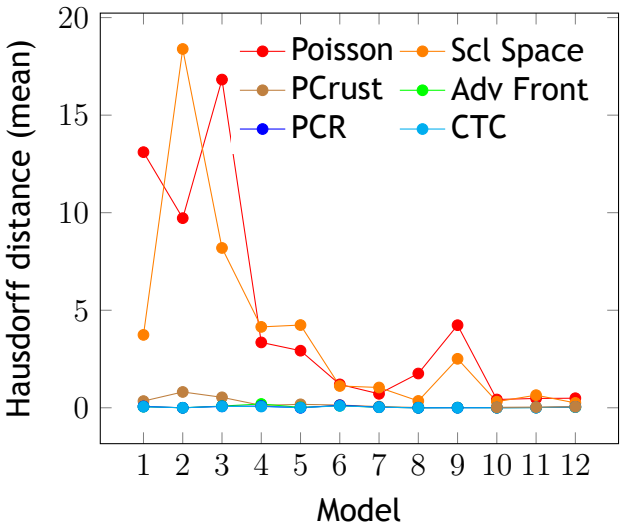


Figure 4.15: Hausdorff distance's mean for reconstructed meshes, comparing CTC, PCR, power-crust, advancing-front, scale-space, and Poisson algorithms.

arithmetic mean of the squares of the set of minimum distances between two point sets S and S', that is,

$$d_{RMS} = \sqrt{\frac{1}{n+m}\Big(\sum_{i=1}^{n}(\delta(\mathbf{x}_i, S'))^2 + \sum_{j=1}^{m}(\delta(\mathbf{x}_j, S))^2\Big)}. \quad (4.5)$$

As shown in Fig. 4.16, we see that CTC and PCR rank first and produce triangulations more similar to the original meshes than the Advancing Front or Power crust algorithms, which agrees with the results shown in Fig. 4.14.

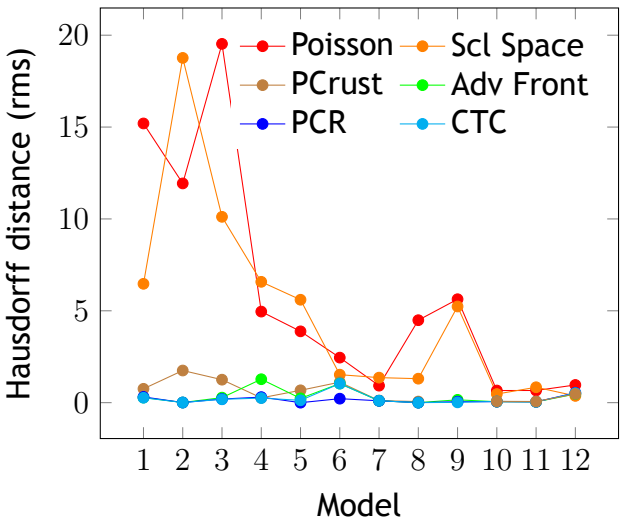


Figure 4.16: Hausdorff distance's root mean square for reconstructed meshes, comparing CTC, PCR, power-crust, advancing-front, scale-space, and Poisson algorithms.

- *Hausdorff distance's standard deviation*. This metric allows us to assess the dispersion of the minimum distances relative to Hausdorff's mean distance (see Eq. (4.4)), that is,

$$d_{SD} = \sqrt{\frac{1}{n+m}\Big(\sum_{i=1}^{n}(\delta(\mathbf{x}_i, S') - d_M)^2 + \sum_{j=1}^{m}(\delta(\mathbf{x}_j, S) - d_M)^2\Big)}. \quad (4.6)$$

  As shown in Fig. 4.17, CTC and PCR have the lower dispersion of the distances for all models.

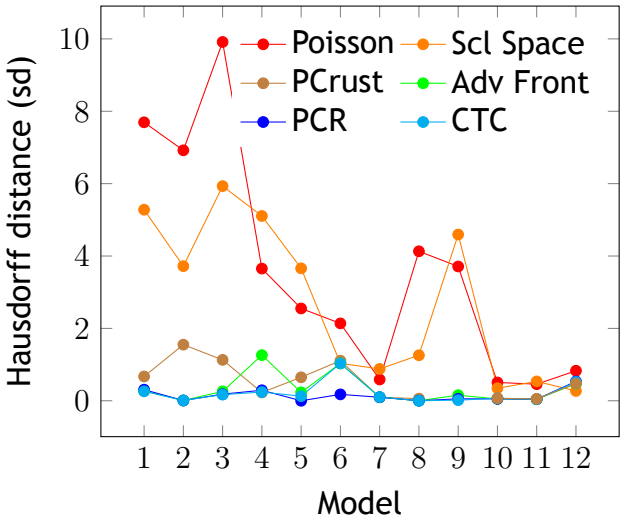


Figure 4.17: Hausdorff distance's standard deviation for reconstructed meshes, comparing CTC, PCR, power-crust, advancing-front, scale-space, and Poisson algorithms.

CTC and PCR algorithms produce the most similar triangulations to the original meshes. In general, interpolated methods generate triangulations that tend to be identical to the original meshes because of their interpolating approach. On the other hand, the approximating approach behind implicit methods generates triangulations that are further away from the original meshes, mainly when the implicit surface results from local functions' blending.

4.4.5 Surface reconstruction times

We have also benchmarked algorithms in terms of time performance. In Table 4.4, and also in Fig. 4.18, we present the mesh reconstruction times taken by each algorithm. In addition to the reconstructed surfaces' quality, we note CTC, PCR, and advancing-front algorithms are the fastest in the mesh reconstruction process. As we observed in Tables 4.1 and 4.2, the power-crust algorithm uses a linear approximation method to increase the density of points of the point

cloud, so we end up adding new vertices to the mesh. This fact explains the slow down of the reconstruction times for this algorithm. The Poisson algorithm is also expectedly slower than the CTC, PCR, and Advancing Front algorithms because of its implicit approach, particularly for larger point clouds.

Table 4.4: Time for surface reconstruction by every algorithms.

Model	Label	Original Mesh		Time (seconds)					
		#Vertices	#Triangles	CTC	PCR	Adv Front	Scl Space	PCrust	Poisson
1	Duck	710	1,416	0.013	0.024	0.051	0.045	1.259	0.863
2	Knot	1,280	2,560	0.021	0.041	0.067	0.072	2.172	1.358
3	Loop	1,440	2,880	0.025	0.049	0.086	0.093	3.200	1.285
4	Dolphin	1,867	3,694	0.049	0.075	0.124	0.134	2.809	2.379
5	Doubletorus	4,352	8,708	0.136	0.175	0.297	0.372	10.955	2.627
6	MaxPlanck	7,399	14,749	0.217	0.334	0.441	0.504	11.050	3.619
7	Egea	8,268	16,532	0.268	0.368	0.520	0.595	18.554	3.863
8	Pear	10,754	21,504	0.271	0.372	0.663	1.086	41.161	3.663
9	Fish	14,000	27,996	0.806	0.658	0.976	1.044	–	13.848
10	Horse	20,000	39,996	0.823	1.176	1.328	1.605	31.500	9.435
11	Armadillo	26,002	52,000	1.596	1.423	1.733	2.018	45.957	15.115
12	Bunny	34,834	69,451	2.297	2.180	2.240	2.635	71.754	14.774

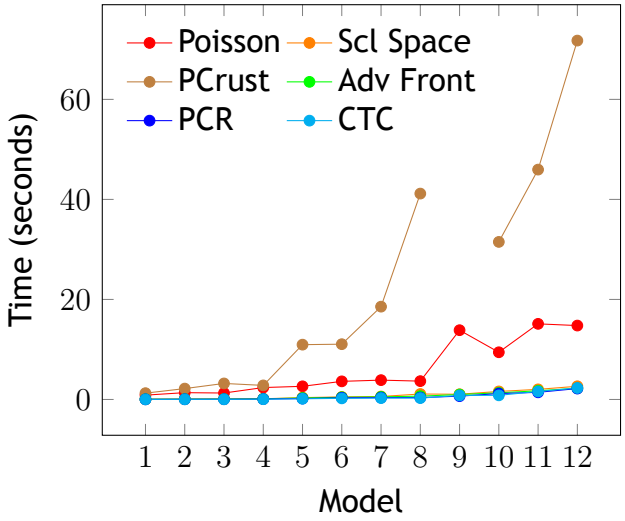


Figure 4.18: Times for reconstructed meshes using CTC, PCR, power-crust, advancing-front, scale-space, and Poisson algorithms.

4.5 Discussion

Our experiments have shown that surface reconstruction algorithms work well for uniform or high-density point clouds. On the contrary, non-uniform or low-density point sampling is the source for a few problems found in surface reconstruction algorithms such as mesh drifting, holes, incomplete meshing, and trimming and rounding of sharp features.

We also observed that the implicit algorithms are more sensitive to the sampling non-uniformity than simplicial methods. As shown in Figs. 4.19(c)-(d), implicit methods may misbehave when the input point cloud does not possess a uniform distribution or high point density, as it results in incomplete or weird triangulations. Looking at Figs. 4.19(a)-(b), we observe that the simplicial methods are more robust than implicit methods because the meshing tends to be complete and less prone to unexpected triangulations. However, we noted that the power-crust algorithm could not reconstruct the fish model (Table 4.2) at all, possibly because this algorithm imposes a predefined minimum distance between points.

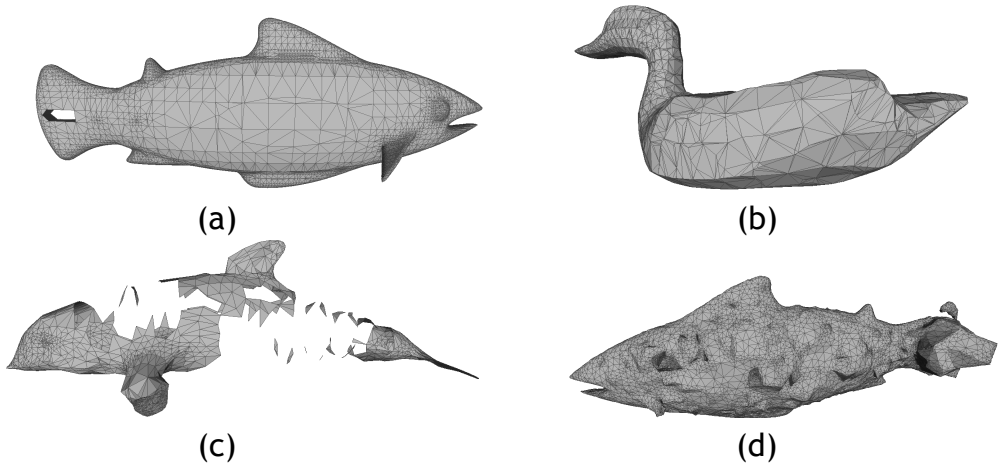


Figure 4.19: Non-uniform point sampling issues: (a) Fish's meshing using advancing-front algorithm originates a hole through the mesh; (b) Duck's meshing using power crust algorithm increases its volume as a result of more points added to the initial point cloud; (c) incomplete Dolphin's meshing using scale-space algorithm; and (d) weird Fish's meshing using Poisson algorithm.

