Reconstruction and Rendering of Implicit Surfaces from Large Unorganized Point Sets

Patrick Reuter

▶ **To cite this version:**

Patrick Reuter. Reconstruction and Rendering of Implicit Surfaces from Large Unorganized Point Sets. Human-Computer Interaction [cs.HC]. Université Sciences et Technologies - Bordeaux I, 2003. English. NNT : . tel-00576950

HAL Id: tel-00576950

https://theses.hal.science/tel-00576950

Submitted on 15 Mar 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : 2793

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ BORDEAUX 1

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Patrick REUTER**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Reconstruction and Rendering of Implicit Surfaces from Large Unorganized Point Sets

---

Soutenue le : 12 décembre 2003

Après avis des rapporteurs :

Marc Alexa	Professeur, Université Technique de Darmstadt
George Drettakis .	Directeur de Recherche INRIA (HDR)

Devant la commission d'examen composée de :

Marc Alexa	Professeur, Université Technique de Darmstadt	Rapporteur
Kadi Bouatouch ..	Professeur, Université Rennes 1	Président du jury
George Drettakis .	Directeur de Recherche INRIA (HDR)	Rapporteur
Laurent Grisoni ..	Maître de Conférences, Polytech' Lille	Examinateur
Pascal Guitton ...	Professeur, Université Bordeaux 1	Examinateur
Christophe Schlick	Professeur, Université Bordeaux 2	Directeur de Thèse

– 2003 –

à Raphaël,
mon filleul

Remerciements

Mes remerciements s'adressent en premier à mon directeur de thèse Christophe Schlick. Il a fait preuve d'un soutien sans faille et a toujours su exprimer sa confiance en moi, non seulement dans les moments de succès mais aussi dans les moments difficiles. Ses qualités pédagogiques et scientifiques extraordinaires m'ont toujours poussés jusqu'au bout.

Je tiens à remercier mes deux rapporteurs de thèse, Marc Alexa et George Drettakis, pour m'avoir fait l'honneur de lire attentivement mon mémoire et pour être venus à Bordeaux lors de ma soutenance. Leurs commentaires sur mon mémoire ont été très enrichissants, et leur présence à la soutenance m'a fait ressentir une valorisation importante de mon travail effectué.

Je remercie Kadi Bouatouch d'avoir accepté de présider mon jury. Je le remercie aussi pour les nombreuses discussions que nous avons menées non seulement à Bordeaux, mais aussi dans le cadre de nombreuses missions à Rennes, Londres ou Paris.

Je tiens à remercier Pascal Guitton pour son talent de mener notre équipe de recherche à Bordeaux et de mettre en avant nos résultats à l'extérieur. Etant donné que je n'ai pas eu de bourse de thèse, c'est grâce à Pascal que j'ai pu démarrer ma thèse sur un contrat ingénieur.

Je remercie Laurent Grisoni d'avoir accepté d'être examinateur de ma thèse et de s'avoir déplacé à Bordeaux lors de la soutenance. C'est lui qui m'a donné le précieux conseil de démarrer une thèse sous la direction de Christophe Schlick.

Je tiens à remercier Johannes Behr pour l'impulsion initiale qu'il m'a donné pour travailler dans l'informatique graphique au sein de l'Institut Fraunhofer.

Je remercie Ireneusz Tobor pour sa gentillesse et disponibilité tout au long de ma thèse. Sans lui, ma thèse n'aurait pas été ce qu'elle est.

Je tiens à remercier Martin Hachet pour son chaleureux accueil depuis mon arrivé au LaBRI et Benjamin Schmitt pour des moments de travail intenses et des discussions ouvrant l'esprit.

Je remercie Carole Blanc pour les nombreuses discussions et pour l'âme qu'elle apporte à notre équipe.

Je remercie mes deux stagiaires Tamy Boubekeur et Julien Hadim pour leur enthousiasme envers l'informatique graphique et pour avoir partagé mon bureau « sauna » pendant la rédaction cet été.

Je tiens à remercier de nombreuses autres personnes au LaBRI pour l'ambiance et les conditions de travail agréables, parmi mes compères doctorants ainsi que Philippe Biais, Pierre Casteran, Robert Strandh, et Anne Vialard.

Je remercie de nombreuses autres personnes d'autres institutions pour des discussions scientifiques, et en particulier Alexander Pasko et Loïc Barthe.

Je tiens à remercier Flo, Baddi, Robbie et Rouss pour leur amitié et pour de nombreuses

discussions profondes qui amènent l'esprit toujours plus loin. Je remercie mes amis Alex, Axel, et Jérôme de m'avoir fait découvrir ma passion du surf. Je tiens à remercier mes amis Roland et Max pour les encouragements et ravitaillements sur la dernière ligne droite. Je remercie mon amie Nat pour son existence et sa présence qui fait à tous les moments qu'on passe ensemble des moments extraordinaires, ainsi que pour son soutien tout au long de ma thèse. Je remercie tous mes autres amis pour des moments inoubliables qu'on a passé ensemble.

Finalement, je remercie mes parents et ma soeur pour leur soutien et leur amour depuis toujours.

Reconstruction and Rendering of Implicit Surfaces from Large Unorganized Point Sets

Abstract: Recent three-dimensional acquisition technologies provide a huge number of unorganized points in three dimensions. It is desirable to reconstruct a continuous surface representation that is faithful to the unorganized points for further processing, and to render the resulting surfaces in order to get a visual feedback.

In this thesis, we present new methods to reconstruct implicit surfaces from large unorganized point sets. The methods are based on locally reconstructed variational surfaces using radial basis functions that are blended together by applying a partition of unity.

In order to get an interactive visual feedback of the generated surfaces, we present new rendering techniques that use not only the reconstructed implicit surfaces, but also the initial unorganized point set. This rendering is either done view-dependently in an output-sensitive multiresolution manner using points as rendering primitive, or by using local differential geometry for every point in the point set. Finally, we discuss a wide variety of applications and potential applications of the presented fundamentals, such as interactive construction of procedural solid textures from unorganized point sets, reconstruction of heightfields from contour lines, or repairing of damaged photographs.

Keywords: implicit surfaces, surface reconstruction, radial basis functions, point-based rendering

Discipline: Computer Science

LaBRI,
Université Bordeaux 1,
351, cours de la libération
33405 Talence Cedex (FRANCE)

Reconstruction et Rendu de Surfaces Implicites à partir de grands ensembles de points non-structurés

Résumé : Les technologies récentes d'acquisition de données en trois dimensions fournissent un grand nombre de points non-structurés en trois dimension. Il est important de reconstruire une surface continue à partir de ces points non-structurés et de la visualiser.

Dans ce document, nous présentons de nouvelles méthodes pour reconstruire des surfaces implicites à partir de grands ensembles de points non-structurés. Ces méthodes mettent en oeuvre des surfaces variationnelles reconstruites localement à partir de fonctions de base radiales, surfaces qui sont combinées entre elles par un mécanisme de partition de l'unité. Afin d'obtenir une visualisation interactive des surfaces générées, nous présentons également des techniques de rendu qui utilisent non seulement la surface implicite reconstruite, mais également l'ensemble de points initial. Une première technique de rendu à base de points s'adapte automatiquement en fonction de la position de l'observateur et de la taille de la fenêtre de visualisation, grâce à une structure hiérarchique à multirésolution, et une deuxième technique de rendu à base de points utilise la géométrie différentielle locale dans chaque point. Enfin, un grand nombre d'applications effectives ou d'applications potentielles des techniques précédentes sont présentées, telles que la construction interactive de textures solides à partir de points non-structurés, la reconstruction altimétrique de terrain en fonction des lignes de niveaux, ou encore la réparation de photographies abîmées.

Mots clés : surfaces implicites, reconstruction de surfaces, fonctions de base radiales, rendu à base de points

Discipline: Informatique

LaBRI,
Université Bordeaux 1,
351, cours de la libération
33405 Talence Cedex (FRANCE)

Contents

List of Figures

List of Tables

Introduction

Recent three-dimensional acquisition technologies such as laser range scanners, light detection and ranging (LIDAR), mechanical touch probes, and computer vision techniques such as depth from stereo, provide a huge number of unorganized points in three dimensions (3D). It is desirable for further processing to get a continuous surface representation in a small amount of time with a low memory overhead that is either faithful to such an unorganized point set or tolerates noise. In order to get a visual feedback of the reconstructed continuous surface, it is rendered on an output device, the most often a computer display.

Basically, there are three approaches to represent a continuous two-dimensional (2D) surface embedded in 3D space. First, the surface can be represented by an *explicit equation*, where all points $\boldsymbol{p} = [x, y, z]^T$ on the surface are expressed as z coordinate in terms of the x and y coordinates, i.e. $f_{explicit}(x, y) = z$. The resulting surface is called a *heightfield*, but the surfaces that can be represented are limited, for example it is easy to see that closed surfaces cannot be described.

The second approach is to represent surfaces as *parametric surfaces*, where the surface is represented as a function $\Pi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ that provides a mapping of the 2D surface into the 3D space in which it is embedded. Unfortunately, topology issues have to be considered.

In this thesis, we focus on a third approach to represent continuous surfaces, namely the *implicit surfaces*. A general implicit surface $\mathcal{S}$ is defined as the zero-set of a scalar *defining function* $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ with

$$f(\boldsymbol{x}) = 0. \tag{1}$$

So, the implicit surface is defined as $\mathcal{S} = \{\boldsymbol{x} \in \mathbb{R}^3 | f(\boldsymbol{x}) = 0\}$, sometimes also called *isosurface* at the value 0. Based on a common convention, we define the interior of the surface to be at all points where the defining function is positive, i.e. $f(\boldsymbol{x}) > 0$. Accordingly, we define the exterior of the surface to be at all points where the defining function is negative, i.e. $f(\boldsymbol{x}) < 0$. Note that a nice characteristic of implicit surfaces with continuous and differentiable defining functions is, that the normal $\boldsymbol{n}$ can be calculated using the gradient of the defining function

$$\nabla f(\boldsymbol{p}) = \left[\frac{\partial f}{\partial x}(\boldsymbol{p}), \frac{\partial f}{\partial y}(\boldsymbol{p}), \frac{\partial f}{\partial z}(\boldsymbol{p})\right] \tag{2}$$

at any point $\boldsymbol{p}$ on the surface with non-zero gradient.

Representing objects and their surfaces implicitly with a defining function has a number of advantages, in particular implicit surfaces

- have a mesh-independent representation, but the mesh can be generated when it is required,
- are compactly represented within any desired precision in a simple data structure,
- are thus memory storage efficient,
- have an associated solid model allowing efficient point membership classification and collision detection,
- are guaranteed to be manifold surfaces with no self intersections and are thus manufacturable,
- allow analytical smoothing and anti-alias filtering as well as stepless zooming, and
- are topologically flexible.

In this thesis, we show how to reconstruct implicit surfaces from unorganized point sets and how to render the resulting surfaces. Moreover, we discuss a wide variety of applications using the underlying fundamentals. This thesis is organized as follows:

Part I: Reconstruction of Implicit Surfaces from Large Unorganized Point Sets

We show in Part I how implicit surfaces can be reconstructed from an unorganized point set.

In particular, we start by a discussion of previous work in Chapter 1 and present a first new method that we call the partition of unity variational method in Chapter 2. This method combines locally reconstructed variational implicit surfaces together by applying a partition of unity blending.

In Chapter 3, we present a second new method that we call the hierarchical partition of unity variational method that extends the ideas of partition of the unity variational method by hierarchically applying a partition of unity blending to the locally reconstructed variational implicit surfaces.

Part II: Rendering of Implicit Surfaces from Large Unorganized Point Sets

In Part II, we focus on rendering of reconstructed implicit surfaces from unorganized points on an output-device in order to get a visual feedback.

In particular, we present in Chapter 4 an extensive survey of previous work on rendering the discrete surface representation, i.e. the unorganized point set, some previous work on rendering the reconstructed continuous surface representation, i.e. the implicit surface, as well as some previous work on rendering that uses both these types of surface representation.

In Chapter 5, we present a new point-based rendering technique that also uses both these types of surface representation. In this technique, the unorganized points are structured in a bounding sphere hierarchy for rendering. Whenever the projected size of the spheres is above a threshold, new points on the surface are generated using the reconstructed implicit surface representation.

In Chapter 6, we present another point-based rendering technique for implicit surfaces reconstructed from unorganized points that is running on modern programmable graphics hardware. The local differential geometry for every point in the unorganized point set is extracted in a preprocess, and then every point is rendered as a fragment-shaded rectangle.

Part III: Applications

In the final part, we present an incomplete list of potential applications that use the fundamentals of the first two parts of this thesis.