CTC algorithm successfully deals with sampling non-uniformity mainly for two reasons. First, the octree subdivision following the use of local bounding boxes enlargement strategy, as detailed in subsection 4.3.2, it allows for every cloud point a more suitable neighborhood taking into account the point distribution (see Fig. 4.7). Secondly, the triangle stars all together ensure connectivity from a point to those surrounding it, as shown in Fig. 4.8. Note that the CTC algorithm does not pre-determine the maximum number of closest points or pre-determine the maximum radius enclosing a given point's neighborhood. The most suitable values vary from one point cloud to another, particularly in a non-uniform point distribution.

Looking at the Hausdorff distance results shown in Fig. 4.14, we see the worst surface mesh reconstructions using CTC are those concerning Max Planck and Bunny models (see Tables 4.1-4.3 and Figs. 4.12(f) and 4.13(l)). This fact is so

because original meshes are not closed meshes underneath, so new triangles must be attached to the mesh to close any mesh border to the outside. In other words, new edges and facets increase the Hausdorff distance between the original mesh and the reconstructed mesh.

4.5.1 Mesh drifting

As can be observed in Fig. 4.20, four reconstructed surfaces suffer from the mesh drifting problem. This problem occurs when non-adjacent regions, passing to a short distance from each other, are wrongly triangulated together, making shortcuts or separate parts. In Fig. 4.20(a), the advancing-front algorithm made a hole due to an internal shortcut at the left pectoral fin. Besides, a separate part appeared at the tail fin. Looking at Fig. 4.20(b), we see that the Power Crust algorithm built an external shortcut that is not part of the knot surface.

As suggested by the triangulations shown in Figs. 4.20(c) and (d), implicit methods have more tendency to the drifting phenomena than simplicial methods. The mesh drifting effects of implicit methods can be explained by the inflating caused by the blending functions used in the approximation to the points of the input cloud, making non-adjacent nearby regions even closer to each other. This fact is in line with the Hausdorff distance results presented in the previous section.

In the CTC algorithm, we combine the priority given to triangle stars (sorting them by smoothness degree) and the minimum planarity conjunction (Eq. 4.1) applied to candidate triangles. This combination aims to select which triangles will be attached to the mesh, preventing at the same time mesh drifting effects during the reconstruction process, as shown in Figs. 4.12 and 4.13.

4.5.2 Trimming and rounding of sharp features

As illustrated in Fig. 4.21(c)-(d), implicit algorithms tend to round sharp features, whereas simplicial algorithms tend to trim sharp features, Figs. 4.21(a)-(b). Regarding the CTC algorithm (see Figs. 4.12 and 4.13), and considering the resulting triangulations for the same input point sets, we see that there are no rounding or trimming effects. This fact is so because the CTC triangulation gives priority to more planar regions. Finally, there are no rounding effects because CTC is an interpolating method; thus, we do not add new mesh points.

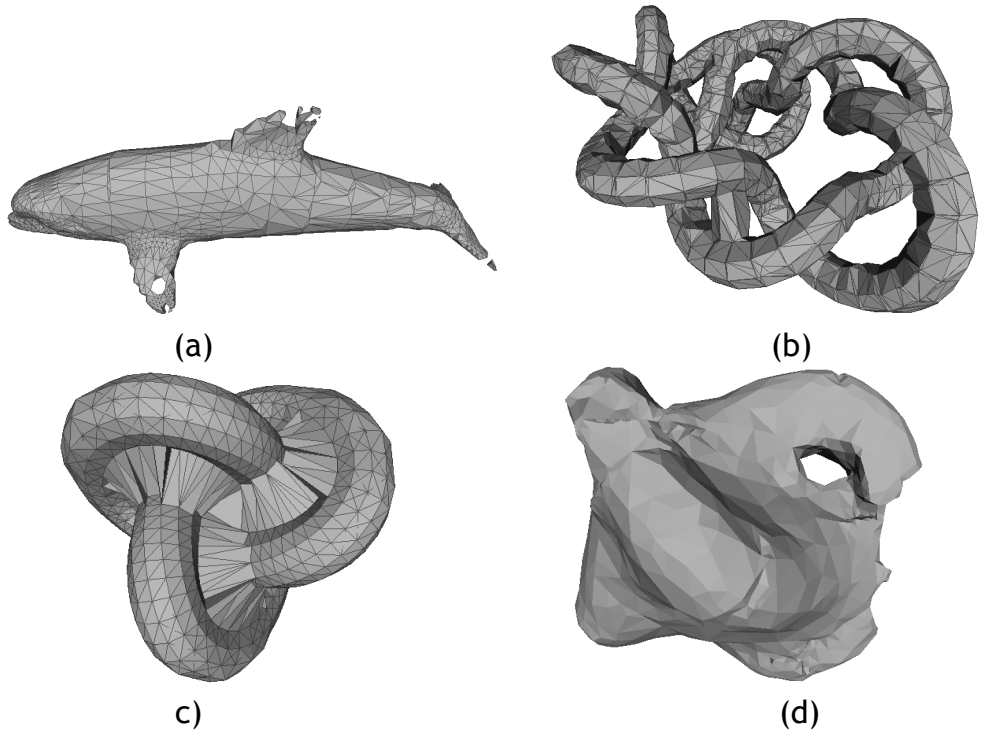


Figure 4.20: Mesh drifting issues: (a) Dolphin's meshing using advancing front algorithm; (b) Knot's meshing using power crust algorithm; (c) Loop's meshing using scale space algorithm; and (d) Loop's meshing using Poisson algorithm.

4.5.3 Dihedral angle and internal angles bounds

The problem of imposing angle bounds in mesh growing algorithms is generalized. These algorithms pre-determine an internal angle minimum and maximum for each new triangle attached to the mesh front. Moreover, such algorithms also pre-determine a maximum dihedral angle between the new triangle and its adjacent triangle in the mesh front. In the CTC algorithm, this problem does not exist due to the regularity function Eq. (3.2) that is applied to build the stars of triangles. Thus, there is no need to impose any bounds on the internal angles of triangles. The CTC algorithm does not set bounds to any dihedral angle between adjacent triangles because one carries out the triangulation, prioritizing smoother stars. Then, one sorts such stars by smoothness degree and applying the smoothness conjunction Eq. (4.1). Hence, the triangulation of smoother regions comes first.

4.6 Summary

The CTC algorithm is a surface mesh reconstruction algorithm that belongs to the family of interpolation methods. We do not use normal vectors in this algorithm. It builds upon triangle stars that cover the whole surface, providing *a priori* a set

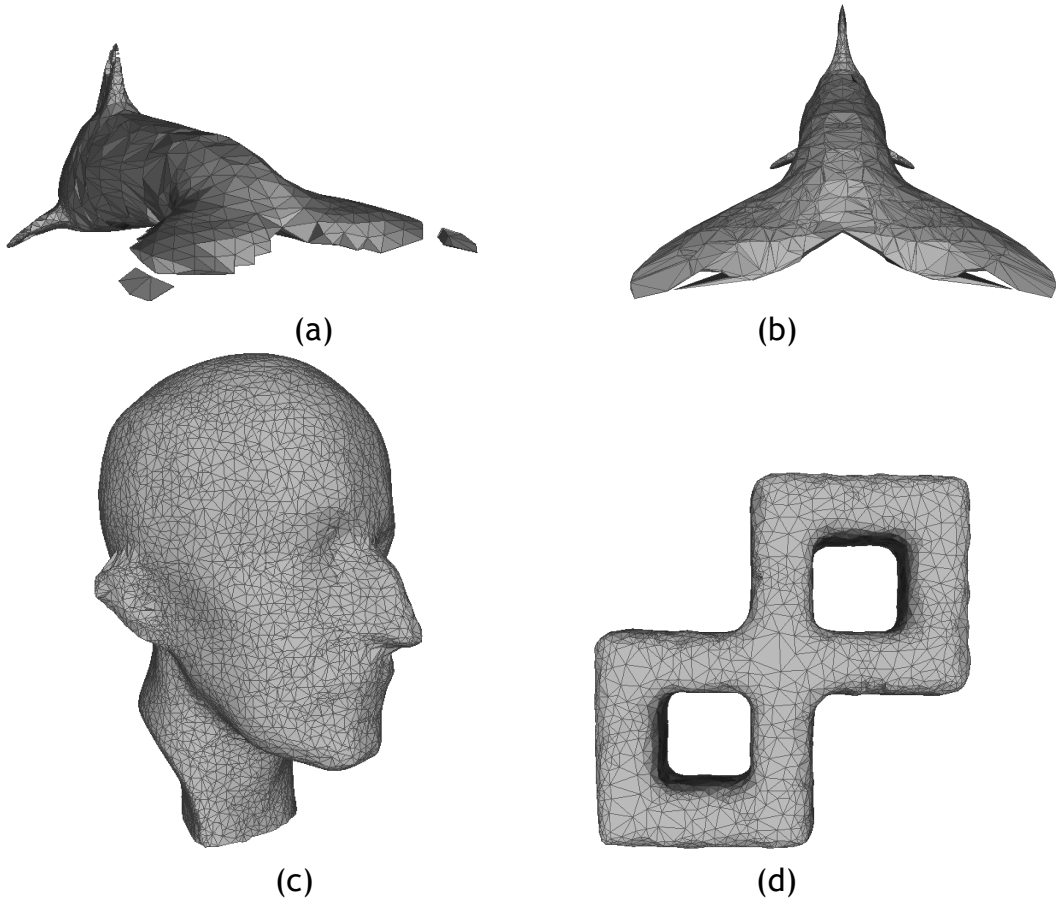


Figure 4.21: Sharp features issues: (a) Dolphin's meshing using advancing front algorithm originates a total trimming of its fluke; (b) Dolphin's meshing using power crust algorithm originates a partial trimming of its fluke; (c) Max Planck's meshing using scale space algorithm; and (d) Double Torus's meshing using Poisson algorithm.

of candidate triangles; most of them will belong to the reconstructed surface. The more planar stars are processed first, as well as their triangles. Under some rules related to manifoldness and planarity, one tests those triangles to decide which ones will be attached to the final mesh.

As a consequence of applying a planarity criterion, the triangulation gives priority to low curvature regions or even planar regions, as can be observed in Fig. 4.9. Therefore, the triangulation of sharp features comes after low curvature regions, avoiding eventual mesh drifting effects. The CTC algorithm copes with non-uniform point sampling and does not impose dihedral angle bounds between adjacent triangles. Moreover, the regularity function used to construct the triangle stars makes the CTC algorithm generate a quasi-regular triangle mesh.

Finally, the enlarged local bounding box strategy applied to each leaf octree

cell's points makes it unnecessary to set a maximum radius or a minimum number of neighbor points around each point. That is, the CTC algorithm copes with an eventual, non-uniform distribution of cloud points.

Chapter 5

Surface Reconstruction: The LIS Algorithm

5.1 Introduction

We here introduce a new reconstruction surface algorithm, called the Linear Implicit Surface (LIS) algorithm. Unlike most implicit surface reconstruction algorithms —see, for example, Ohtake et al.' s algorithm [OBA+03], which builds upon multilevel partition-of-unit (MPUs)—, the LIS algorithm does not make use of an approximation approach. LIS generates an implicit surface that interpolates the input cloud points.

The MPU-based algorithm, due to Othake et al., uses a local quadratic function that fits the data points inside each subdomain (i.e., a leaf cell of an octree). The global fitting function is the result of combining local functions weighted by partition functions. The weighting functions limit the influence range of each local function into the resulting global function. Each weighting function is centered at the subdomain's barycenter (i.e., the center of the octree cell) and has spherical support of radius that overlaps the subdomain boundaries. Therefore, the weighting functions influence the local functions in the resulting global function. However, the MPU-based algorithm does not consider how one distributes the points inside each subdomain; in other words, one does not consider the surface shape inside each subdomain. That is, the barycenter of a subdomain may not represent the point subset inside such a subdomain.

On the contrary, the LIS algorithm considers a local linear function (and a weighting function) centered at each cloud point. Each point is the center of a subdomain defined by a manifold chart (whose triangulation would be a triangle star, as seen in Section 3.3.1). The normal vector at each point represents each local linear function and, consequently, the tangent plane at each point. Moreover, the weighting function at each point has support with radius given by its star's mean radius. The global function representing the final reconstructed surface stems from the weighted blending of local linear functions, one function per point.

5.2 Background

We assume that the point cloud includes the input points and their normal vectors to the surface. So, we can define a local linear implicit function in $\mathbb{R}^3$ at each point taking in consideration the its normal vector. Thus, considering the point $\mathbf{p}_i = (x_i, y_i, z_i)$ with normal vector $\mathbf{n} = (A, B, C)$, the local linear implicit function represents the tangent plane at such a point, and is given by the following expression:

$$f_i(x, y, z) = Ax_i + By_i + Cz_i + D = 0 \quad (5.1)$$

where the coefficient D is given by,

$$D = -(Ax_i + By_i + Cz_i) \quad (5.2)$$

Weighting functions determine the influence of those local functions in the global function. A weighting function's support determines how much impact its corresponding local function has on the global function. Hence, the *mean radius of the star* around each point determines the radius of the spherical support of its weighting function (see Fig. 5.1).

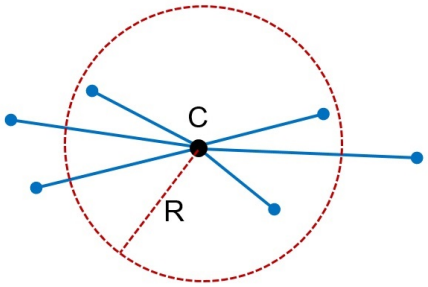


Figure 5.1: The center $\mathbf{c}$ and the mean radius R of a point star.

Therefore, given the set of k points $\mathbf{p}_j$ ($j = 1, \cdots, k$) that represents the star around a point $\mathbf{c}$, the mean radius R of that star is given by the following expression:

$$R = \sum_{j=1}^{k} \frac{\| \mathbf{p}_j - \mathbf{c} \|}{k} \quad (5.3)$$

5.3 The LIS Algorithm

Given an input point cloud with normal vectors, the LIS algorithm consists of the following main steps:

1. Construct manifold stars;
2. Determine local functions;
3. Determine weighting functions;
4. Determine global function;
5. Make marching cubes triangulation;

These steps are detailed in the next subsections.

5.3.1 Building manifold stars

Recall that each star defines a subdomain whose center is a cloud point. Therefore, the first step of this algorithm corresponds to perform the same three initial steps of the CTC algorithm (see Sections 4.3.1, 4.3.2 and 4.3.3). They are: (i) make point cloud octree subdivision; (ii) find neighbor points for each point (using the enlargement strategy for local bounding boxes, one per leaf octree cell); and (iii) finally construct the triangle stars. In practice, a sequence of points around a cloud point represents its triangle star.

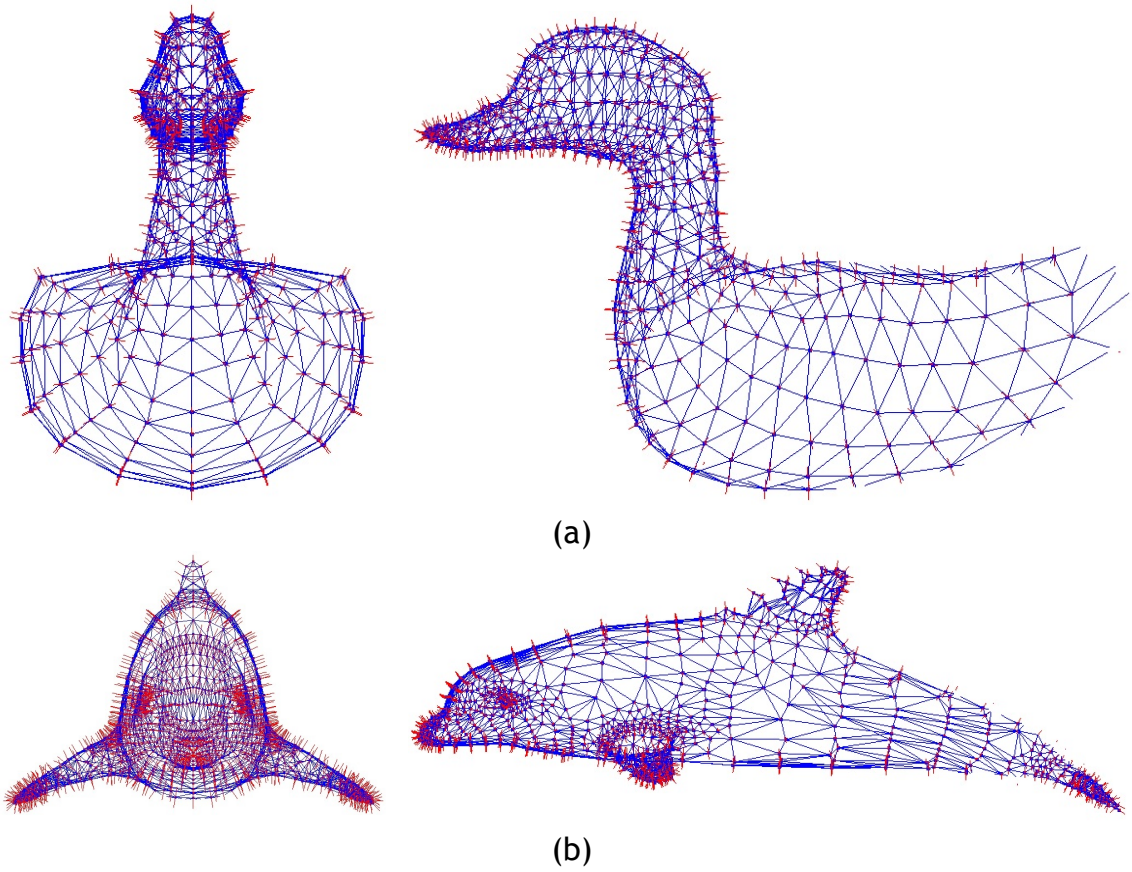
5.3.2 The local functions

The set of normal vectors associated with the point cloud, one per sample point, defines the tangent space on the surface. Figure 5.2 shows outward normal vectors for Duck and Dolphin point clouds.

Let us assume that a point cloud consists of N sample points. As mentioned in Section 5.2, the local functions represent tangent planes at the cloud points. That is, given the i-th sample point $\mathbf{p}_i = (x_i, y_i, z_i)$ with normal vector $\mathbf{n}_i = (A_i, B_i, C_i)$, its local linear function is given by

$$f_i(\mathbf{p}_i) = f_i(x_i, y_i, z_i) = A_i x_i + B_i y_i + C_i z_i + D_i \quad (5.4)$$

with $i = 1, \cdots, N$.



(a)

(b)

Figure 5.2: Normal vectors at sample points for Duck (a) and Dolphin (b).

5.3.3 The weighting functions

The weighting function limits the influence of each local function in the global function. The weighting function is a nonnegative function centered at each point, being its support determined by the mean radius of the point's star. Besides, we need to normalize the weighting function and make it decay quadratically to zero. Thus, the weighting function varies inversely to the mean star radius raised to 2.

So, considering the star of the i-th sample point $\mathbf{p}_i = (x_i, y_i, z_i)$ has mean radius R_i, the weighting function $W_i(\mathbf{p})$ is given by

$$W_i(\mathbf{p}) = \begin{cases} \frac{{R_i}^2 - \|\mathbf{p}-\mathbf{p}_i\|^2}{{R_i}^2}, & \| \mathbf{p} - \mathbf{p}_i \| \leq R_i \\ \\ 0, & \| \mathbf{p} - \mathbf{p}_i \| > R_i \end{cases} \tag{5.5}$$

where $\mathbf{p} = (x, y, z)$ is an arbitrary point, and $i = 1, \cdots, N$, with N denoting the total number of cloud points.

5.3.4 The global function

The resulting global function $F(\mathbf{p})$ is weighted blending of local functions $f(\mathbf{p}_i)$, with $i = 1, \cdots, N$, though it is normalized by the weighting functions as follows:

$$F(\mathbf{p}) = \frac{\sum_{i=1}^{N} W_i(\mathbf{p}) f_i(\mathbf{p})}{\sum_{i=1}^{N} W_i(\mathbf{p})} \quad (5.6)$$

where $\mathbf{p} = (x, y, z)$ is an arbitrary point, and N denotes the total number of cloud points.

This global function built from linear local shape functions represents the reconstructed surface, here called *linear implicit surface*. Its linearity comes from the discrete tangent space induced by the point cloud. To triangulate the linear implicit surface, we can use any triangulation algorithm commonly found in the literature for implicit surfaces, including the marching cubes (MC) algorithm due to Lorensen and Cline [LC87].