In Chapter 7 we show how the ideas of the partition of unity variational method and the hierarchical partition of unity variational method can be used in further reconstruction applications than implicit surfaces. In particular, we define a new class of procedural solid texture that can be reconstructed from unorganized points, we show the reconstruction of terrains from 2D contour lines as well as applications in 2D image processing.

In Chapter 8, we present a new interactive environment for constructive texturing of surfaces of arbitrarily defined 3D objects (including, of course, implicit surfaces). A user can texture the surface by defining space partitions that are combined using constructive texturing, and by specifying attributes that are applied in the space partition. The partition of unity variational method can be used to define space partitions amongst other primitives. When specifying the attributes, the new class of procedural solid textures can be used as well. In order to give an interactive feedback, a point-based multiresolution representation of the surface is used that is not only exploited for rendering, but also for the evaluation of the texture.

Part I

Reconstruction of Implicit Surfaces from Large Unorganized Point Sets

In this part, we present different methods to reconstruct implicit surfaces from large unorganized point sets. We start by giving some formal definitions.

Let $\mathcal{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_N\}$ be a an unorganized point set consisting of N points $\boldsymbol{p}_i \in \mathbb{R}^3$ without any connectivity information. As a tenor, we will make no further assumptions on the distribution of the point set $\mathcal{P}$, unless we assume *quasi-uniformly distributed* points. We define quasi-uniformity according to Schaback [139]: a point set $\mathcal{P}$ is quasi-uniformly distributed on a bounded domain $\Omega \subset \mathbb{R}^3$, when the quotient of the fill distance $h_{\mathcal{P},\Omega} = \max_{\boldsymbol{x}\in\Omega} \left(\min_{\boldsymbol{p}_i\in\mathcal{P}} \left(\|\boldsymbol{x} - \boldsymbol{p}_i\|\right)\right)$ and the separation distance $q_{\mathcal{P}} = \frac{1}{2}\min_{\boldsymbol{p}_i,\boldsymbol{p}_j\in\mathcal{P},\boldsymbol{p}_i\neq\boldsymbol{p}_j} \left(\|\boldsymbol{p}_i - \boldsymbol{p}_j\|\right)$ is bounded above by a constant.

Starting from the point set $\mathcal{P}$, we want to reconstruct an implicit surface that passes either through the points of $\mathcal{P}$, we will then refer to an *interpolating implicit surface*, or nearby the points of $\mathcal{P}$, in this case we will refer to an *approximating implicit surface*. Moreover, as a genus for these expressions, we will refer to a *reconstructed implicit surface*.

Of course, the problem to reconstruct implicit surfaces from unorganized points is ill-posed, because there are an infinite number of implicit surfaces that are interpolating or approximating the unorganized points. In this thesis, we emphasize the reconstruction of implicit surfaces that are smooth, and we define the smoothness of the surface $\mathcal{S} = \{\boldsymbol{x} \in \mathbb{R}^3 | f(\boldsymbol{x}) = 0\}$ by the smoothness of the defining function f in terms of C^α *continuity*. Remember that a C^α function is a function with α continuous derivatives. In particular, C^0 denotes the space of continuous functions, C^1 denotes the space of continuously differentiable functions, and a C^∞ is a function that is differentiable for all degrees of differentiation that we call *infinitely smooth*, because neither the function nor its derivatives have discontinuities. We rate the ability to reconstruct smooth implicit surfaces by defining C^α functions with a high value for α as very important. This is, because implicit surfaces with defining functions that are only in C^0 are not smooth, that can be seen with the naked eye, and even smoother defining functions are desirable for illumination, as they are often based on the normals of the implicit surfaces that are calculated using the derivatives (see Equation (2)).

In order to allow the reconstruction and evaluation of implicit surfaces from a point set $\mathcal{P}$ consisting of a high number N of points, the reconstruction method should be efficient in terms

of required computational time and memory usage. Following Schaback [139], we consider a reconstruction method to be efficient, when it produces at most $O(N \log N)$ intermediate data at a computational cost bounded by $O(N \log N)$. Moreover, we consider the evaluation of implicit surfaces, i.e. the evaluation of the defining function f, to be efficient when it can be done in at most $O(\log N)$ operations.

This part of the thesis is organized as follows. After an extensive study of previous work in Chapter 1, we present the first new method that we call the partition of unity variational method in Chapter 2. Based on some fundamentals of this method, we present the second new method that we call the hierarchical partition of unity variational method in Chapter 3.

Chapter 1

Previous Work

1.1 Blobby Objects

To our knowledge, one of the first attempts to automatically reconstruct implicit surfaces from unorganized point sets in 3D has been investigated by Muraki [114]. The reconstructed implicit surface is described by successively blending several simple 3D base primitives together that are minimizing an energy function. As the base primitive, Blinn's *blobby model* [21] is used where the potential function $q_i(\boldsymbol{x})$ of the *blobby primitive* B_i for a 3D shape defined by the function f_i is described as

$$q_i(\boldsymbol{x}) = b_i e^{-a_i f_i(\boldsymbol{x})}. \tag{3}$$

Muraki et al. limit the base primitive to *blobby spheres*, radially symmetric shapes around a skeleton point $\boldsymbol{s}_i$ defined by

$$q_i(\boldsymbol{x}) = b_i e^{-a_i \|\boldsymbol{x},\boldsymbol{s}_i\|^2}, \tag{4}$$

which can be described by the parameters $(a_i, b_i, \boldsymbol{s}_i)$. Blending N base primitives together to obtain the defining function f is simply done by summing up the potential functions and defining a threshold τ

$$f(\boldsymbol{x}) = \tau - \sum_{i=1}^{N} b_i e^{-a_i f_i(\boldsymbol{x})}. \tag{5}$$

Muraki starts with a single primitive B_0 centered at the mass of the data at $\boldsymbol{s}_0$ with $a_0 = b_0 = \tau e$ which is put in a list. The reconstruction process iteratively selects and removes an arbitrary primitive B_i from the list, which is splitted into two new primitives $\mathsf{B}_i^{'}$ and $\mathsf{B}_i^{''}$ that are added to the list. The parameters of the new primitives $\mathsf{B}_i^{'}$ and $\mathsf{B}_i^{''}$ are calculated by minimizing an energy function. This energy function takes into account the measurement of the defining function f at the initial point set $\mathcal{P}$, the deviation of the gradients of the function f from the normals at the initial point set, and a term designed to localize the surface to the initial points. By continuing the iterative reconstruction process, the resulting defining function gradually comes to approximate the initial points since the number of primitives is increased, and the reconstruction is finished when the energy function

value falls below a threshold. This gradual refining procedure can also be considered as one of the first multiresolution approaches in reconstructing implicit surfaces, since intermediate results during the iterative process can be used as coarse surface approximations.

The outlined method of Muraki is a skeletal approach, since the center of each base primitive describes a point skeleton, and the radius of each base primitive defines a surface around its center. Unfortunately, reconstructing implicit surfaces using this fully automatic method is rather costly in terms of computational effort. Using the original implementation, Muraki reported for example, that calculating 243 primitives to describe a surface reconstructed from 2893 initial points "took a few days on a UNIX workstation (Stardent TITAN3000 2CPU)". To tackle this problem, Tsingar et al. [153] allowed to interactively specify the initial skeleton, which was further enhanced to an automated method by Bittar et al. [20].

However, we think, that even with the advances of computer resources and using the optimization methods, fitting blobby spheres is not adapted to reconstruct implicit surfaces from large unorganized point sets because of the required computational effort. Furthermore, the surfaces lack of control as they are based on squeleton points and hence not directly defined from the unorganized points. As a consequence, they have a slippery behavior when additional points are added to the unorganized point set.

1.2 Surface Reconstruction by Signed Distance Function Estimation

One of the most cited works about surface reconstruction from an unorganized point set $\mathcal{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_N\}$ is the work of Hoppe et al. [81]. A signed geometric distance function $d : \mathbb{R}^3 \to \mathbb{R}$ between points $\boldsymbol{x} \in \mathbb{R}^3$ and the unknown surface $\mathcal{S}$ is estimated, and the zero-set of the signed distance function $d(\boldsymbol{x}) = 0$ defines the reconstructed surface implicitly. The surface $\mathcal{S}$ is described by local linear approximations using the tangent planes of the initial points. The distance from an arbitrary point $\boldsymbol{x} \in \mathbb{R}^3$ to the surface $\mathcal{S}$ is the distance between $\boldsymbol{x}$ and the tangent plane associated to the closest initial point. In order to obtain the signed distance function d, this distance is multiplied by ± 1 depending on which side of the tangent plane the point $\boldsymbol{x}$ lies. Hence, there are two key ingredients of this procedure: first, estimating the tangent planes of the initial points, and second, consistently orientating the tangent planes so that the tangent planes of all pairs of "sufficiently close" points have the same orientation.

In the first step, the tangent plane of an initial point $\boldsymbol{p}_i \in \mathcal{P}$ is approximated by the least squares best fitting plane of the H-neighborhood $\mathcal{N}_i^{\{H\}}$, i.e. the H points of $\mathcal{P}$ being nearest to $\boldsymbol{p}_i$. In order to avoid bumps on the surface, the stronger the noise in the input data, the more points have to be considered. The tangent plane is passing through the centroid $\boldsymbol{o}_i$ of $\mathcal{N}_i^{\{H\}}$, and its normal is determined using principal component analysis: the normal is the unit eigenvector which is associated to the highest eigenvalue of the symmetric, positive definite covariance matrix $\boldsymbol{C}$ with

$$\boldsymbol{C} = \sum_{\boldsymbol{y} \in \mathcal{N}^{\{H\}}} (\boldsymbol{y} - \boldsymbol{o}_i) \times (\boldsymbol{y} - \boldsymbol{o}_i). \tag{6}$$

In the second step, the sign of the normal is modified so that all tangent planes are

consistently oriented, i.e. that the normals $\boldsymbol{n}_i$ and $\boldsymbol{n}_j$ associated to two geometrically close points $\boldsymbol{p}_i \in \mathcal{P}$ and $\boldsymbol{p}_j \in \mathcal{P}$ have a similar orientation. In the case of a smooth surface described by dense initial points $\mathcal{P}$, and Hoppe et al. assume that $\boldsymbol{n}_i \bullet \boldsymbol{n}_j \approx 1$. The problem can be modeled as an optimization problem using a graph with one node per tangent plane, and with an edge (i, j) if either $\boldsymbol{o}_i$ is in the H-neighborhood $\mathcal{N}_j^{\{H\}}$ of $\boldsymbol{o}_j$, or $\boldsymbol{o}_j$ is in the H-neighborhood $\mathcal{N}_i^{\{H\}}$ of $\boldsymbol{o}_i$, associated with a cost $\boldsymbol{n}_i \bullet \boldsymbol{n}_j$. The solution is given by selecting the orientations for the tangent planes that are maximizing the total cost of the graph. Since finding this solution is an NP complete problem, Hoppe et al. define the following heuristic. By assigning for each edge (i, j) a cost $1 - \mid \boldsymbol{n}_i \bullet \boldsymbol{n}_j \mid$, an initial tangent plane orientation is propagated by a depth-first order traversal of the *minimal spanning tree* of the resulting graph. This traversal order avoids ambiguous situations since orientations are propagated primarily in directions of low curvature.

The resulting implicit surface is described by the zero-set of the signed distance function d. Hoppe et al. demonstrated the power of their algorithm on various point sets from different sources, and it worked even on models with high topological complexity. However, the resulting surfaces only have C^0 continuity, and thus in practice, the method is not adapted to reconstruct smooth surfaces.

Nevertheless, Hoppe's method that estimates consistently oriented tangent planes is used to determine normals for the points of an unorganized point set in various reconstruction methods.

1.3 Moving Least Squares

Recently, Alexa et al. presented a purely local method to reconstruct implicit surfaces from an unorganized point set $\mathcal{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_N\}$ based on work from Levin [97, 98], and they call the resulting surface a *point set surface* [7, 6]. By defining a projection procedure that projects any point near the point set $\mathcal{P}$ onto the surface based on the method of moving least squares [96], the surface $\mathcal{S}$ is defined implicitly as all points that project onto themselves. Although the defining function f of the reconstructed implicit surface is not determined explicitly, it is implicitly defined by the projection procedure.

The projection procedure can be divided into two steps. First, for a point $\boldsymbol{p}^{near}$ near the surface, a local reference plane defined in Hessian normal form $\mathsf{H} = \{\boldsymbol{n} \bullet \boldsymbol{x} - d = 0, \boldsymbol{x} \in \mathbb{R}^3\}$ with $\boldsymbol{n} \in \mathbb{R}^3$ and $\|\boldsymbol{n}\| = 1$ is determined by minimizing a local weighted sum of squared distances of the points $\boldsymbol{p}_i$ to the plane H

$$\sum_{i=1}^{N} (\boldsymbol{n} \bullet \boldsymbol{p}_i - d)^2 \theta(\|\boldsymbol{p}_i - \boldsymbol{q}\|), \tag{7}$$

where $\boldsymbol{q}$ is the orthogonal projection of $\boldsymbol{p}^{near}$ on H, and θ is a radially symmetric, positive, and monotonically decreasing weight function.

In a second step, the local reference domain for $\boldsymbol{p}^{near}$ that is given by the orthonormal coordinate system on H with the origin $\boldsymbol{q}$ is used to compute a local bivariate degree d polynomial approximation $g \in \mathbb{R}^d \rightarrow \mathbb{R}$ of the surface in a neighborhood of $\boldsymbol{p}^{near}$. Let $\boldsymbol{q}_i$ be the projection of $\boldsymbol{p}_i$ onto H, then $\boldsymbol{p}_i$ has the height $h_i = \boldsymbol{n} \bullet (\boldsymbol{p}_i - \boldsymbol{q})$ over H, and the coefficients

of the polynomial approximation g are determined by minimizing the weighted least squares error

$$\sum_{i=1}^{N}(g(x_i,y_i)-h_i)^2\theta(\|\boldsymbol{p}_i-\boldsymbol{q}\|), \tag{8}$$

where $[x_i,y_i]^T$ is the representation of boldsymbol $\boldsymbol{q}$ in the local reference domain. Alexa et al. propose to use cubic or quartic polynomials for g, since they experienced oscillatory artefacts with polynomial approximations of higher degree . The projection $\Psi:\mathbb{R}^3\rightarrow\mathbb{R}^3$ of $\boldsymbol{p}^{near}$ onto the surface is defined by the polynomial value in the origin $\Psi(\boldsymbol{p}^{near})=\boldsymbol{q}+g(0,0)\boldsymbol{n}$. Note that the property $\Psi(\Psi(\boldsymbol{p}^{near}))=\Psi(\boldsymbol{p}^{near})$ is extremely important, since this means that a point on the surface projects onto itself.

For θ, the same weight function of Equation (7) and Equation (8), Alexa et al. use the Gaussian function $\theta(x)=e^{-x^2/h^2}$ that was already proposed by Levin [97, 98]. The parameter h called feature size influences the smoothness of the reconstructed surface and is usually set to the anticipated spacing between neighboring points. Higher values for h result in a more global surface approximation by smoothing out sharp or oscillatory features of size smaller than h. For all weight functions θ with $\theta\in C^\infty$, Levin believes that the resulting implicit surface is infinitely smooth [97, 98], i.e. that is also has C^∞ continuity.

Note that the local reference plane changes at every point, and thus the implicitly defined point set surface is not piecewise parametrically defined. However, the computation of the projection involves some expensive operations. Finding the minimum of Equation (7) is a non-linear minimization problem that has to be solved iteratively, and finding the minimum of Equation (8) is a standard linear least squares problem where the coefficients of the polynomial g are determined by solving a system of linear equations. Hence, the required computational effort makes most practical shape operations with point set surfaces expensive.

1.4 Implicit Surfaces defined by Radial Basis Functions

1.4.1 Variational Techniques and Radial Basis Functions

Reconstructing implicit surfaces as the zero-set of a function $f:\mathbb{R}^d\rightarrow\mathbb{R}$ from given point sets $\mathcal{P}$ on the surface can be formulated as a general scattered data interpolation problem: for a given finite set of N distinct points in $\mathbb{R}^d$ with associated function values h_i, find a function $f:\mathbb{R}^d\rightarrow\mathbb{R}$, that is satisfying the constraints

$$f(\boldsymbol{p}_i)=h_i \qquad i=1,...,N. \tag{9}$$

The problem (9) is clearly ill-posed, since it has an infinite number of solutions. In order to choose one particular solution, some more a priori knowledge about the function to be reconstructed is required. The most common form of a priori knowledge consists in assuming smoothness, i.e., that two similar inputs $\boldsymbol{x}_0\approx\boldsymbol{x}_1$ create two similar outputs $f(\boldsymbol{x}_0)\approx f(\boldsymbol{x}_1)$, and in the special case of reconstructing implicit surfaces, smooth surfaces with a high C^α continuity should be obtained. Moreover, it has to be determined whether the function to be reconstructed should fulfill the constraints (9) exactly, we refer then to an *interpolation*

problem, or approximately, where we refer to an *approximation problem*.

By underlying regularization theory, the solution of the ill-posed problem can be obtained from a *variational principle* containing both data closeness and smoothness information. Since we look for a function that is simultaneously close to the data and also smooth, solving the reconstruction problem by a *variational technique* consists in finding a function f that is minimizing the variational functional $\boldsymbol{V}$ of Equation (10).

$$\boldsymbol{V}[f] = \sum_{i=1}^{N} (f(\boldsymbol{p}_i) - h_i)^2 + \lambda \boldsymbol{E}[f] \tag{10}$$

The first term is enforcing closeness to the data, and the second term is enforcing smoothness. The positive number λ is usually called the *regularization parameter* to control the trade-off between these two terms, and with $\lambda = 0$ the function indeed interpolates the scattered data. Moreover, $\boldsymbol{E}$ is called *stabilizer* or *smoothness functional*.

In this thesis, we define the smoothness functional by Equation (11) that considers a function to be smoother when it oscillates less. We denote the Fourier transformation with $\sim$, and $\tilde{\phi}$ is some positive basic function that falls off to zero as $\|\boldsymbol{s}\| \to \infty$.

$$\boldsymbol{E}[f] = \int_{\mathbb{R}^d} d\boldsymbol{s} \frac{|\, \tilde{f}(\boldsymbol{s}) \,|^2}{\tilde{\phi}(\boldsymbol{s})} \tag{11}$$

It can be shown [60], that the function that minimizes the functional (11) has the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \omega_i \phi(\boldsymbol{x} - \boldsymbol{p}_i) + \sum_{\alpha=1}^{Q} \pi_\alpha p_\alpha(\boldsymbol{x}) \tag{12}$$

where $\{p_\alpha\}_{\alpha=1}^{Q}$ is a basis in the m-dimensional null space containing all real-valued polynomials in d variables and of order at most m, hence $Q = \binom{m-1+d}{d}$, and we require $N \geq Q$ as well as the side conditions ensuring orthogonality

$$\sum_{i}^{N} \lambda_i = \sum_{i}^{N} \lambda_i \pi_1 = \ldots = \sum_{i}^{N} \lambda_i \pi_Q = 0. \tag{13}$$

Moreover, ϕ is a *basic function*, and ω_i and π_α are the *coefficients* that satisfy the following linear system:

$$
\begin{aligned}
\boldsymbol{A}\boldsymbol{x} &= \boldsymbol{b} \\
\boldsymbol{A} &= \begin{bmatrix} \boldsymbol{\Phi} & \boldsymbol{P}^T \\ \boldsymbol{P} & \boldsymbol{0} \end{bmatrix} \\
\boldsymbol{\Phi} &= \left[\phi(||\boldsymbol{p}_i, \boldsymbol{p}_j||)\right] \qquad \text{where } i = 1 \ldots N, \qquad j = 1 \ldots N \\
\boldsymbol{P} &= \left[p_\alpha(\boldsymbol{p_i})\right] \qquad \text{where } i = 1 \ldots N, \qquad \alpha = 1 \ldots Q \\
\boldsymbol{x} &= [\omega_1, \omega_2, \ldots, \omega_N, \pi_1, \pi_2, \ldots, \pi_Q]^T \\
\boldsymbol{b} &= \left[h_1, h_2, \ldots, h_N, \underbrace{0, 0, \ldots, 0}_{Q \text{ times}}\right]^T
\end{aligned}
\tag{14}
$$

A sufficient condition for the basic function ϕ, under which the linear system (15) has a unique solution for any choice of $\boldsymbol{p}_i$ and h_i, is conditionally positive definiteness. In this thesis, we will limit ϕ to radially symmetric basic functions, i.e. $\phi(\boldsymbol{x}) = \phi(\boldsymbol{R}\boldsymbol{x})$ for any rotation matrix $\boldsymbol{R}$. This choice reflects the a priori assumption, that there are no privileged directions and thus that all constraint locations $\boldsymbol{p}_i$ have the same relevance.

By using radially symmetric basic functions, the Equation (12) is commonly referred to *radial basis functions (RBF)*, and Franke [56] identified in an extensive survey RBFs as one of the most accurate and stable functions to solve the scattered data interpolation problem that produces surfaces, that are "usually pleasing and very smooth".

Note that in the radial basis function jargon, the constraint locations $\boldsymbol{p}_i$ are traditionally referred as *centers*, since the basic functions are symmetric around them.

The most commonly known classes of basic functions ϕ in dimension d of order m are multivariate splines and have been identified by Duchon [53].

$$
\phi_{m,d}(\boldsymbol{x}) = \begin{cases} \|\boldsymbol{x}\|^{2m-d} \ln \|\boldsymbol{x}\| & \text{for } d \text{ even} \\ \|\boldsymbol{x}\|^{2m-d} & \text{for } d \text{ odd} \end{cases}
\tag{15}
$$

In two dimensions, for $m = 2$, this yields to the so-called thin-plate spline

$$
\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|^2 \ln \|\boldsymbol{x}\|,
\tag{16}
$$

inspired by the following metaphor: by taking a thin sheet of metal, laying it horizontally and bending it in a way that it touches the tips of the vertical poles at height h_i at the positions $\boldsymbol{p}_i$, the metal plate resists bending so that it smoothly changes its height between the positions $\boldsymbol{p}_i$. This springy resistance is mimicked by the smoothness measure (11) to minimize, and in two dimensions the smoothness functional $\boldsymbol{E}$ can be considered as

$$
\boldsymbol{E}[f] = \int_\omega f_{xx}^2(\boldsymbol{x}) + f_{xy}^2(\boldsymbol{x}) + f_{yy}^2(\boldsymbol{x})\, d\boldsymbol{x}.
\tag{17}
$$

In three dimensions ($d = 3$), the corresponding smoothness functional $\boldsymbol{E}$ to minimize is

$$\boldsymbol{E}[f] \quad = \quad \int_\omega f_{xx}^2(\boldsymbol{x}) + f_{yy}^2(\boldsymbol{x}) + f_{zz}^2(\boldsymbol{x}) + 2f_{xy}^2(\boldsymbol{x}) + 2f_{xz}^2(\boldsymbol{x}) + 2f_{yz}^2(\boldsymbol{x})\, d\boldsymbol{x}, \tag{18}$$

and for the order $m = 2$, the Equation (15) yields the bivariate spline (*biharmonic basic function*) with C^0 continuity,

$$\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|, \tag{19}$$

and for order $m = 3$ the trivariate spline (*trivariate basic function*) with C^2 continuity:

$$\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|^3, \tag{20}$$

Table 1 summarizes classical and useful choices for the basic function ϕ, that are either positive definite or conditionally positive definite of order m. In contrast to using positive definite basic functions for the RBFs of Equation (12) where no polynomial $\sum_{\alpha=1}^{Q} \pi_\alpha p_\alpha(\boldsymbol{x})$ is required, conditionally positive definite basic functions of order m require a polynomial with a degree of at least m.

Gaussian	$\phi(\boldsymbol{x}) = e^{-\delta\|\boldsymbol{x}\|^2}$	positive definite	
multiquadric	$\phi(\boldsymbol{x}) = \sqrt{\|\boldsymbol{x}\|^2 + c^2}$	conditionally positive definite	order 1
inverse multiquadric	$\phi(\boldsymbol{x}) = \frac{1}{\sqrt{\|\boldsymbol{x}\|^2+c^2}}$	positive definite	
multivariate splines	$\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|^{2m+1}$	conditionally positive definite	order m
multivariate splines	$\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|^{2m} \ln \|\boldsymbol{x}\|$	conditionally positive definite	order m

Table 1: Classical and useful choices for the basic function ϕ.

More recent developments [168, 163] have provided *compactly supported* radial basis functions, e.g. Wendland [163] has derived positive definite compactly supported basic functions ϕ with minimum degree for a given continuity C^α of the resulting radial basis function (Equation (12)). See Table 2 for some examples of Wendland's basic functions in three dimensions.