5.3.5 Marching cubes triangulation

We used a marching cubes algorithm to triangulate the reconstructed surface through the LIS algorithm ([NY06]). An essential issue in implementing this MC algorithm is the size of each cube, also called step size. The cube size must vary from one point cloud to another; otherwise, the implicit surface triangulation may present holes. This issue is critical in the presence of non-uniform point sampling. Furthermore, it is necessary to bear in mind that the surface results from blending several local functions, with each local function support limited by its corresponding weighting function. Therefore, the marching cube step size depends on the input dataset of cloud points. Here, we use a marching cube step size equal to the minimum influence radius among all local functions.

5.4 Experimental Results

5.4.1 Hardware and software setup

We implemented the LIS algorithm in C++ and GLUT 3.7 (Microsoft Visual Studio 2017) on a desktop computer equipped with an Intel processor Core(TM) i3-4005U, 1.73 GHz, 6 GB RAM, running Windows 8.1. Furthermore, we used MeshLab v1.3.3 (`http://www.meshlab.net/`) to evaluate the mesh reconstruction quality of the benchmarking algorithms (including the LIS algorithm) through

the Hausdorff distance between original meshes and their corresponding reconstructed meshes.

5.4.2 Benchmarking surface reconstruction methods

In this section, we compare the LIS implicit surface reconstruction algorithm to other methods. Two of them are PCR and CTC algorithms we have implemented and presented in the previous two chapters. In addition to those, we used four other methods for benchmarking. The Poisson surface reconstruction algorithm due to Kazhdan et al. [KBH06], the advancing-front meshing algorithm due to Cohen-Steiner and Da [CSD04], and the scale-space reconstruction algorithm due to Digne et al. [DMSL11] which are embed in CGAL (`http://www.cgal.org`). The last method is the Power Crust meshing algorithm due to Amenta et al. [ACK01a] (`http://web.cs.ucdavis.edu/~amenta/powercrust.html`).

5.4.3 Testing mesh models

Our testing point cloud models were obtained from triangle meshes retrieved from Princeton Shape Benchmark (`http://shape.cs.princeton.edu/benchmark/`), 3D Segmentation Benchmark (`http://segeval.cs.princeton.edu/`), tf3dm (`http://tf3dm.com/`), John Burkardt Home Page (`https://people.sc.fsu.edu/~jburkardt/data` and MeshLab Samples (`https://people.sc.fsu.edu/~jburkardt/data/meshlab/meshlab.html`), some of which are listed in Tables 5.1 and 5.2. Such triangle meshes form our ground-truth dataset for benchmarking surface reconstruction methods. Several mesh models are depicted in Figs. 5.3 and 5.4 after reconstruction through LIS algorithm.

Table 5.1: Original meshes and reconstructed meshes generated by LIS, CTC and PCR algorithms.

Model	Label	Original Mesh		LIS Mesh		CTC Mesh		PCR Mesh	
		#Vertices	#Triangles	#Triangles	H	#Triangles	H	#Triangles	H
1	Duck	710	1,416	31,724	8.087	1,414	3.062	1,414	4.639
2	Knot	1,280	2,560	18,516	5.712	2,560	0.281	2,560	0.380
3	Loop	1,440	2,880	27,292	5.845	2,880	1.441	2,880	2.691
4	Dolphin	1,867	3,694	128,022	4.792	3,724	4.413	3,734	4.413
5	Doubletorus	4,352	8,708	17,536	3.451	8,708	2.597	8,708	0.000
6	MaxPlanck	7,399	14,749	135,887	1.495	14,792	21.584	14,796	22.446
7	Egea	8,268	16,532	121,867	1.868	16,529	1,889	16,532	1.889
8	Pear	10,754	21,504	224,913	0.795	21,504	0.153	21,504	0.153
9	Fish	14,000	27,996	208,575	2.268	27,991	0,462	28,005	3.624

Abbreviations:
H: Hausdorff distance

Table 5.2: Reconstructed meshes generated by advancing-front, power-crust, scale-space, and Poisson algorithms.

		Advancing Front Mesh		PowerCrust Mesh		Scale-Space Mesh		Poisson Mesh	
Model	Label	#Vertices	#Triangles	#Triangles	H	#Triangles	H	#Triangles	H
1	Duck	1,416	3.098	13,279	7.705	1,988	27.355	2,974	27.746
2	Knot	2,560	0.281	15,492	23.791	4,356	29.430	2,490	34.903
3	Loop	2,874	7.502	28,613	14.869	4,498	32.077	4,454	38.958
4	Dolphin	3,658	23.400	32,006	3.614	5,436	25.066	6,562	24.008
5	Doubletorus	8,708	3.612	70,132	10.997	16,752	18.413	6,356	14.921
6	MaxPlanck	14,794	22.261	132,560	22.779	28,732	8.932	22,212	22.515
7	Egea	16,531	1.889	155,970	2.065	32,818	6.827	7,132	5.778
8	Pear	21,504	0.127	209,675	1.068	37,840	25.895	5,984	28.969
9	Fish	27,998	5.064	–	–	27,998	5.064	24,080	21.299

Abbreviations:
H: Hausdorff distance

5.4.4 Mesh reconstruction quality

The higher quality of the reconstructed mesh, the closer of the original mesh is to such reconstruction. Hausdorff distance measures how far two subsets of the metrics space are from each other. Thus, we used the Hausdorff distance between two surface meshes as our mesh reconstruction quality metric. Hausdorff distance is defined as follows:

$$H(S, S') = \max\Big(\max_{\mathbf{x}_i \in S} \delta(\mathbf{x}_i, S'), \max_{\mathbf{x}_j \in S'} \delta(\mathbf{x}_j, S)\Big) \quad (5.7)$$

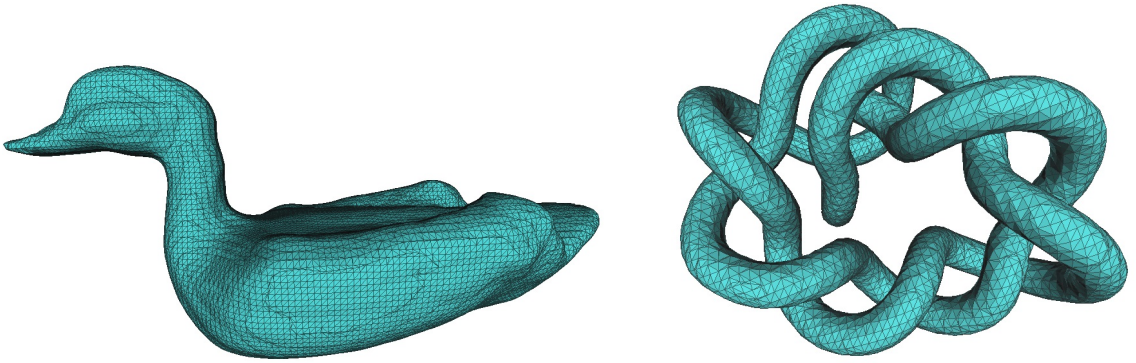
where S, S' denote the two surface meshes, the ground-truth mesh and testing mesh, with

$$\delta(\mathbf{x}_i, S') = \min_{\mathbf{x}_j \in S'} d(\mathbf{x}_i, \mathbf{x}_j) \quad \text{and} \quad \delta(\mathbf{x}_j, S) = \min_{\mathbf{x}_i \in S} d(\mathbf{x}_j, \mathbf{x}_i) \quad (5.8)$$

where $d(\mathbf{x}, \mathbf{x}')$ is the Euclidean distance between points $\mathbf{x}$ and $\mathbf{x}'$.

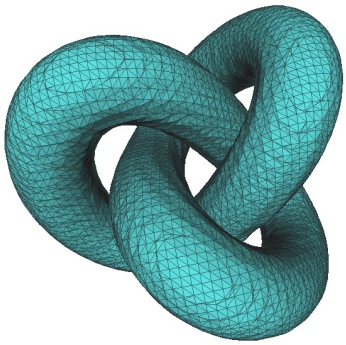
The MeshLab was used to compute the Hausdorff distance during the tests of the LIS algorithm. According to Eq. (5.7), Hausdorff distance applies to sets of points. Therefore, we use the Hausdorff distance to determine the distance between two meshes but considering only vertices. However, doing so for the CTC, PCR, advancing-front, and scale-space meshes, the distance would be zero because those meshes' vertices are the same as the ones of the original mesh. We considered the barycenters of edges and triangles in computing the Hausdorff distance between meshes to overcome the previous problem.

Looking at Tables 5.1 and 5.2 we observe the following:

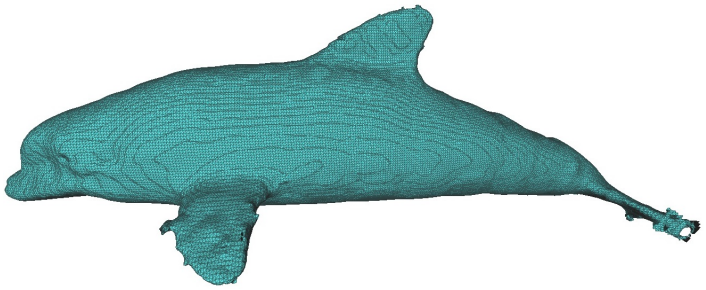


(a) Duck

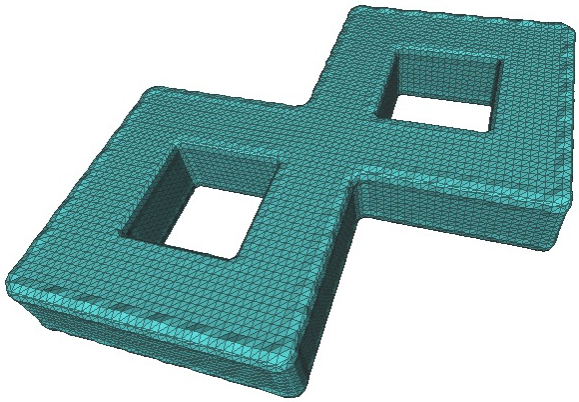
(b) Knot



(c) Loop



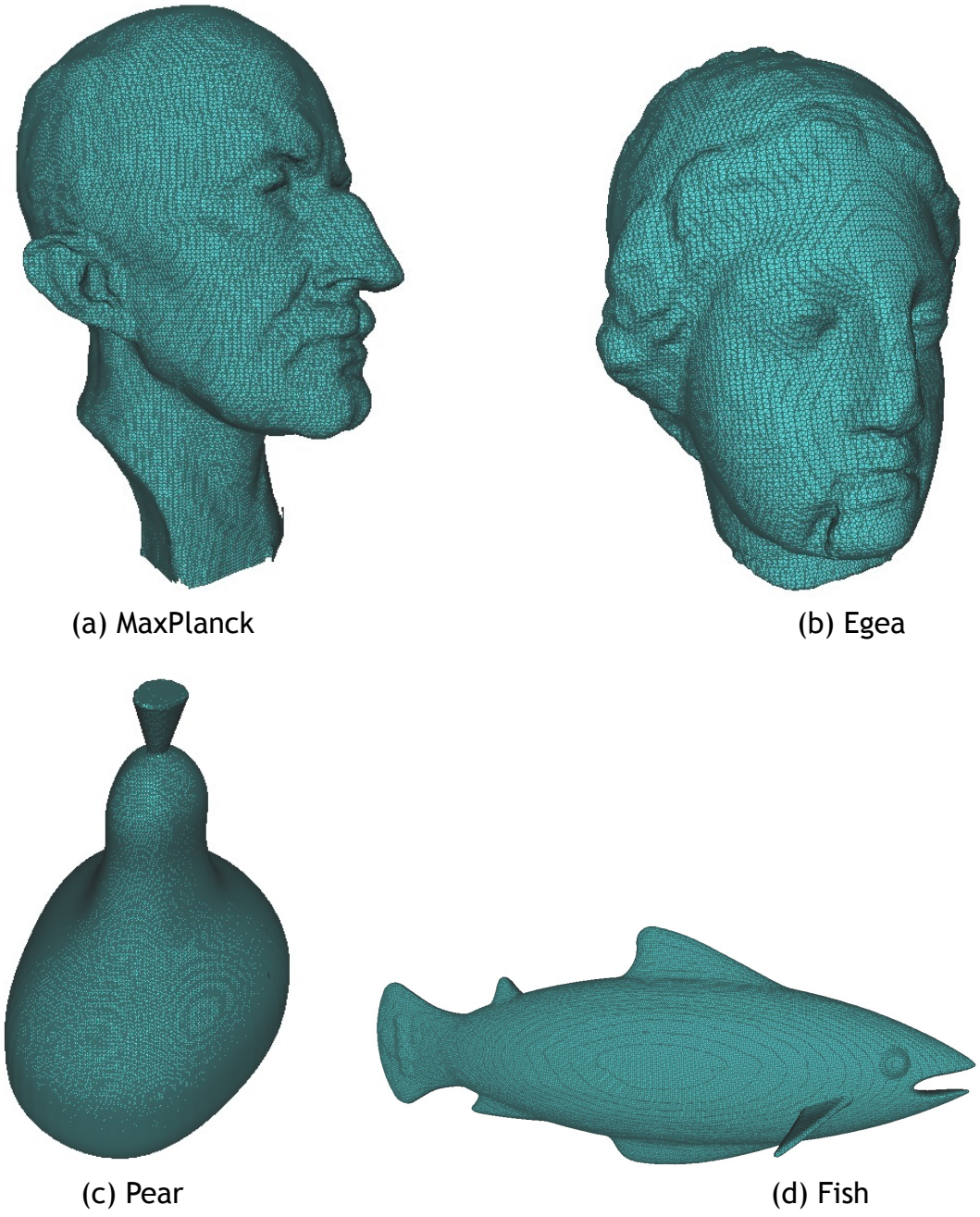
(d) Dolphin



(e) Doubletorus

Figure 5.3: Reconstructed surface meshes using the LIS algorithm.

- *Number of triangles.* The number of triangles in the case of the LIS algorithm is much higher than the original meshes due to the Marching cubes triangulation. In general, the same happens when such a comparison to any other algorithm takes place. Concerning competing algorithms, the number of triangles tends to remain unchanged for interpolated methods like CTC, PCR, and advancing-front methods (see Table 5.1 and 5.2). This fact is not visible in the power-crust algorithm because one adds many more points to the point cloud, producing a mesh with many more triangles. On the other hand, Poisson and scale-space reconstruction methods



(a) MaxPlanck (b) Egea

(c) Pear (d) Fish

Figure 5.4: More reconstructed surface meshes using the LIS algorithm.

generate implicit surfaces that approximate the input point cloud so that such surfaces are *a posteriori* triangulated. In other words, implicit methods produce triangles whose number does not necessarily match those of the original mesh, as shown in Table 5.2.

- *Hausdorff distance*. Recall that the Hausdorff distance is the maximum of the minimum distance between two sets of points (see Eq. (5.7)). As shown in Tables 5.1 and 5.2, the Hausdorff distance between the original meshes and the meshes generated by the LIS, CTC, and PCR algorithms

are, in general, smaller than for the benchmarking surface reconstruction methods (cf. Fig. 5.5). Notwithstanding that the LIS algorithm is an implicit surface method, it performs similarly to CTC and PCR algorithms but in a more consistent manner because it does not depend on the object's shape underlying the point cloud. To grasp this idea better, note that LIS has the smallest Hausdorff distance for model 6 (Max Planck) because it can reconstruct that model keeping the underneath opened, such as in the original mesh. By comparing LIS with reconstruction methods that produce implicit surfaces, such as Poisson and scale-space algorithms, we see that LIS presents much lower Hausdorff distance values (Fig. 5.5). The meshes generated by the LIS, CTC or PCR algorithms can be observed in Figs. 5.3 and 5.4 for the first, 4.12 and 4.13 for the second, and in Figs. 3.9 and 3.10 for the last.

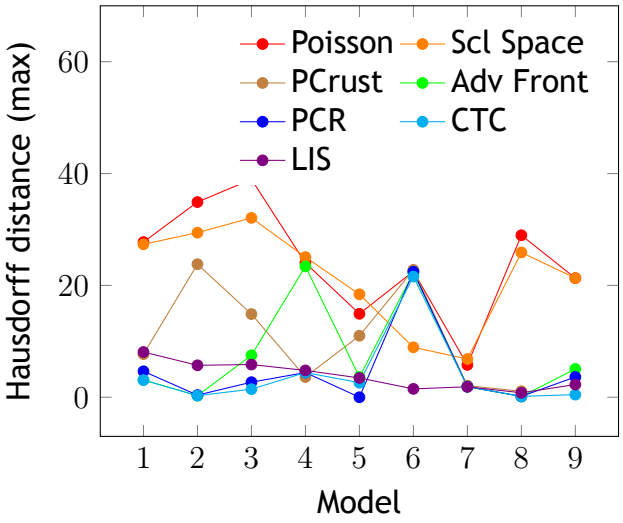


Figure 5.5: Hausdorff distance for reconstructed meshes, comparing LIS, CTC, PCR, power-crust, advancing-front, scale-space, and Poisson algorithms.

Looking at Fig. 5.5, globally, we observe, the interpolated methods (CTC, PCR, advancing-front, and power-crust algorithms) generate meshes closer to the original meshes than methods that approximate the input point cloud (Poisson and scale-space algorithms). Also, the rounding of sharp features is due to the blending of kernel functions of implicit methods. In turn, the LIS implicit method generates meshes close to the original meshes, in line with the interpolated methods and even better than advancing-front and power-crust algorithms. We explain this fact considering that LIS uses local linear functions that interpolate sampling cloud points and variable radii of influence for each one.

To make sure the ranking of these algorithms, we carried out a more refined

analysis of results in terms of other Hausdorff metrics as follows:

- *Hausdorff distance's mean*. This measure represents the mean distance between two sets of points, that is,

$$d_M = \frac{1}{n+m}\Big(\sum_{i=1}^{n}\delta(\mathbf{x}_i, S') + \sum_{j=1}^{m}\delta(\mathbf{x}_j, S)\Big) \quad (5.9)$$

  where $\delta(\mathbf{x}_i, S')$ is the minimum distance from the point $\mathbf{x}_i \in S$ to the point set S' and $\delta(\mathbf{x}_j, S)$ is the minimum distance from the point $\mathbf{x}_j \in S'$ to the point set S, as given by Eq. (5.8).

  In Fig. 5.6, in terms of Hausdorff mean distance, we observe that LIS (implicit method), CTC, PCR, and advancing-front (interpolation algorithms) outperform approximating algorithms. Besides, LIS, CTC, PCR, and advancing-front algorithms are indistinguishable concerning Hausdorff's mean distance for most models.

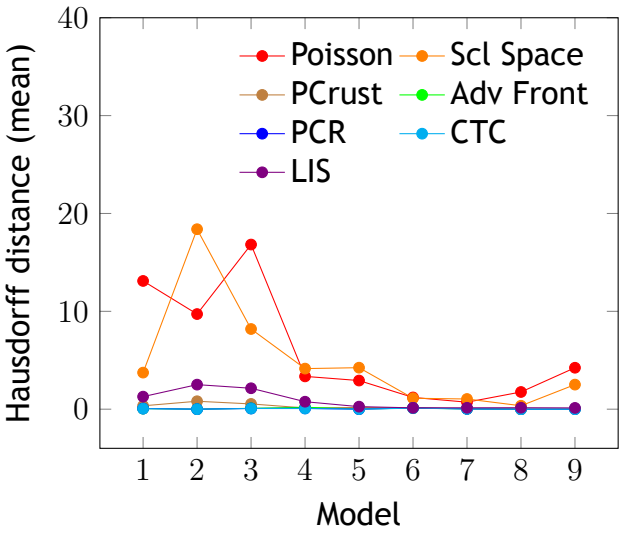


Figure 5.6: Hausdorff distance's mean for reconstructed meshes, comparing LIS, CTC, PCR, power-crust, advancing-front, scale-space, and Poisson algorithms.

- *Hausdorff distance's root mean square* (RMS). This metric measures the arithmetic mean of the squares of the set of minimum distances between two point sets S and S', that is,

$$d_{RMS} = \sqrt{\frac{1}{n+m}\Big(\sum_{i=1}^{n}(\delta(\mathbf{x}_i, S'))^2 + \sum_{j=1}^{m}(\delta(\mathbf{x}_j, S))^2\Big)}. \quad (5.10)$$

  As shown in Fig. 5.7, LIS, CTC, PCR, advancing-front, and power-crust al-

gorithms produce triangulations more similar to the original meshes, which agrees with the results shown in Fig. 5.6, although CTC and PCR algorithms rank first. However, concerning the LIS algorithm, the slightly different triangulations relative to the originals were verified for those models with lower density points.

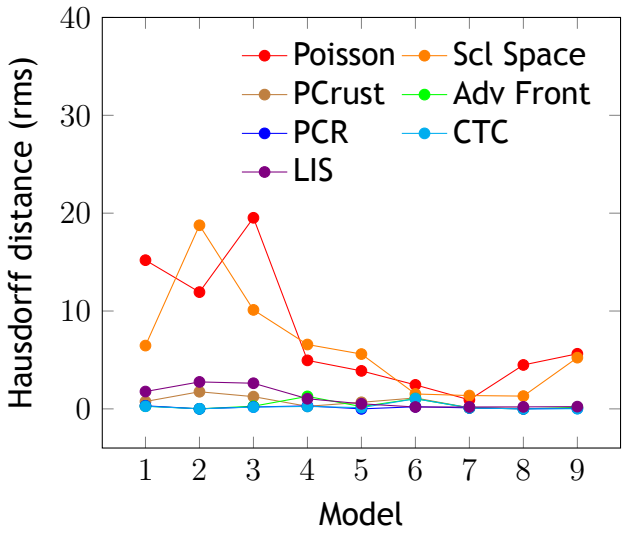


Figure 5.7: Hausdorff distance's root mean square for reconstructed meshes, comparing LIS, CTC, PCR, power-crust, advancing-front, scale-space, and Poisson algorithms.

- *Hausdorff distance's standard deviation*. This metric allows us to the dispersion of the minimum distances in relation to the Hausdorff's mean distance (see Eq. (5.9)), that is,

$$d_{SD} = \sqrt{\frac{1}{n+m}\Big(\sum_{i=1}^{n}(\delta(\mathbf{x}_i, S') - d_M)^2 + \sum_{j=1}^{m}(\delta(\mathbf{x}_j, S) - d_M)^2\Big)}. \quad (5.11)$$

  As shown in Fig. 5.8, CTC and PCR have the lower dispersion of the distances considering all models. In general, the LIS algorithm has a slightly higher dispersion than CTC and PCR. However, the LIS algorithm has lower dispersion than CTC considering model 6 because LIS keeps the underneath hole in the reconstructed mesh like the original mesh.

In general, interpolated methods produce triangulations that tend to be similar to the original meshes because of their interpolating approach. On the other hand, the approximating approach behind implicit methods generates triangulations that are further away from the original meshes, mainly when the implicit surface results from local functions' blending. Although LIS is an im-

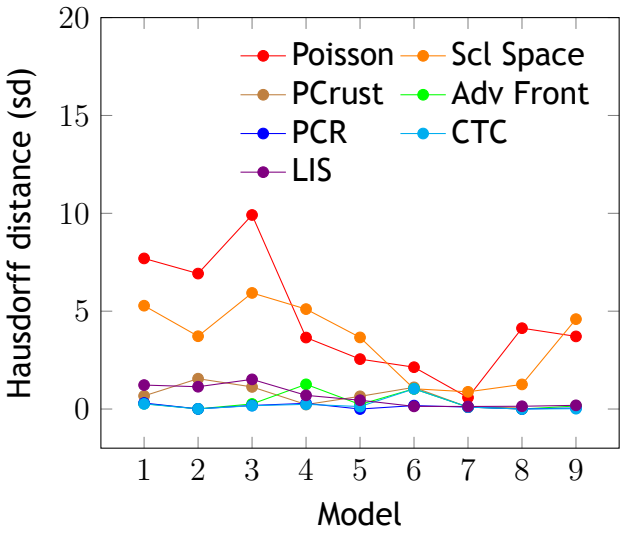


Figure 5.8: Hausdorff distance's standard deviation for reconstructed meshes, comparing LIS, CTC, PCR, power-crust, advancing-front, scale-space, and Poisson algorithms.

plicit method, it can reconstruct meshes very similar to the original meshes in line with the interpolation methods. CTC and PCR algorithms produce the most similar triangulations to the original meshes. However, for most models, LIS performs like those competitors.

5.5 Discussion

A glance at our experiments shows us that, in general, surface reconstruction algorithms work well for uniform or high-density point clouds. However, when the point cloud presents non-uniform, low-density point sampling, it rather difficult to come up with a reconstruction surface algorithm capable of dealing with all the following issues: mesh drifting, holes, and incomplete meshing, as well as trimming and rounding of sharp features.

Also, we see that the simplicial methods generate more robust triangulations than implicit methods (see Fig. 5.9(a)-(b)). This fact is so because simplicial methods interpolate, rather than approximate, the point cloud. However, the power-crust algorithm was not capable of triangulating the fish model (Table 5.2) correctly, possibly because it imposes a predefined minimum distance between points. Also, implicit algorithms are more sensitive than simplicial methods to the presence of non-uniform sampling (Figs. 5.9(c)-(d)), resulting in incomplete or weird triangulations.

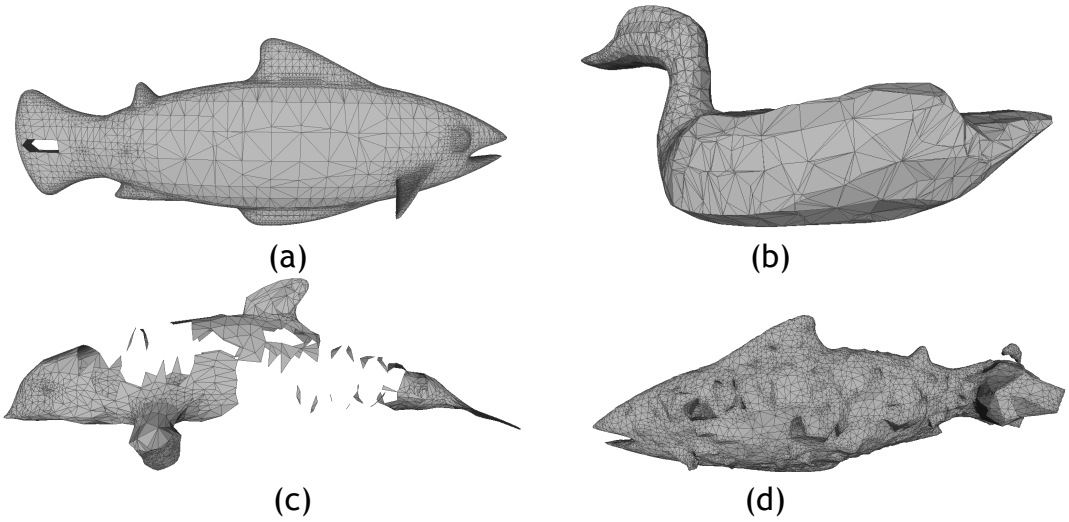


Figure 5.9: Non-uniform point sampling issues: (a) Fish's meshing using advancing-front algorithm originates an hole through the mesh; (b) Duck's meshing using power-crust algorithm increases its volume as a result of more points added to the initial point cloud; (c) incomplete Dolphin's meshing using scale-space algorithm; and (d) weird Fish's meshing using Poisson algorithm.

On the contrary, the LIS algorithm successfully deals with non-uniform point clouds. The following reasons can explain this fact. First, the stars are not so sensitive to the point distribution of the point cloud because the triangle stars cover the whole surface, ensuring the connectivity from each point to its neighbors surrounding it. Second, LIS takes advantage of an interpolation approach so that each local function (the tangent plane) interpolates its corresponding sample point. Besides, its spherical-support weighting function is centered at each point spreading its influence up to its star's mean radius so that the local functions altogether end up covering the entire surface.

5.5.1 Mesh drifting

Figure 5.10 shows the mesh drifting effects on the triangulation generated by other reconstructed surfaces other than LIS. This problem occurs when non-adjacent regions come close to each other, originating triangle shortcuts or even separate parts. In Fig. 5.10(a), the advancing-front algorithm produced a hole due to a shortcut through the left pectoral fin. Furthermore, a small part of the tail fin appears now separate from the triangulation. In Fig. 5.10(b), the Power Crust reconstruction also produced a shortcut between two different knot surface locations. Even so, implicit methods are more sensitive to drifting phenomena than simplicial methods, as illustrated in Figs. 5.10(c) and (d). This sensitivity stems from the fact that most blending functions approximate, rather than interpolate, the cloud points, increasing the probability of non-adjacent

nearby regions touching each other. This fact agrees with the Hausdorff distance results put forward in the previous section.

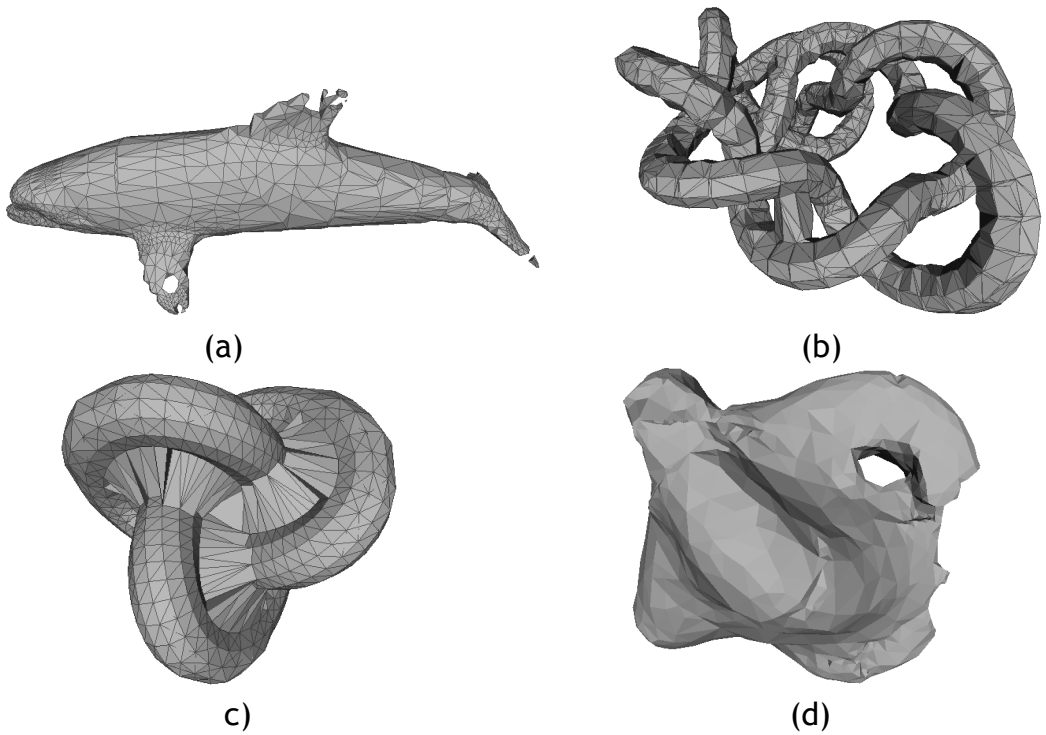


Figure 5.10: Mesh drifting issues: (a) Dolphin's meshing using advancing-front algorithm; (b) Knot's meshing using power-crust algorithm; (c) Loop's meshing using scale-space algorithm; and (d) Loop's meshing using Poisson algorithm.