C^0	$\phi(\boldsymbol{x}) = (1 - \|\boldsymbol{x}\|)_+^2$	positive definite
C^2	$\phi(\boldsymbol{x}) = (1 - \|\boldsymbol{x}\|)_+^4 (4\|\boldsymbol{x}\| + 1)$	positive definite
C^4	$\phi(\boldsymbol{x}) = (1 - \|\boldsymbol{x}\|)_+^6 (35\|\boldsymbol{x}\|^2 + 18\|\boldsymbol{x}\| + 3)$	positive definite
C^6	$\phi(\boldsymbol{x}) = (1 - \|\boldsymbol{x}\|)_+^8 (32\|\boldsymbol{x}\|^3 + 25\|\boldsymbol{x}\|^2 + 8\|\boldsymbol{x}\| + 1)$	positive definite

Table 2: Wendland's compactly supported basic functions ϕ of minimal degree for a given continuity C^α in three dimensions.

In the remainder of this subsection, we will discuss some previous work using variational techniques to reconstruct surfaces from unorganized point set $\mathcal{P}$, and we will divide it into four categories. The first category are "naive" reconstruction methods using radial basis functions with global support. The second category are methods allowing a higher number of constraints using greedy fast evaluation techniques of radial basis functions with global support. The third category are methods using compactly supported radial basis functions, and thus also allow a higher number of constraints. Finally, the fourth category are methods using multi-level reconstruction methods.

1.4.2 Global Support

The pioneering work to reconstruct implicit surfaces from given point sets using variational techniques can be attributed to Savchenko et al. [138]. The implicit surface $f : \mathbb{R}^d \to \mathbb{R}$ is reconstructed by introducing a *carrier solid* with a defining function $f_c : \mathbb{R}^d \to \mathbb{R}$, which, in the simplest case, can be a sphere with radius r defined by $f_c(\boldsymbol{x}) = r^2 - \|\boldsymbol{x}\|^2$. First, the values r_i of the carrier function f_c are calculated at all given points $\boldsymbol{p}_i$ as can be seen in Equation (21).

$$r_i = f_c(\boldsymbol{p}_i) \qquad i = 1, ..., N \tag{21}$$

The reconstruction problem is now shifted to finding a volume spline function $u : \mathbb{R}^d \to \mathbb{R}$ that interpolates all values r_i of the function f_c in the points $\boldsymbol{p}_i$ as can be seen in Equation (22).

$$u(\boldsymbol{p}_i) = r_i \qquad i = 1, ..., N \tag{22}$$

Then, the zero set of the algebraic difference between u and f_c describes the reconstructed implicit surface f,

$$f(\boldsymbol{x}) = u(\boldsymbol{x}) - f_c(\boldsymbol{x}). \tag{23}$$

Savchenko et al. describe the interpolating spline function u as a function of the class defined in (12) being a linear combination of Duchon's basic functions of (15). We find

$$u(x) = \sum_{i=1}^{N} \omega_i \phi(\boldsymbol{x} - \boldsymbol{p}_i) + \sum_{\alpha=1}^{Q} \pi_\alpha p_\alpha(\boldsymbol{x}), \tag{24}$$

and hence, the constraints of Equation (22) and Equation (13) together with the Equation (24) lead to the linear system shown in Equation (26):

$$\begin{aligned}
\boldsymbol{A}\boldsymbol{x} &= \boldsymbol{b} \\
\boldsymbol{A} &= \begin{bmatrix} \boldsymbol{\Phi} & \boldsymbol{P}^T \\ \boldsymbol{P} & \boldsymbol{0} \end{bmatrix} \\
\boldsymbol{\Phi} &= \left[\phi(||\boldsymbol{p}_i, \boldsymbol{p}_j||)\right] \qquad \text{where } i = 1 \ldots N, \qquad j = 1 \ldots N \\
\boldsymbol{P} &= [p_\alpha(\boldsymbol{p_i})] \qquad \text{where } i = 1 \ldots N, \qquad \alpha = 1 \ldots Q \\
\boldsymbol{x} &= [\omega_1, \omega_2, \ldots, \omega_N, \pi_1, \pi_2, \ldots, \pi_Q]^T \\
\boldsymbol{b} &= \begin{bmatrix} r_1, r_2, \ldots, r_N, \underbrace{0, 0, \ldots, 0}_{Q \text{ times}} \end{bmatrix}^T
\end{aligned} \tag{25}$$

Savchenko et al. use the triharmonic basic function $\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|^3$ that has global support since $\phi(\boldsymbol{x}) > 0 \quad \forall \boldsymbol{x} \neq \boldsymbol{0}$, and thus the linear system (26) is dense.

The authors proposed to use the Householder method [62] to solve the linear system and have shown, that the runtime of the algorithm decreased linearly proportional with the

number of processors used in a parallel computing network. However, the algorithm is only applicable to reconstruct implicit surfaces from a restrained number of unorganized points, say several thousands, due to the $O(N^3)$ behavior. Furthermore, the evaluation of Equation (23) is in $O(N)$, which can get rather costly. Consequently, the surface can neither be efficiently reconstructed nor efficiently evaluated. Another drawback of this method is, that besides the choice of the volume spline function, the carrier solid to use has to be defined. The choice of the carrier solid has a significant impact on the shape of the reconstructed surface since it is isomorphic to the carrier solid, and it is still an open problem how to determine an adapted carrier solid.

We can state that the definition of the reconstructed implicit surface as a difference of a carrier solid and a volume spline in (23) avoids that the constructed linear system (15) from the interpolation constraints (9) with $h_i = 0$ has the trivial solution. Another way to avoid the trivial solution has been introduced by Turk and O'Brien [156] in their work called *variational implicit surfaces*. In addition to the reconstruction constraints (9) where $h_i = 0$, K so-called *off-surface constraints*

$$f(\boldsymbol{p}_{N+j}) = h_{N+j} \neq 0, \qquad j = 1, ..., K \tag{26}$$

are introduced at the *off-surface points* $\boldsymbol{p}_{N+j}$ allowing to reconstruct the implicit surface directly without defining a carrier solid.

Turk and O'Brien discuss the usage of three different kinds of off-surface constraints, based on the common convention that $f(\boldsymbol{x}) > 0$ inside and $f(\boldsymbol{x}) < 0$ outside the surface. First, *positive interior constraints* can be introduced by specifying positive values for h_i at one or more off-surface points $\boldsymbol{p}_i'$ that are to be in the interior of the surface. Second, *negative exterior constraints* can be introduced by specifying negative values for h_i at off-surface points $\boldsymbol{p}_i''$ on the exterior of the shape to be reconstructed, for example at selected positions on a sphere around the surface to be reconstructed. Third, *normal constraints* can be used when there is knowledge about the normals of the surface at the reconstruction constraints, that is either directly obtained during data acquisition or estimated from neighboring points [81]. *Normal off-surface points* can be placed at positions $\boldsymbol{p}_i' = \boldsymbol{p}_i + \kappa \boldsymbol{n}_i$ computed starting from the initial points $\boldsymbol{p}_i$ and moving them along its normal vector $\boldsymbol{n}_i$. Turk and O'Brien add one off-surface constraint at the exterior of the surface ($\kappa_i > 0, h_i = -1$) for every initial point.

After specifying the K off-surface constraints, the implicit surface can be described as a function of class (12). For an interpolation in $d = 3$ dimensions, Turk and O'Brien use the triharmonic basic function $\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|^3$ with a polynomial of degree one accounting for the linear and constant portions of f, i.e. $Q = 4$ and $\sum_{\alpha=1}^{4} \pi_\alpha p_\alpha(\boldsymbol{x}) = [\pi_1, \pi_2, \pi_3] \bullet \boldsymbol{x} + \pi_4$, and the radial basis function is

$$f(\boldsymbol{x}) = \sum_{i=1}^{N+K} w_i \phi(\boldsymbol{x} - \boldsymbol{p}_i) + [\pi_1, \pi_2, \pi_3] \bullet \boldsymbol{x} + \pi_4. \tag{27}$$

The resulting linear system that can be written as

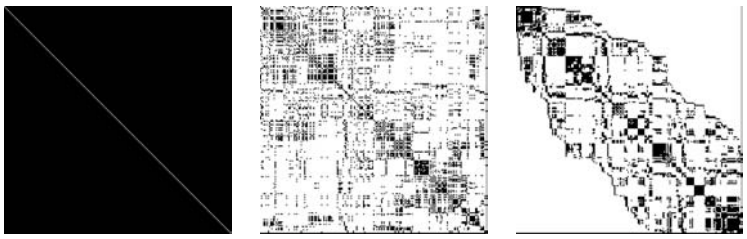
$$
\begin{aligned}
\boldsymbol{A}\boldsymbol{x} &= \boldsymbol{b} \qquad (28)\\
\boldsymbol{A} &= \begin{bmatrix} \boldsymbol{\Phi} & \boldsymbol{P}^T & 1 \\ \boldsymbol{P} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}\\
\boldsymbol{\Phi} &= \left[\phi(||\boldsymbol{p}_i, \boldsymbol{p}_j||)\right] \qquad \text{where } i = 1 \ldots N+K, \qquad j = 1 \ldots N+K\\
\boldsymbol{P} &= [\boldsymbol{p}_i] \qquad \text{where } i = 1 \ldots N+K\\
\boldsymbol{x} &= [\omega_1, \omega_2, \ldots, \omega_{N+K}, \pi_1, \pi_2, \pi_3, \pi_4]^T\\
\boldsymbol{b} &= [h_1, h_2, \ldots, h_{N+K}, 0, 0, 0, 0]^T
\end{aligned}
$$

Again, the linear system can be solved using direct solution techniques, and Turk and O'Brien propose to use *LU-decomposition* or *Singular Value Decomposition (SVD)*. However, setting up the linear system and the memory requirement are in $O(N^2)$, and with the cost of $O(N^3)$ to find a solution for the linear system, the computational effort is still in $O(N^3)$. Hence the number of constraints is still limited to, say, several thousands, and the method of Turk and O'Brien cannot be considered as efficient according to the definition of Schaback [139].

1.4.3 Fast Evaluation Techniques

The latter two methods fail to reconstruct implicit surfaces from a large number of constraints, since the computational cost of $O(N^3)$ to solve the linear system becomes prohibitively expensive, and the huge amount of required memory in $O(N^2)$ rapidly grows above available memory amounts. Furthermore, evaluating the defining function is in $O(N)$, and this becomes prohibitively expensive since the evaluation step usually has to be performed many times.

The pioneering work to reconstruct implicit surfaces using radial basis functions with global support from a high number of constraints can be attributed to Carr et al. [33]. They overcome the previously discussed computational cost and storage limitations by using a greedy fast evaluation technique of radial basis functions that allows fast and storage-efficient computation of matrix-vector products [17, 15], hence making iterative solutions very attractive as solvers for the linear system. The greedy fast evaluation technique is based on the *Fast Multipole Method* of Greengard and Rohklin [66] that takes benefit of the fact, that when evaluating the radial basis function f at an argument $\boldsymbol{x}$, infinite precision is not required and the evaluation can be approximated by dividing into far- and near-field expansions for a given argument $\boldsymbol{x}$. On the one hand, all radial basis function terms in the near-field, i.e. where the centers $\boldsymbol{p}_i$ are close to the argument $\boldsymbol{x}$, are computed directly and explicitly. On the other hand, many radial basis function terms of the far-field whose centers $\boldsymbol{p}_i$ are close to each other (but far away from the argument $\boldsymbol{x}$) are approximated simultaneously by a *Taylor series* of truncated *Laurent expansions*. The accuracy, i.e. the length of the truncated Laurent expansion, can be preset, for example to a multiple of the machine precision of the computer in use. In order to identify the near and far fields of a given argument efficiently, the data has to be structured hierarchically. See the introductory short course [14] for more details of the Fast Multipole Method.



(a) Dense matrix. (b) Sparse matrix. (c) Band-diagonal matrix.

Figure 1: Involved matrices in the linear system (white for zero-value elements).

Using the greedy evaluation technique, the evaluation cost drops from $O(N)$ to $O(1)$ after a $O(N \log N)$ setup, and calculating a matrix-vector product drops from $O(N^2)$ to $O(N)$. This matrix-vector product is involved when solving the linear system iteratively, either by pre-conditioned conjugate gradient methods [17], preconditioned GMRES iterations [13], or domain decomposition methods [16], and the computational cost to solve the linear system drops from $O(N^3)$ to $O(N \log N)$. Furthermore, since the involved matrix of the linear system never has to be calculated explicitly, storage requirements are also greatly reduced to only $O(N)$ compared to $O(N^2)$ before. Consequently, the method of Carr et al. can be considered as efficient.