Despite its implicit nature, the LIS algorithm does not suffer from drifting effects because it builds upon an interpolation approach (see Figures 5.3 and 5.4). That is, the linear implicit surface interpolates the cloud points. In a way, this is equivalent to using a planarity condition because the local shape functions represent tangent planes at cloud points. The quality of the LIS mesh agrees with the Hausdorff distance results presented in the previous section.

5.5.2 Trimming and rounding of sharp features

As known, implicit methods tend to round sharp features, while simplicial algorithms tend to trim sharp features, as illustrated in Figs. 5.11(c)-(d) and Figs. 5.11(a)-(b), respectively.

Regarding the LIS algorithm, the resulting triangulations do not suffer from rounding effects on sharp features (see Figs. 5.3(e) and 5.4(f)), mainly because LIS uses local linear functions interpolating all sample points. Besides, the local function's radius centered at each sample point limits its influence on each point's neighborhood.

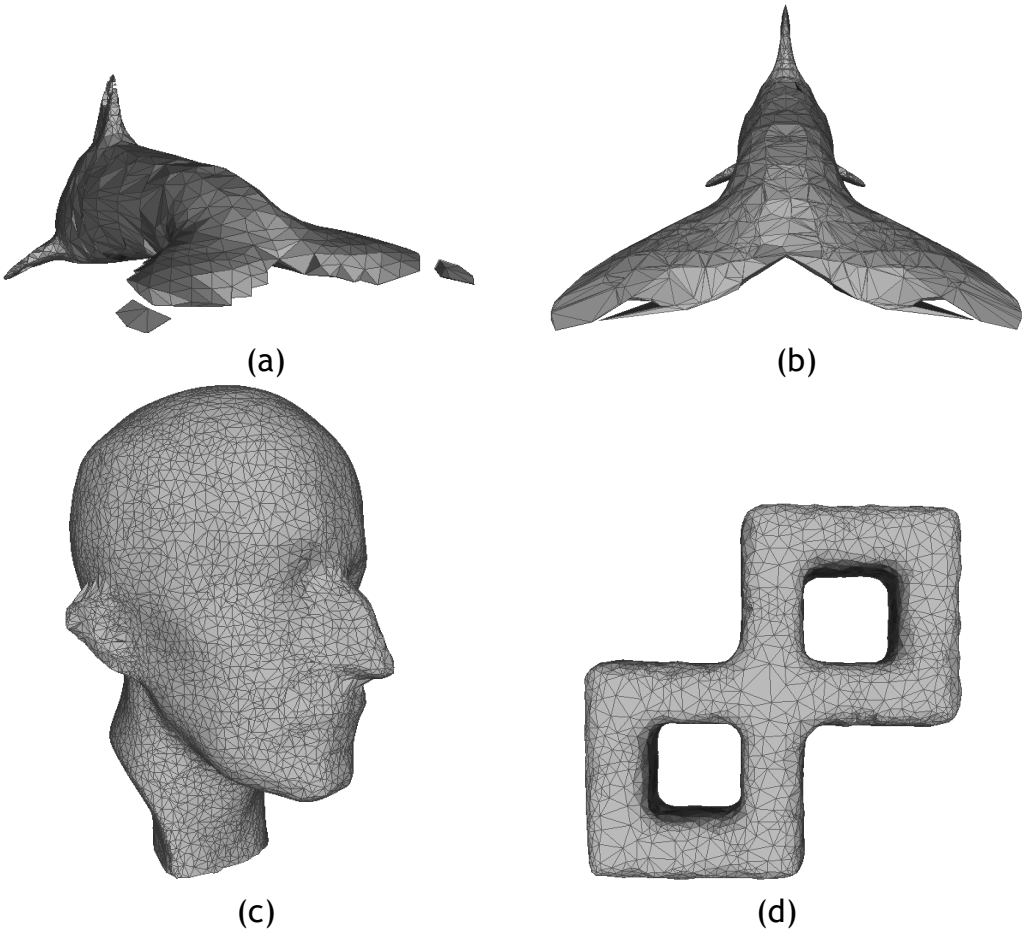


Figure 5.11: Sharp features issues: (a) Dolphin's meshing using advancing-front algorithm originates a total trimming of its fluke; (b) Dolphin's meshing using power-crust algorithm originates a partial trimming of its fluke; (c) Max Planck's meshing using scale-space algorithm; and (d) Double Torus' meshing using Poisson algorithm.

LIS does not cause trimming effects on sharp features either. However, the Dolphin's reconstructed mesh through the LIS algorithm (Fig. 5.3(d)) presents a few extra and divergent triangles on its side fin and back fin. The over-influence of local functions may explain this fact through a few marching cubes. A possible way to solve this issue is to use oblate spherical weighted functions rather than spherical weighted functions.

5.6 Summary

We have introduced a new implicit method to reconstruct surfaces from point clouds, LIS (Linear Implicit Surface) algorithm. Its global function results from the weighted blending of the local functions. We use the Marching Cubes algorithm to triangulate the resulting implicit surface, but we might use any other triangulation technique for implicit surfaces.

This LIS algorithm behaves similarly to the interpolation methods because of the interpolating tangent space of the surface. Indeed, concerning the mesh quality, the Hausdorff measurement results presented in Section 5.4 show us that LIS compares to the interpolation algorithms. That is, LIS produces triangle meshes quite similar to the original meshes. Furthermore, LIS successfully deals with non-uniform point density or sampling, avoiding mesh drifting effects simultaneously. Interestingly, we did not observe the rounding effects that are typical in implicit methods.

Chapter 6

Conclusions and Future Work

Surface reconstruction from a point cloud through interpolation methods is the focus of this thesis. This chapter presents the main conclusions that resulted from the research work described in this thesis. Besides, this chapter puts forward some hints for future work.

6.1 Revisited Research Work

Looking back to work done throughout the doctoral studies, we identify three main milestones:

- *PCR algorithm*: PCR algorithm builds upon three geometric criteria: proximity, coplanarity, and regularity. PCR triangulation gives priority to the most coplanar or lower curvature regions, making dihedral angle bounds unnecessary. It also avoids shape drifting. Moreover, the algorithm is less sensitive to non-uniform point density. In turn, the innovative regularity function allows producing meshes with triangles that tend to be regular, making it unnecessary a regularization step after mesh reconstruction.

- *CTC algorithm*: Building stars of compatible manifold triangles around each cloud point is the leading idea of this algorithm. The construction of each star obeys the criteria of proximity, coplanarity, and regularity. However, when it comes to matching triangle stars, the manifoldness is the main criterion to make them compatible. This strategy makes this algorithm different from any other simplicial algorithm. It is the first algorithm to take advantage of the atlas of charts, where each chart represents a triangle star.

- *LIS algorithm*: LIS is an interpolating implicit algorithm in line with other methods based on RBFs (Radial Basis Functions) [GVJ+09]. However, the LIS algorithm than RBF-based algorithms because it does use time-consuming matrix computations. Also, unlike MPU-based algorithms, [OBA+03], LIS does not use each octree leaf subdomain's center to approximate the

surface. Instead, each point is the center of a subdomain defined by its star. Besides, each point is associated with a linear local function, which represents its tangent plane. Also, each point is the center of the weighting function that controls the linear local function's influence. Thus, the reconstructed surface interpolates the cloud points.

6.2 Conclusions

This research work's main contribution is the general geometric framework we used to design and implement distinct surface reconstruction algorithms from point clouds. Such a framework builds upon four conditions: proximity, planarity, regularity, and manifoldness. We proved that it is feasible to design and implement surface reconstruction algorithms that produce surfaces with topological guarantees.

The three algorithms fit into the category of interpolation methods, although they fall into different approaches:

- *PCR algorithm*: An interpolating simplicial algorithm that uses a mesh-growing approach to interpolate the sampling points.
- *CTC algorithm*: An interpolating simplicial algorithm that uses an approach based on triangle stars' compatibility. These stars cover the surface wholly.
- *LIS algorithm*: An interpolating implicit algorithm that interpolates the sampling points using local linear implicit functions representing their tangent planes.

The first two surface reconstruction algorithms prioritize lower curvature regions (or more planar regions) concerning triangulation. It is unnecessary to use such a priority in the third algorithm because each point's neighborhood lies in its tangent plane. This fact avoids mesh drifting effects, making it also unnecessary to impose any dihedral angle bounds. These algorithms produce meshes with quasi-regular triangles, making unnecessary any post-processing step to regularize the final mesh.

Also, let us mention that the second algorithm opens a window for a new category of surface reconstruction algorithms. In turn, the third algorithm is the first interpolating implicit algorithm we may find in the literature that does not perform time-consuming matrix computations.

6.3 Future Work

We anticipate three possible improvements for our algorithms:

- *Proximity*. The algorithms introduced in this dissertation use the concept of proximity to find a set of neighbors for each point. This procedure is time-consuming and takes longer when the number of sampling points increases. We used the octree subdivision to speed up this procedure to find the neighbors of each point. For that, we had to impose an empirical threshold as subdivision stopping criterion, that is, a minimum number of 24 points within the octree's leaf cell. Therefore, it would be helpful to look for another approach to find each point's neighbors without using empirical thresholds.

- *Compatibility*. Another research track for future work would be to improve the CTC algorithm's compatibility condition to reduce or even eliminate mesh holes. Thus, there would make redundant the hole-repairing step at the end of this algorithm.

- *Over-influence*. To improve the LIS algorithm, it would fine to avoid the over-influence of local functions over some regions of the surface, as occurred in the case of Dolphin's fins with the appearance of some extra and divergent triangles (Fig. 5.3(d)). A possible solution is to use *oblate* spheroidal weighting functions rather than spherical weighting functions because oblate spheroidal functions adapt better to the surface's local shape.

Finally, let us say that these algorithms' codes will be publicly available from a repository at `gitub.com`.

Bibliography

[AB99] N. Amenta and M. Bern. Surface reconstruction by voronoi filtering. In *Discrete and Computational Geometry*, volume 22, pages 481–504. Springer-Verlag, 1999. 12, 16, 17

[ABK98] N. Amenta, M. Bern, and M. Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (SIGGRAPH'98), pages 415–421. ACM press, 1998. 2, 12, 13

[ACDL00] N. Amenta, S. Choi, T. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. *International Journal of Computational Geometry and Applications*, 12(6):125–141, 2000. 2, 16, 17, 21

[ACK01a] N. Amenta, S. Choi, and R. Kolluri. The power crust. In *Proceedings of 6th ACM Symposium on Solid Modeling*, pages 249–260. ACM Press, 2001. 14, 19, 45, 47, 71, 76, 92

[ACK01b] N. Amenta, S. Choi, and R. Kolluri. The power crust , unions of balls and the medial axis transform. In *Computational Geometry: Theory and Application*, volume 9, pages 127–153. ACM Press, 2001. 14, 19

[AGJ00] Udo Adamy, Joachim Giesen, and Matthias John. New techniques for topologically correct surface reconstruction. In *Proceedings of the IEEE Conference on Visualization* (VIS'00), Salt Lake City, Utah, USA, pages 373–380. IEEE Computer Society Press, 2000. 25

[ASG11] L. D. Angelo, P. H. Stefano, and L. Giaccari. A new mesh growing algorithm for fast surface reconstruction. *Computer Aided Design*, 43(6):639–650, 2011. 25

[BMR+99] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999. 2, 22

[BTS+17] M. Berger, A. Tagliasacchi, L. Seversky, P. Alliez, G. Guennebaud, J. Levine, A. Sharf, and C. Silva. A survey of surface reconstruc-

tion from point clouds. *Computer Graphics Forum*, 36(1):301–329, 2017. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12802. 2

[CBC+01] J. Carr, R. Beatson, J. Cherrie, T. Mitchell, W. Fright, B. McCallum, and T. Evanse. Reconstruction and representation of 3d objects with radial basis functions. *In Proceedings of ACM SIGGRAPH*, pages 67–76, 2001. 3, 28

[CBM+03] J. Carr, R. Beatson, B. McCallum, T. McLennan W. Fright and, and T. Mitchell. Smooth surface reconstruction from noisy range data. In *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South-East Asia (GRAPHITE'03), February 11-14, Melbourne, Australia*, pages 119–126. ACM Press, 2003. 28

[CFB97] J. Carr, W. Fright, and R. Beatson. Surface interpolation with radial basis functions for medical imaging. *IEEE Transactions on Medical Imaging*, 16(1):96–107, feb 1997. 27

[CSD04] David Cohen-Steiner and Frank Da. A greedy Delaunay-based surface reconstruction algorithm. *The Visual Computer*, 20(1):4–16, apr 2004. 2, 23, 45, 71, 92

[DDW11] T. Dey, R. Dyer, and L. Wang. Localized cocone surface reconstruction. *Computers and Graphics*, 35:483–491, 2011. 19, 20

[DG01] T. Dey and J. Giesen. Detecting undersampling in surface reconstruction. *Proceedings of 17th Symposium on Computational Geometry*, pages 257–263, 2001. 18

[DG03] T. Dey and S. Goswami. Tight cocone: a water-tight surface reconstruction. In *Proceedings of 8th ACM Symposium on Solid Modeling and Applications*, pages 127–134. ACM Press, 2003. 17, 18

[DG06] T. Dey and S. Goswami. Provable surface reconstruction from noisy samples. *Computational Geometry: Theory and Application*, 35(1-2):124–141, 2006. Available from: http://www.sciencedirect.com/science/article/pii/S0925772105000969. 19

[DGGZ02] T. Dey, J. Giesen, S. Goswami, and W. Zhao. Shape dimension and

approximation from samples. In *Symposium on Discrete Algorithms*, pages 772–780, 2002. 21

[DGH01] T. Dey, J. Giesen, and J. Hudson. Delaunay based shape reconstruction from large data. *Proceedings IEEE of Symposium in Parallel and Large Data Visualization and Graphics*, pages 19–27, 2001. 17, 19

[DGQ+12] T. Dey, X. Ge, Q. Que, I. Safa, L. Wang, and Y. Wang. Feature-preserving reconstruction of singular surfaces. *Computers Graphics Forum*, 35(5):1787–1796, 2012. 21

[Dig14] J. Digne. An analysis and implementation of a parallel ball pivoting algorithm. *Image Processing On Line*, 4:149–168, 2014. 22

[Dig15] Julie Digne. An implementation and parallelization of the scale-space meshing algorithm. *Image Processing On Line*, 5:282–295, 2015. 24

[DK99] T. Dey and P. Kumar. A simple provable algorithm for curve reconstruction. In *Symposium on Discrete Algorithms*, pages 893–894, 1999. 21

[DMSL11] Julie Digne, Jean-Michel Morel, Charyar-Mehdi Souzani, and Claire Lartigue. Scale space meshing of raw data point sets. *Computer Graphics Forum*, 30(6):1630–1642, 2011. 3, 24, 45, 71, 92

[DS05] T. Dey and J. Sun. An adaptive mls surface for reconstruction with guarantees. *Symposium on Geometry Processing*, pages 43–52, 2005. Available from: `ftp://ftp.cse.ohio-state.edu/pub/tech-report/2005/TR26.pdf`. 3

[Duc77] J. Duchon. Splines minimizing rotation-invariant semi-norms in sobolev spaces. in constructive theory of functions of several variables. *Lecture Notes in Mathematics*, 571:85–100, 1977. 28

[DW13] T. Dey and L. Wang. Voronoi-based features curves extraction for sampled singular surfaces. *Computers and Graphics*, 37(6):659–668, 2013. 21

[DZ02] T. Dey and W. Zhao. Approximate medial axis as a voronoi subcomplex. *Proceedings of 7th Symposium in Solid Modeling and Applica-*

tions, pages 356–366, 2002. 18

[FH05] M. Floater and K. Hormann. Surface parameterization: a tutorial and survey. In N. Dodgson, M. Floater, and M. Sabin, editors, *Advances in Multiresolution for Geometric Modelling*, pages 157–186. Springer, Berlin, Germany, 2005. 3

[Fie73] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973. 15

[Gro93] K. Grover. Critical point theory for distance functions. *In Proceedings of Symposia in Pure Mathematics*, pages 357–385, 1993. 3

[GVJ+09] A. Gomes, I. Voiculescu, J. Jorge, B. Wyvill, and C. Galbraith. *Implicit curves and surfaces: mathematics, data structures and algorithms*. Springer, London, United Kingdom, 2009. x, 2, 3, 105

[Hal70] K. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 11(3):219–229, 1970. 15

[HDD+92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):71–78, 1992. Available from: `http://research.microsoft.com/en-us/um/people/hoppe/proj/recon/`. 3

[KBH06] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In Konrad Polthier and Alla Sheffer, editors, *Proceedings of the Eurographics Symposium on Geometry Processing* (SGP'06), Cagliari, Sardinia, Italy, June 26-28, pages 61–70. Eurographics Association Press, 2006. 3, 29, 45, 71, 92

[KBSS01] L. Kobbelt, M. Botsch, U. Schwanecke, and H-P. Seidel. Feature sensitive surface extraction from volume data. *ACM SIGGRAPH Computer Graphics*, 33(3):57–66, 2001. 3

[KSO04] R. Kolluri, J. Shewchuk, and J. Obrien. Spectral surface reconstruction from noisy point clouds. In *Eurographics Symposium on Geometry Processing*. The Eurographics Association, 2004. 15

[KvD92] Jan Koenderink and Andrea van Doorn. Surface shape and curvature scales. *Image and Vision Computing*, 10(8):557–564, October 1992.

61

[KY03] C. Kuo and H. Yau. Reconstruction of virtual parts from unorganized scanned data for automated dimensional inspection. *Journal of Computing and Information Science in Engineering*, 3(1):76–86, 2003. 3

[KY05] C. Kuo and H. Yau. A delaunay-based region-growing approach to surface reconstruction from unorganized points. *Computer-Aided Design*, 37(8):825–835, 2005. 3

[KY06] C. Kuo and H. Yau. A new combinatorial approach to surface reconstruction with sharp features. *IEEE Transactions on Visualization and Computer Graphics*, 12(1):825–835, 2006. 3

[Laf15] Jacques Lafontaine. *An Introduction to Differential Manifolds*. Springer, Switzerland, 2015. 59

[LC87] W. Lorensen and H. Cline. Marching cubes: a high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987. xi, 4, 91

[LHW09] X. K. Li, C. Y. Han, and W. G. Wee. On surface reconstruction: A priority driven approach. *Computer Aided Design*, 41(9):626–640, 2009. 25, 44

[Nie93] G. Nielson. Scattered data modeling. *IEEE Computer Graphics and Applications*, 13(1):60–70, jan 1993. 27

[NY06] T. S. Newman and H. Yi. A survey of the marching cubes algorithm. *Computer and Graphics*, 30:854–879, 2006. 91

[OBA+03] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H-P. Seidel. Multilevel partition of unity implicits. *ACM Transactions on Graphics*, 22(3):463–470, 2003. xii, 3, 5, 87, 105

[PL03] H. Pottmann and S. Leopoldseder. A concept for parametric surface fitting which avoids the parametrization problem. *Computer Aided Geometric Design*, 20(6):343–362, 2003. 3

[PSL90] A. Pothen, D. Simon, and K. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal on Matrix Analysis and Appli-*

cations, 11(3):430–452, July 1990. 15

[RY07] L. Rineau and M. Yvinec. A generic software design for delaunay refinement meshing. *Computational Geometry Theory and Applications*, 38:100–110, 2007. 30

[SMG10] J. SuBmuth, Q. Meyer, and G. Greiner. Surface reconstruction based on hierarchical floating radial basis functions. *Computer Graphics Forum*, 29(6):1854–1864, 2010. 3

[SPOK95] V. Savchenko, A. Pasko, O. Okunev, and T. Kuni. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14(4):181–188, 1995. 27

[TO99] G. Turk and J. O'Brien. Shape transformations using variational implicit surfaces. *ACM SIGGRAPH Computer Graphics*, 33(3):335–342, aug 1999. 27

[Wen95] H. Wendland. Piecewise polynomial, positive definite and compactly supported radial basis functions of minimal degree. *Advances in Computational Mathematics*, 4(1):389–396, dec 1995. 29

[Wit83] P. Witkin. Scale-space filtering. In *8th Int. Joint Conf. Artificial Intelligence*, volume 2, pages 1019–1022, 1983. 24

[WTS12] N. Wongwaen, S. Tiendee, and C. Sinthanayothin. Method of 3d mesh reconstruction from point cloud using elementary vector and geometry analysis. In *Proceedings of the International Conference on Information Science and Digital Content Technology*, volume 1, pages 156–159, jun 2012. xi, 4, 26, 44

[WZZW13] N. Wang, Q. Zhang, D. Zhou, and X. Wei. Local optimum triangulation for unorganized point cloud. *Research Journal of Applied Sciences, Engineering and Technology*, 6(10):1862–1867, 2013. xi, 4, 26

[XLC14] I. Xumin, Y. Lixin, and L. Cailing. A robust mesh growing surface reconstruction algorithm based on octree. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, 7(3):135–146, 2014. xi, 2, 4, 26, 44