Unfortunately, the development of the far and near fields is very complex to implement, and its deviation has to be done for every radial basis function separately. Carr et al. have developed the far fields for the biharmonic and triharmonic basic functions, but their implementation is only commercially available [9].

Besides the gain in computational and storage complexities, the reconstruction of the implicit surfaces is quite similar to Turk and O'Brien. Carr et al. [33] also define off-surface constraints, however, they propose to add two new constraints at normal off-surface points on both sides of the surface ($\kappa > 0, h_i = -1$ or $\kappa < 0, h_i = 1$) for a subset of the initial points according to Equation (26). Experimental results have shown, that their scale-independent global reconstruction method behaves well on a variety of point sets coming from different sources such as polygonal meshes, LIDAR data, or range scanner data.

1.4.4 Compact Support

Morse et al. [113] propose another way to allow a higher number of constraints when reconstructing implicit surfaces by using Wendland's compactly supported radial basis functions of minimal degree [163]. In contrast to the methods presented so far, the resulting linear system is sparse since $\phi(\boldsymbol{x}) = 0$ for $\|\boldsymbol{x}\| \geq \sigma$ for a given support radius σ. Figure 1(b) shows an example of the *sparse matrix* involved in the linear system compared to a dense matrix generated with globally supported radial basis functions (Figure 1(a)).

Similar to Turk and O'Brien [156] and Carr [33], Morse et al. use normal constraints in order to avoid the trivial solution of the linear system.

The sparse linear system that is generated has a nice impact on the computational complexity: by using hierarchical data structures for point queries like k-d trees, the linear system can be built up in $O(N \log N)$ time, solved in the range $O(N^{1.2})$ to $O(N^{1.5})$ [113], and the evaluation drops to $O(\log N)$. Concerning storage complexity, by using sparse-matrix representations, only $O(N)$ storage is required.

The major drawback of this method is, that the support radius σ has to be chosen globally, hence an additional parameter compared to globally supported radial basis functions is introduced. Choosing a large radius σ drops the performance, and may result in the same computational and storage complexities as by using globally supported radial basis functions. On the other hand, choosing a small radius σ may result in a too local reconstruction, possibly with the generation of holes in the surface. For these reasons, compactly supported radial basis functions should only be used to reconstruct implicit surfaces from quasi-uniformly distributed point sets.

Another disadvantage is, that the reconstructed implicit surface f is not the only set of zero-valued points in space, but there are extra zero-sets. Hence, *point membership classifications*, which are often used in *constructive solid geometry (CSG)* are not directly applicable to this kind of surfaces, as the common convention of the implicit surface to be positive inside and negative outside is not met.

The work of Morse et al. was further improved by Kojekine et al. [90] by organizing the involved sparse matrix into a *band-diagonal sparse matrix* (see Figure 1(c)) in an efficient manner using an *octree* data structure. The resulting linear system can be solved more efficiently using a combination of block Gaussian solution and Cholesky decomposition [59]. Furthermore, Kojekine et al. use a carrier solid instead of normal constraints, which is reducing the number of constraints and thus further improving efficiency. But Kojekine et al. state also, that compactly supported radial basis functions are only suitable for "moderately sized" 3D point sets, they did not report reconstruction results for point sets consisting of more than one hundred thousand points.

1.4.5 Multi-level methods

Compactly supported radial basis functions are also used by Ohtake et al. [119] to reconstruct implicit surfaces from larger point sets $\mathcal{P}$ in a multiresolution manner. Basically, a hierarchy of point sets $\mathcal{P}^{[1]}, \mathcal{P}^{[2]}, \ldots, \mathcal{P}^{[L]}$ is computed recursively, and the point set $\mathcal{P}^{[l+1]}$ at level $l+1$ is interpolated by offsetting the interpolation function used to interpolate the point set $\mathcal{P}^{[l]}$ at the previous level l.

The point set $\mathcal{P}^{[1]}$ at the coarsest level of subdivision is constructed by fitting the initial point set $\mathcal{P}$ into a bounding box, subdividing the bounding box into eight equal octant cells, and computing the eight centroids with averaged unit normals of the points of $\mathcal{P}$ which are contained in the corresponding cell. By continuing this process recursively, the point set $\mathcal{P}^{[l]}$ at level l of the hierarchy contains 8^l points, and the recursion is stopped when each cell does not contain more than eight points.

After the hierarchy is constructed, a base function $f^0(\boldsymbol{x}) = -1$ is defined, and the set of interpolation functions $f^{[l]}$ interpolating $\mathcal{P}^{[l]}$ is defined in a coarse-to-fine way by offsetting $f^{[l-1]}$ with the offsetting function $o^{[l]}$,

$$f^{[l]}(\boldsymbol{x}) = f^{[l-1]}(\boldsymbol{x}) + o^{[l]}(\boldsymbol{x}) \qquad l = 1, \ldots, L. \tag{29}$$

In order to reconstruct the implicit surface at level l that is interpolating $\mathcal{P}^{[l]}$,

$$f^{[l]}(\boldsymbol{x}) = 0, \tag{30}$$

the offsetting function $o^{[l]}$ has to be determined. Ohtake et al. define $o^{[l]}$ as a linear combination of Wendland's compactly supported radial basis function $\phi(\boldsymbol{x}) = (1 - \frac{\|\boldsymbol{x}\|}{\sigma}^4_+)(4\frac{\|\boldsymbol{x}\|}{\sigma} + 1)$ with support radius σ

$$o^{[l]}(\boldsymbol{x}) = \sum_{\boldsymbol{p}_i^{[l]} \in \mathcal{P}^{[l]}} [g_i^{[l]}(\boldsymbol{x}) + \omega_i^{[l]}] \phi_{\sigma^{[l]}}(\|\boldsymbol{x} - \boldsymbol{p}_i^{[l]}\|). \tag{31}$$

Note, that a quadric function $g_i^{[l]}$ is determined by least-squares minimization that is approximating the shape of $\mathcal{P}^{[l]}$ in a small vicinity of $\boldsymbol{p}_i^{[l]}$. As the Equation (31) has the form of the class of functions of Equation (12) introduced above, the coefficients $\omega_i^{[l]}$ can be calculated by solving the corresponding linear system with the constraints $o^{[l]}(\boldsymbol{x}) = -f^{[l-1]}(\boldsymbol{x})$ obtained by inserting (30) into (29). Ohtake et al. use the preconditioned biconjugate gradient method to solve the linear system, and since the function $g_i^{[l]}$ can be considered as a local carrier solid, no extra off-surface constraints need to be introduced. The support radius $\sigma^{[1]}$ is defined as 75 % of the length of the $\mathcal{P}$'s bounding box diagonal, and $\sigma^{[l]}$ is defined recursively by $\sigma^{[l+1]} = \frac{\sigma^{[l]}}{2}$.

The multiresolution approach of Ohtake et al. overcomes several limitations encountered before when compactly supported radial basis functions were used. First, the number of constraints can be significantly higher, i.e. Ohtake et al. reconstruct implicit surfaces from several hundreds of thousands of input points. Second, as the support radius varies throughout the different levels of the hierarchy, highly non-uniformly sampled point sets can be reconstructed allowing to fill larger holes and repair incomplete data. On the other hand, similar to the drawback of traditional reconstruction methods using compactly supported radial basis functions, extra zero-sets are generated when the surface described by the sampled point set has thin parts. Furthermore, since the reconstruction of the finest levels of the hierarchy involves the solution of sparse linear systems with a high number of constraints, the number of input points is still limited.

1.5 Multi-level Partition of Unity Implicits

In order to reconstruct implicit surfaces from a very large number of unorganized points with associated normals, Ohtake et al. [118] use *multi-level partition of unity implicits*, i.e. an implicit surface with a global defining function that is reconstructed by partition of unity [12] blending of local shape approximations. Basically, the reconstruction process starts by rescaling the point set $\mathcal{P}$ into a bounding cube and creating an octree-based subdivision of this cube. At each cell of the octree, a local shape approximation is calculated, and while the local shape approximation is not accurate enough, the cell is subdivided recursively until a certain accuracy is achieved. Note that in this way the depth of the octree is rather dependent

on the complexity of the reconstructed shape than on the number of points.

Depending on the number of points in the given cell and the distribution of their associated normals, one of three types of local shape approximations is chosen. The first type of local shape approximation is used to approximate larger parts of the surface, and the second one is used to approximate a local smooth patch. Both types are identified, when there are more than T_{min} points in a sphere defined by the cell's center and a radius being 75 % the length of the cell's main diagonal. When the deviation of the normal direction of the points to their average normal direction is more than $\frac{\pi}{2}$, the first type, i.e. a large part of the surface, is approximated by a 3D quadratic surface by using a least-squares fitting procedure to find the parameters. Otherwise, when the normal deviation is less than $\frac{\pi}{2}$, the second type, i.e. a local smooth patch, is approximated by a bivariate quadratic polynomial by again using a least-squares fitting procedure.

The third type of local shape approximations is used to reconstruct sharp features, and it is identified when there are less than T_{min} points in the ball associated to the given cell. Furthermore, some edge and corner detection criterions have to be matched according to Kobbelt et al. [89], otherwise, a bivariate quadratic polynomial is fitted as in the second type of local shape approximations. But in the case, when an edge or a corner is detected, a piecewise quadratic function is fitted to the points.

The three types of local shape approximations are blended together in slightly overlapping domains using the partition of unity method, that will be discussed in more detail in Section 2.2, as it is also part of our new reconstruction methods. Since all local shape approximations are done by quadratic functions, multi-level partition of unity implicits can also be understood as piecewise defined algebraic surfaces.

Multi-level partition of unity implicits are a powerful tool since implicit surfaces can be reconstructed from a large number of input points as well as from incomplete data, and they are one of the only methods that enables to reconstruct implicit surfaces with sharp features.

1.6 Other Work

In this chapter, we presented an extensive study of previous work reconstructing implicit surfaces from unorganized points. Nevertheless, there are several other techniques to reconstruct implicit surfaces from unorganized points that have been developed recently, but it is out of the scope of this thesis to discuss them all. For example, *adaptively sampled distance fields* [57] reconstruct an implicitly defined surface from a distance function that has to be given on a regular grid in space, thus the surface detail is limited by the fixed grid resolution. As another example, *level set methods* deform an initial surface following the gradient flow of an energy functional, and a minimal surface like model with partial differential equations (PDE) is used [172, 171].

The presented previous work reveals the requirement to define other methods to reconstruct implicit surfaces from unorganized point sets that overcome the stated drawbacks of the respective existing methods. In the following two chapters, we present two new methods that overcome, at least partly, these drawbacks.

Chapter 2

The Partition of Unity Variational Method

2.1 Overview

In this Chapter, we present a new method to reconstruct an implicit surface from a large number of unorganized points that we call the *partition of unity variational method*. This purely local method was inspired by a fast evaluation technique of radial basis functions based on the partition of unity method [164, 165].

After presenting the partition of unity method in the following section, we present our new method that can be decomposed into three principal steps. First, the global domain of interest containing the unorganized point set is decomposed into smaller local subdomains, based on regular or adaptive domain decomposition methods (Section 2.3). Second, the reconstruction problems of the local subdomains are solved using variational techniques with radial basis functions (Section 2.4). Third, the solution of the global reconstruction problem is obtained by sticking the solutions of the local reconstructions problems together by using weighting functions defined by the partition of unity method (Section 2.5). Using this method, the complexity of the surface reconstruction and the evaluation of the defining function is reduced significantly compared to straightforward methods as we will see in Section 2.6, but it introduces some new parameters that define the locality of the method. We will present the results of a study that we have conducted in order to determine the impact of these parameters (Section 2.7). Finally, we discuss an illustrative example (Section 2.8) and present some results of our reconstruction method showing its robustness and scalability (Section 2.9) before we conclude in Section 2.10.

2.2 The Partition of Unity Method

The main idea of the partition of unity method is to divide the global domain of interest into smaller domains where the problem can be solved locally. More precisely, the global problem is decomposed into several smaller local problems, and their local solutions are combined together by weighting functions that act as smooth blending functions in order to obtain the global solution.

Consider a global domain $\Omega \subseteq \mathbb{R}^d$ and divide it into M overlapping subdomains $\mathcal{D} = \{\Omega_i\}_{i=1}^M$ with $\Omega \subseteq \bigcup_i \Omega_i$. On this set of subdomains $\mathcal{D}$, we construct a partition of unity, i.e. a set of non-negative *normalized weighting functions* $\mathcal{W} = \{w_i\}_{i=1}^M$ with limited support $\text{supp}(w_i) \subseteq \Omega_i$, i.e. $\forall \boldsymbol{x} \notin \Omega_i \Rightarrow w_i = 0$, and with $\sum_{i=1}^M w_i(\boldsymbol{x}) = 1$ at all points $\boldsymbol{x} \in \Omega$.

By applying the partition of unity method to reconstruct implicit surfaces, for each Ω_i, the point set $\mathcal{P}_i = \{\boldsymbol{p} \in \mathcal{P} | \boldsymbol{p} \in \Omega_i\}$ is collected, and a local defining function f_i for $\mathcal{P}_i$ is computed. The global defining function f is then defined as a sum of the local functions f_i multiplied by the corresponding normalized weighting functions w_i

$$f(\boldsymbol{x}) = \sum_{i=1}^{M} f_i(\boldsymbol{x}) w_i(\boldsymbol{x}). \tag{32}$$

The condition $\sum w_i = 1$ is obtained from any other set of weighting functions $\{\Omega_i\}_{i=1}^M \hat{w}_i$ by a normalization procedure

$$w_i(\boldsymbol{x}) \quad = \quad \frac{\hat{w}_i(\boldsymbol{x})}{\sum_j \hat{w}_j(\boldsymbol{x})}. \tag{33}$$

Any function $\hat{w}_i$ is appropriated, but to guarantee the continuity of the global interpolation function f, it has to be continuous at the boundary of the domains Ω_i. Obviously, the global interpolant f inherits the smoothness of the local functions f_i and the normalized weighting functions w_i. As a consequence, interpolants of arbitrary smoothness can be constructed depending on the choice of the local reconstruction functions f_i and the normalized weighting functions w_i.

Recall that the only constraint of the decomposition of Ω into M subdomains Ω_i is, that each subdomain Ω_i is overlapping with all neighboring subdomains in order to guarantee continuity. This fact leaves much freedom in the choice of the domain decomposition method, and a large variety of strategies can be adapted. We will present in the following two concrete domain decomposition methods relying on several decisions that have been taken, based on a regular space subdivision using a fixed grid and the use of an adaptive space subdivision based on an octree. Note that some of these decisions have been taken rather arbitrarily, pointing out the freedom of choice of the domain decomposition method.

2.3 Domain Decomposition

2.3.1 Fixed Grid Domain Decomposition

This first domain decomposition method defines the bounding box of the point set $\mathcal{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_N\}$ as the entire domain $\Omega \subset \mathbb{R}^3$, and decomposes it into a fine regular grid of axis-parallel cells. To each cell, a spherical or cubical domain Ω_i is associated with a diagonal that is higher than the cell's main diagonal in order to obtain overlapping subdomains.

A severe drawback of this method is, that the number of points in each cell can vary significantly depending on the distribution of the points. Only in case of quasi-uniformly distributed points, the number of points per cell can be bounded above by a constant T_{grid}, but there could be some cells containing no points at all. Anyway, in order to get stable local

reconstructions, the number of points per subdomain should be between an upper and a lower bound that can be freely determined.

For this reason, we did not implement the fixed grid domain decomposition method, since our initial challenge is to reconstruct surfaces from non-uniformly distributed, unorganized point sets.

2.3.2 Octree Domain Decomposition

Consequently, we developed a second domain decomposition method using an adaptive space subdivision method based on an octree [82, 109]. Given a point set $\mathcal{P}$ and two threshold values T_{min} and T_{max} with $T_{min} < T_{max}$, the entire domain Ω^0 is subdivided recursively into overlapping subdomains $\{\Omega_i\}_{i=1}^{M}$, until each subdomain Ω_i contains at least T_{min} and at most T_{max} points.

To determine the entire domain $\Omega = \Omega^0$, we fit the smallest axis aligned bounding cube $c^{[0]}$ with main diagonal d_c around the point set $\mathcal{P}$ in order to create cubic octant cells during the octree domain decomposition. Finally, we set Ω to be a centered bounding sphere or bounding cube around the cube with the diagonal d_Ω that is a factor β with $\beta \geq 1$ of the bounding cube's main diagonal d_c:

$$d_\Omega = \beta d_c \tag{34}$$

Starting from the entire domain Ω^0 at level 0 and as long as the number of points in the domain $\Omega_i^{[l]}$ associated to a cubic cell $c_i^{[l]}$ at level l of the octree is higher than T_{max}, i.e. $\text{Card}(\mathcal{P}_i^{[l]}) > T_{max}$with $\mathcal{P}_i^{[l]} = \{\boldsymbol{p} \in \mathcal{P} | \boldsymbol{p} \in \Omega_i^{[l]}\}$, we divide the cube recursively into eight disjoint cubic cells $c_1^{[l+1]} \dots c_8^{[l+1]}$. To each cubic cell $c_i^{[l+1]}$, we associate again a spherical or cubical domain with similar center and the diagonal of Equation (34). Note that the higher the value for β, the bigger are the subdomains and thus the more they overlap. When the number of points in $\Omega_i^{[l]}$ is smaller than T_{min}, we increase d_{Ω_i} by a factor γ until the number of points in $\Omega_i^{[l]}$ is higher than T_{min}:

$$d_{\Omega_i^{[l]}} = d_{\Omega_i^{[l]}} + \gamma d_{\Omega_i^{[l]}} \tag{35}$$

Possibly, when the number of points in Ω_i exceeds T_{max}, we have to decrease d_{Ω_i} by a factor smaller than γ accordingly. Note that in case that $d_{\Omega_i^{[l]}}$ is increased using Equation (35), this is only done temporarily in order to guarantee that the domain $\Omega_i^{[l]}$ contains more than T_{min} points for a stable local reconstruction. Then, after the local reconstruction function for $\Omega_i^{[l]}$ is created, the domain's diagonal is set to its original value of Equation (34) that will be considered in further domain queries.

We sketched the octree domain decomposition method in the recursive Algorithm 1 that has to be called with `decompose(`$\mathcal{P}$`,` $c^{[0]}$`,0)`, and the result of the domain decomposition method by associating either spherical or cubical domains to the octree cells can be seen in Figures 2(a) and 2(b), respectively. To form the partition of unity, we do not consider the domains resulting from the interior nodes of the octree that are sketched in dashed lines, but only the domains resulting from its leafs that we call $\mathcal{D} = \{\Omega_i\}_{i=1}^{M}$ in the following.

We will see in Section 2.6, that the octree used to create the overlapping subdomains can

Algorithm 1 decompose($\mathcal{P}$, $c^{[l]}$, l)

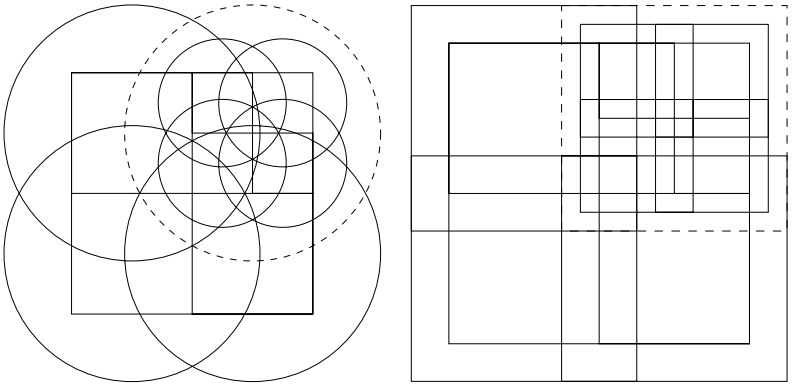
Require: points $\mathcal{P}$, cubic cell $c^{[l]}$, level l
Ensure: set of subdomains domains $\mathcal{D} = \{\Omega_i\}_{i=1}^{M}$
  compute $\Omega_i^{[l]}$ with $d_{\Omega_i^{[l]}} = \beta d_{c^{[l]}}$
  set n the number of points of $\mathcal{P}$ in $\Omega_i^{[l]}$
  if $n > T_{max}$ **then**
    subdivide $c^{[l]}$ into cubic octant cells $c_1^{[l+1]}, \ldots, c_8^{[l+1]}$
    decompose($\mathcal{P}$, $c_1^{[l+1]}$,$l+1$)
    ...
    decompose($\mathcal{P}$, $c_8^{[l+1]}$,$l+1$)
  else if $n < T_{min}$ **then**
    while $n \notin [T_{min}, T_{max}]$ **do**
      if $n < T_{min}$ **then**
        enlarge $\Omega_i^{[l]}$
      else if $n > T_{max}$ **then**
        reduce $\Omega_i^{[l]}$
      end if
      set n the number of points of $\mathcal{P}$ in $\Omega_i^{[l]}$
    end while
    domain OK, add $\Omega_i^{[l]}$ to $\mathcal{D}$
  else
    domain OK, add $\Omega_i^{[l]}$ to $\mathcal{D}$
  end if

be exploited for efficient range queries resulting in an efficient evaluation of the global defining function f of the reconstructed implicit surface.

2.4 Local Reconstruction

2.4.1 RBF Reconstruction

Once the set of local subdomains $\mathcal{D}$ is determined and the point sets $\mathcal{P}_i$ in the subdomains are collected, the local functions f_i have to be reconstructed. Of course, the partition of unity method does not impose any particular choice on the local reconstruction functions to use. In contrast to the multi-level partition of unity implicits [118], we reconstruct the local functions using radial basis functions, which seem to be the best suitable functions to reconstruct continuous smooth functions from scattered data [56]. Moreover, Wendland proved that putting the local radial basis functions together by the partition of unity method inherits the same approximation order than the global method [164]. Since we can delimit the number of points Card($\mathcal{P}_i$) per subdomain Ω_i to an interval $[T_{min}, T_{max}]$ using the octree domain decomposition method, we can use a rather "naive" reconstruction method for the local functions f_i. Thus, we reconstruct the local reconstruction functions f_i in spirit of Turk and O'Brien's variational implicit surfaces [156], that we already outlined in Subsection 1.4.2. We also define K off-surface constraints as explained in the following subsection. As



(a) Overlapping spheres ($\beta = 1.5$).

(b) Overlapping cubes ($\beta = 1.5$)

Figure 2: Slightly overlapping subdomains of an octree.

basic functions, we either use biharmonic basic functions $\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|$ associated with a linear polynomial (Equation (36)) resulting in C^0 continuous local reconstructions, or triharmonic basic functions $\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|^3$ associated with a quadratic polynomial (Equation (37)) resulting in C^2 continuous local reconstructions.

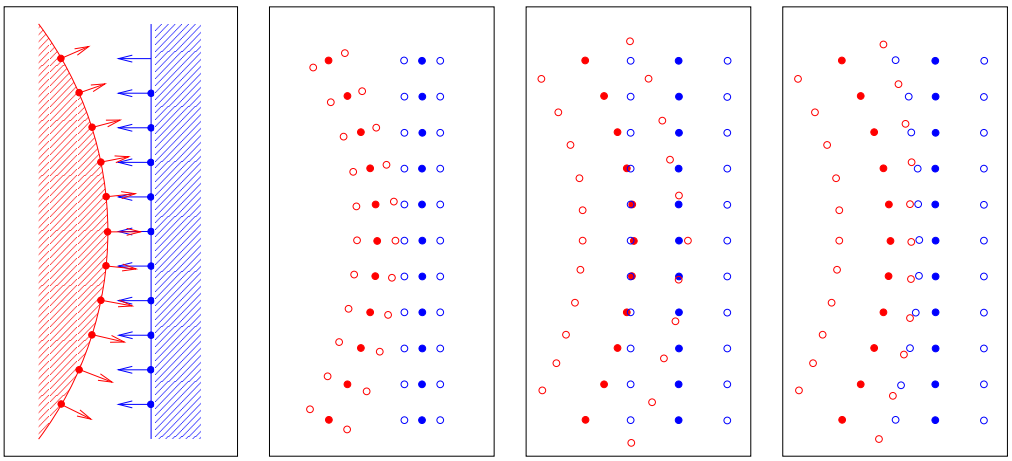
$$f(\boldsymbol{x}) = \sum_{i=1}^{N+K} w_i \|\boldsymbol{x} - \boldsymbol{p}_i)\| + \sum_{\alpha=1}^{4} \pi_\alpha p_\alpha(\boldsymbol{x}), \tag{36}$$

$$f(\boldsymbol{x}) = \sum_{i=1}^{N+K} w_i \|(\boldsymbol{x} - \boldsymbol{p}_i)\|^3 + \sum_{\alpha=1}^{10} \pi_\alpha p_\alpha(\boldsymbol{x}), \tag{37}$$

2.4.2 Off-surface Constraints

Recall that off-surface constraints have to be introduced in order to avoid the trivial solution of the linear system. We create two normal constraints from each initial point using the normal vector inspired by Carr et al. [33] and in contrast to Turk and O'Brien who create only one normal constraint for each initial point. To be more precise, starting from the point set $\mathcal{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_N\}$ with $f(\boldsymbol{p}_i) = h_i = 0$, for each point $\boldsymbol{p}_i$ in the point set, we create two normal constraints $f(\boldsymbol{p}_i') = h_i'$ and $f(\boldsymbol{p}_i'') = h_i''$ at two temporarily created off-surface points $\boldsymbol{p}_i'$ and $\boldsymbol{p}_i''$. The positions of the off-surface points $\boldsymbol{p}_i'$ and $\boldsymbol{p}_i''$ are computed starting from the initial points $\boldsymbol{p}_i$ and moving them along its normal vector $\boldsymbol{n}_i$, i.e. $\boldsymbol{p}_i' = \boldsymbol{p}_i + \kappa \boldsymbol{n}_i$ and $\boldsymbol{p}_i'' = \boldsymbol{p}_i - \kappa \boldsymbol{n}_i$, respectively. The normal is usually obtained during data acquisition, however, when the normal is not available, we estimate it from neighboring points [81] as already explained in Section 1.2.

Our experiments have shown that the final implicit surface is rather independent of the choice of the assigned function values h_i at the off-surface points. This has also been observed by Carr et al. as well as Wendland [33, 165]. In contrast, the parameter κ that specifies the *projection distance*, i.e. the distance between an initial point and one of its off-surface points,



(a) Unorganized points with normals.

(b) Correct projection distance.

(c) Incorrect projection distance.

(d) Validated projection distance.

Figure 3: Generation of off-surface points (outlined points) from unorganized points (bold points).

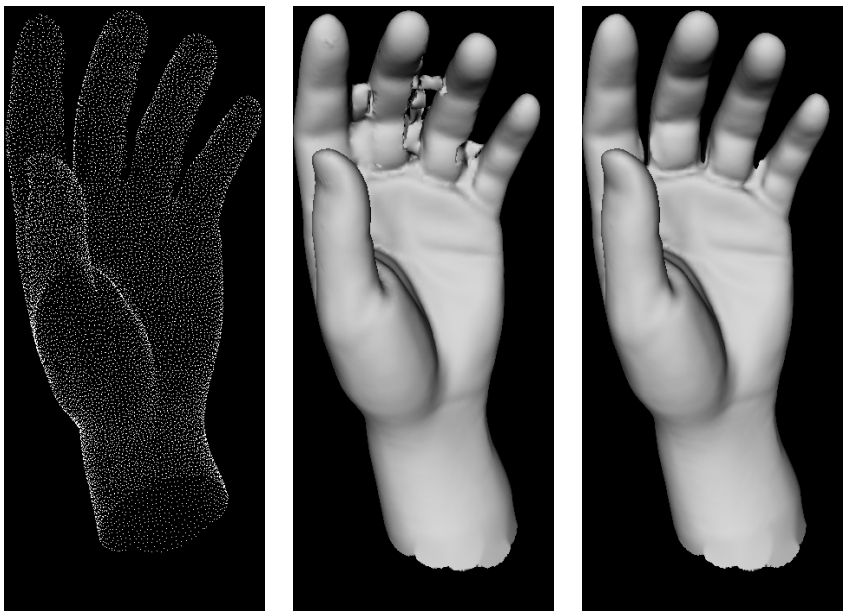
has to be chosen more carefully. Consider the unorganized points with their associated normal vectors of Figure 3(a). For a correct value of κ, the off-surface points form a shell around the points (Figure 3(b)), but for a too high value of the projection distance κ, projecting the off-surface point could intersect other parts of the surface (Figure 3(c)).

In order to qualify correct values for the projection distance κ, we use the *projection distance validation* method of Carr et al. [33, 55]: when the closest off-surface point $\boldsymbol{p}_i'$ or $\boldsymbol{p}_i''$ generated from $\boldsymbol{p}_i$ using the projection distance κ is not $\boldsymbol{p}_i$ itself, the off-surface point is moved towards $\boldsymbol{p}_i$ until it is the closest point of $\boldsymbol{p}_i$. See Figure 3(d) for the validated projection distance using the same value of κ as in Figure 3(c). As a general rule, we use $\frac{1}{\sqrt[3]{N}}$ % of the bounding box's diagonal as the value for κ. Note the resulting artefacts during the reconstruction of a point set of a hand (Figure 4(a)), where a too high value for κ can lead to cracks in the surface (Figure 4(b)) compared to the reconstruction of the hand using the validated projection distance (Figure 4(c)).

2.4.3 Approximating local Reconstructions

By using the method of Turk and O'Brien [156] for the local surface reconstruction, the reconstructed implicit surfaces interpolates the point set $\mathcal{P}_i$. In case of noise in the point set, it is desirable to reconstruct surfaces that are only approximating the point set. To this end, we take benefit of the variational character of the local reconstruction method, and allow a regularization parameter $\lambda > 0$ in Equation (10).

As a consequence, the diagonal of the matrix A involved in the linear system Equation (29) has to be modified to $A \leftarrow A - 8N\pi\lambda\boldsymbol{I}$ according to Carr et al. [33], where $\boldsymbol{I}$ is the identity matrix. The parameter λ controls the fitting tolerance, and the result is getting smoother



(a) Unorganized points.

(b) Too high projection distance.

(c) Validated projection distance.

Figure 4: Reconstruction from a point set of a hand.

when λ is increased.

See the excellent overview of Carr et al. for further methods to manage noise in the point set [34].

2.4.4 Surface Normals

Recall, that the normals of implicit surfaces are calculated using the gradient of the continuous and differentiable defining function f, and the normal $\boldsymbol{n}$ at a point $\boldsymbol{p} \in \mathcal{S}$ is

$$\boldsymbol{n} = \nabla f(\boldsymbol{p}) = \left[\frac{\partial f}{\partial x}(\boldsymbol{p}), \frac{\partial f}{\partial y}(\boldsymbol{p}), \frac{\partial f}{\partial z}(\boldsymbol{p})\right]^T. \tag{38}$$

One way to calculate the partial derivatives is to take a small value for ϵ:

$$\frac{\partial f}{\partial x}(\boldsymbol{p}) = \lim_{\epsilon \to 0} \frac{f(\boldsymbol{p} + \epsilon * [1,0,0]^T) - f(\boldsymbol{p})}{\epsilon}$$

$$\frac{\partial f}{\partial y}(\boldsymbol{p}) = \lim_{\epsilon \to 0} \frac{f(\boldsymbol{p} + \epsilon * [0,1,0]^T) - f(\boldsymbol{p})}{\epsilon}$$

$$\frac{\partial f}{\partial z}(\boldsymbol{p}) = \lim_{\epsilon \to 0} \frac{f(\boldsymbol{p} + \epsilon * [0,0,1]^T) - f(\boldsymbol{p})}{\epsilon} \tag{39}$$

(40)

In order to calculate the normals of implicit surfaces that are defined using radial basis functions, the choice of the value of ϵ should be neither too small, as this can lead to undesirable results due to numerical instabilities, nor too high because that would smooth out fine details. Hence, we decided to calculate the gradient of the local reconstruction functions f_i analytically, see Appendix A for the partial derivatives of the defining function using biharmonic basic functions and triharmonic basic functions.

2.5 Sticking Solutions together

Recall, that the global reconstruction function f is calculated by sticking together the local reconstruction functions f_i using the normalized weighting functions w_i, and by inserting Equation (32) in Equation (33), we obtain

$$f(\boldsymbol{x}) = \frac{\sum_{i=1}^{M} f_i(\boldsymbol{x})\hat{w}_i(\boldsymbol{x})}{\sum_{i=1}^{M} \hat{w}_i(\boldsymbol{x})}. \tag{41}$$

We will now see how to choose the weighting functions $\hat{w}_i$ that determine the continuity between the local solutions f_i and therefore the continuity of the global reconstruction function F. To this end, we define the weighting functions $\hat{w}_i$ as the composition of a distance function $d_i : \mathbb{R}^n \to [0,1]$, where $d_i(\boldsymbol{x}) = 1$ at the boundary of the local subdomain, and a decay function $\upsilon : [0,1] \to [0,1]$: $\hat{w}_i(\boldsymbol{x}) = \upsilon \circ d_i(\boldsymbol{x})$.

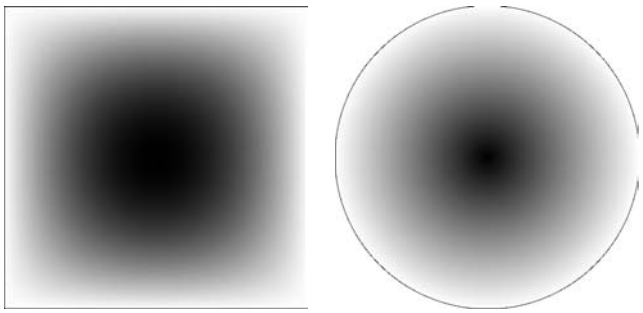
The distance function is defined depending on the shape of the local subdomains. The distance functions d_i^{cube} to use for a cube spanned between $\boldsymbol{a} = [x_a, y_a, z_a]^T$ and $\boldsymbol{b} = [x_b, y_b, z_b]^T$ with side length $u = |x_a - x_b| = |y_a - y_b| = |z_a - z_b|$ is shown in Equation (42). The distance function d_i^{sphere} for a sphere with center $\boldsymbol{c}$ and radius r is shown in Equation (43). We also sketched the respective function values in Figure 5.

$$d_i^{cube}(x,y,z) = 1 - \frac{64}{u^6} \prod_{v \in x,y,z} (v - v_a)(v_b - v) \tag{42}$$

$$d_i^{sphere}(\boldsymbol{x}) = \frac{\|\boldsymbol{x}, \boldsymbol{c}\|}{r} \tag{43}$$

The choice of the decay function υ influences the continuity of the global reconstruction function f, and in the following we show how to construct polynomial decay functions for different continuities. We only consider the construction of degree n polynomial decay functions of the form $\upsilon(x) = a_n x^n + \ldots + a_2 x^2 + a_1 x + a_0$, since they are the most efficient functions to evaluate, and nevertheless they are smooth. The construction of the decay functions is done in a similar way to the construction of spline functions and is explaned in the following.

In order to ensure that the weighting functions $\hat{w}_i$ have limited support to guarantee the locality of our approach, we set $\upsilon(1) = 0$, since with $\hat{w}_i = \upsilon \circ d_i$ and $d_i = 1$ at the boundary, the resulting weighting functions $\hat{w}_i$ are 0 at the boundary. Furthermore, without loss of generality, we set $\upsilon(0) = 1$, since its value will be overridden anyway during the normalization procedure of Equation (33). These two restrictions $\upsilon(0) = 1$ and $\upsilon(1) = 0$ are sufficient to



(a) Cubical distance function.

(b) Spherical distance function

Figure 5: Cubical and spherical distance functions values according to the convention $\left[\begin{smallmatrix}0\\ black\end{smallmatrix} \dashrightarrow \begin{smallmatrix}1\\ white\end{smallmatrix}\right]$.

determine the coefficients a_0 and a_1 of the linear decay function $v_0(x) = a_1 x + a_0$ as follows:

$$v_0(x) = 1 - x. \tag{44}$$

The decay function v_0 offers only a C^0 continuity at the boundary of the subdomains. In order to construct a decay function for ensuring a C^1 continuity between the local reconstruction functions, in addition to the constraints $v(0) = 1$ and $v(1) = 0$, the constraint $v'(1) = 0$ has to be satisfied, and for symmetry reasons we also require $v'(0) = 0$. The lowest degree polynomial function that can satisfy these four constraints is a cubic polynomial $v(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ with the derivative $v'(x) = 3a_3x^2 + 2a_2x + a_1$, and by inserting the four constraints, we obtain the following linear system:

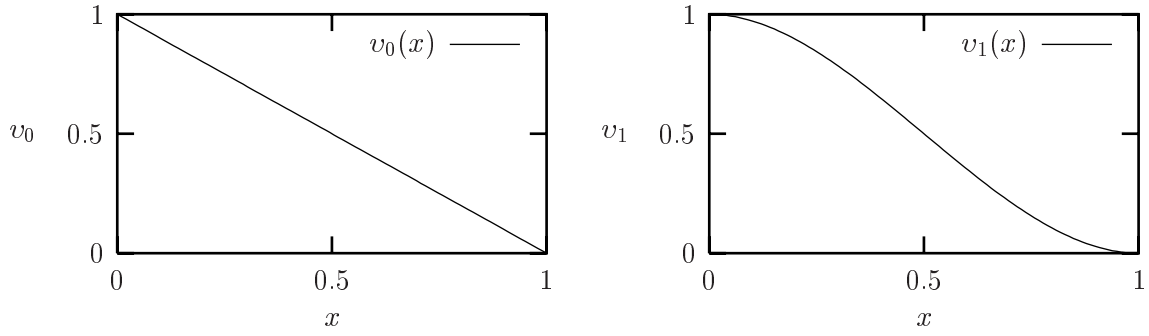
$$\begin{array}{lccccccccc}
v(0) = 1: & & & & & & & a_0 & = & 1 \\
v(1) = 0: & a_3 & + & a_2 & + & a_1 & + & a_0 & = & 0 \\
v'(1) = 0: & 3a_3 & + & 2a_2 & + & a_1 & & & = & 0 \\
v'(0) = 0: & & & & & a_1 & & & = & 0
\end{array}$$

Solving these systems leads to the decay function v_1 ensuring C^1 continuity between the local reconstruction functions shown in Equation (45).

$$v_1(x) = 2x^3 - 3x^2 + 1 \tag{45}$$

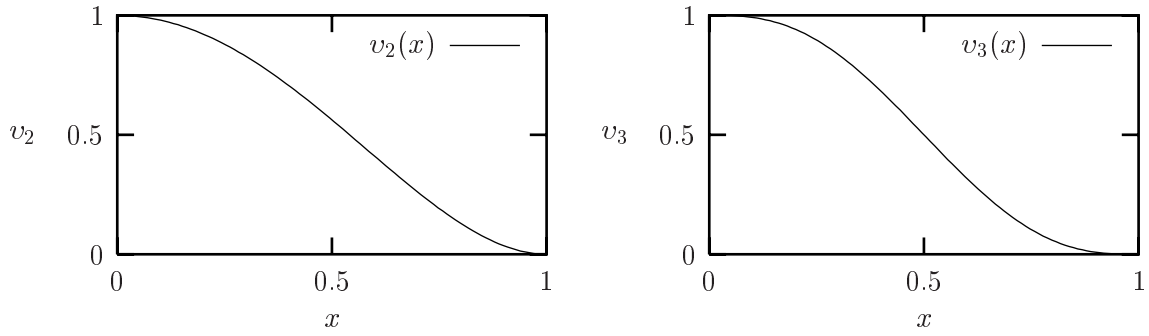
Decay functions ensuring higher continuity are constructed similarly, for a C^α continuity with $\alpha > 0$, the constraints $v^{(\alpha)}(0) = v^{(\alpha)}(1) = 0$ with $v^{(alpha)}$ being the α^{th} derivative have to be satisfied. See Table 3 for various decay functions for a given continuity with the corresponding derivatives, and their corresponding graphs in Figure 6 as well as the graphs of their first derivatives in Figure 7.

When evaluating the global defining function of Equation (41) at a given point $\boldsymbol{x}$, it is sufficient to collect all domains including the point $\boldsymbol{x}$, since the weighting functions $\hat{w}_i(\boldsymbol{x})$ are



(a) $\upsilon_0(x) = 1 - x$

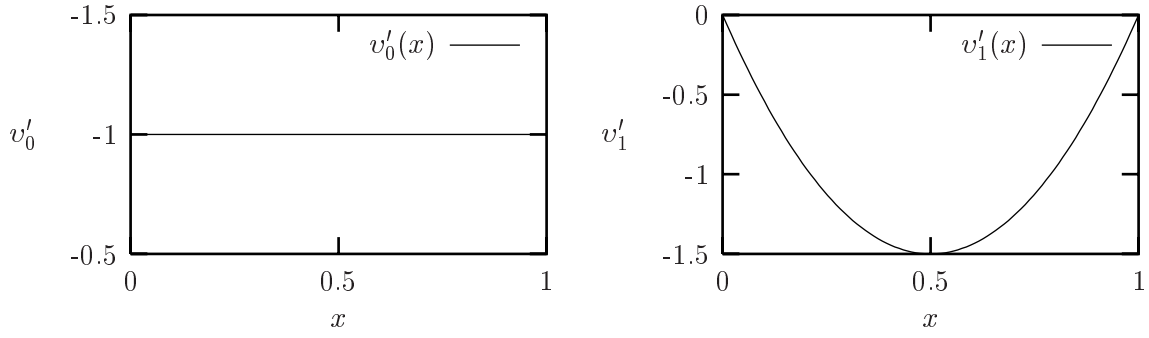
(b) $\upsilon_1(x) = 2x^3 - 3x^2 + 1$



(c) $\upsilon_2(x) = (x^2 - 1)^2$

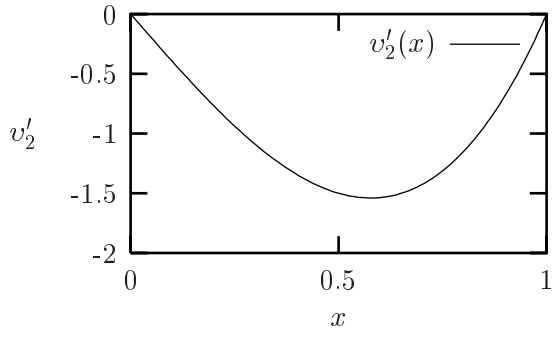
(d) $\upsilon_3(x) = -6x^5 + 15x^4 - 10x^3 + 1$

Figure 6: Graphs of the decay functions υ.

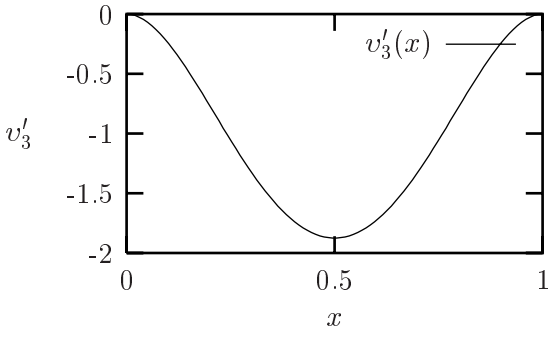


(a) $v_0'(x) = -1$

(b) $v_1'(x) = 6x^2 - 6x$



(c) $v_2'(x) = 4x(x^2 - 1)$



(d) $v_3'(x) = -30x^4 + 60x^3 - 30x^2$

Figure 7: Graphs of the derivatives v' of the decay functions.

Continuity	v	v'
C^0	$v_0(x) = 1 - x$	$v_0'(x) = -1$
C^1	$v_1(x) = 2x^3 - 3x^2 + 1$	$v_1'(x) = 6x^2 - 6x$
C^1	$v_2(x) = (x^2 - 1)^2$	$v_2'(x) = 4x^3 - 4x$
C^2	$v_3(x) = -6x^5 + 15x^4 - 10x^3 + 1$	$v_3'(x) = -30x^4 + 60x^3 - 30x^2$

Table 3: Some decay functions of different smoothness.

bounded ($\boldsymbol{x} \notin \Omega_i \Rightarrow \hat{w}_i(\boldsymbol{x}) = 0$).

In order to obtain the normals in a point on the reconstructed surface, we have to calculate the gradient and thus the partial derivatives of the global defining function f of Equation (41), and we refer to Appendix A for the determination of the partial derivatives.

2.6 Scalability

In this section, we analyze the scalability of the partition of unity variational method in terms of computational complexity. We divide this analysis into two phases, first the complexity analysis of the reconstruction of the global defining function f starting from a point set $\mathcal{P}$ consisting of N points, and second, the complexity analysis of the evaluation of the global defining function f at a point $\boldsymbol{x} \in \mathbb{R}^3$.

Concerning the first phase, let us first summarize the three necessary steps to reconstruct the global defining function f from an unorganized point set $\mathcal{P}$ using the partition of unity method.

1. Determine the entire domain Ω.
2. Decompose Ω into M local subdomains Ω_i and collect all points $\mathcal{P}_i$ that are in the local subdomains Ω_i, i.e. $\mathcal{P}_i = \{\boldsymbol{p} \in \mathcal{P} | \boldsymbol{p} \in \Omega_i\}$.
3. Compute the local defining functions f_i for all local subdomains Ω_i.

In the first step, the domain Ω is determined by a first $O(N)$ scan over the N points. Of course, the second step depends on the choice of the domain decomposition method. In the fixed grid domain decomposition method (Subsection 2.3.1), the grid can be determined in $O(1)$, and all points $\mathcal{P}_i$ are collected in $O(N)$ in case of quasi-uniformly distributed points, since the number of points per grid cell is bounded above by the constant T_{grid}. In the octree domain decomposition method, the points are collected during decomposition, and both is done in $O(N \log N)$ complexity. The third step is done in $O(N)$, since the number of points per subdomain is bounded by a constant and thus the number of subdomains M is proportional to the number of points N. Consequently, the memory requirement is in $O(N)$. Note that the upper bound constants T_{grid} and T_{max}, respectively, have to be of moderate size, otherwise the computation of the local interpolants in the third step can become prohibitively expensive.

Concerning the second phase, the following steps are necessary in order to evaluate the global defining function f at a point $\boldsymbol{x} \in \mathbb{R}^3$.

1. Find all subdomains Ω_i with $\boldsymbol{x} \in \Omega_i$.

2. Evaluate all local defining functions f_i and the corresponding weighting functions $\hat{w}_i$.

3. Determine the normalized weighting functions w_i by Equation (33), and sum up using Equation (32).

The first step is done in $O(1)$ using the fixed grid domain decomposition method and in $O(\log N)$ using the octree domain decomposition method, since every $\boldsymbol{x}$ is only contained in $O(1)$ subdomains. This follows immediately because the number of points per subdomain is bounded by a constant. Steps two and three can be computed in $O(1)$ as well, since the normalized weighting functions have limited support $\text{supp}(w_i) \subseteq \Omega_i$, i.e. $\forall \boldsymbol{x} \notin \Omega_i \Rightarrow w_i = 0$.

Summing up, the surface can be reconstructed in $O(N)$ time from quasi-uniformly distributed point sets $\mathcal{P}$ consisting of N points by using the fixed grid method, and in $\mu_{rec}O(N \log N)+\nu_{rec}O(N) = O(N \log N)$ time using the octree domain decomposition method for non-uniformly distributed point sets $\mathcal{P}$. Since the storage requirement is in $O(N)$, and the surface evaluation is in $O(1)$ for fixed grids and in $\mu_{eval}O(\log N)\nu_{eval}O(1) = O(\log N)$ for octrees, the partition of unity variational method is efficient according to the definition of Schaback [139].

But the practical experience with the octree domain decomposition method in the partition of unity variational method has even shown a better behavior. For the reconstruction phase, the constant μ_{rec} in the $\mu_{rec}O(N \log N)$ domain decomposition method is very small with respect to the constant ν_{rec} in the $\nu_{rec}O(N)$ local reconstructions. Similarly, during the evaluation, the constant μ_{eval} in the $\mu_{eval}O(\log N)$ determination of the involved local subdomains is rather small compared to the constant ν_{eval} in the $\nu_{eval}O(1)$ evaluation of the local defining functions.

2.7 Parameter Impact

2.7.1 Overview

Depending on the domain decomposition method, different parameters have been used to define the degree of locality. In this section, we will first present a procedure that compares two implicit surfaces in order to check the surface reconstruction quality. Based upon, we will analyse the quality of the surface reconstruction and the required time using different values for the parameters.

2.7.2 Measuring the Distance between two Implicit Surfaces

Usually, the difference between two surfaces is calculated using the *Hausdorff distance*, named after Felix Hausdorff (1868-1942), in Euclidean space. Based on the distance $d(x, S')$ between a point $\boldsymbol{p}$ on a first surface $\mathcal{S}$ and a second surface $\mathcal{S}'$ defined as

$$d(\boldsymbol{p}, \mathcal{S}') = \min_{\boldsymbol{p}' \in \mathcal{S}'} \{\|\boldsymbol{p} - \boldsymbol{p}'\|\}, \tag{46}$$

the Hausdorff distance is the maximum distance $d(\boldsymbol{p}, \mathcal{S}')$ of all points of the first surface $\boldsymbol{p} \in S$ to the nearest points of the other surface. More formally, the *maximin* function defining the Hausdorff distance $d_H(\mathcal{S}, \mathcal{S}')$ between two surfaces $\mathcal{S}$ and $\mathcal{S}'$ is

$$d_H(\mathcal{S},\mathcal{S}') = \max_{\boldsymbol{p}\in\mathcal{S}}\{d(\boldsymbol{p},\mathcal{S}')\} \quad (47)$$

$$= \max_{\boldsymbol{p}\in\mathcal{S}}\{\min_{\boldsymbol{p}'\in\mathcal{S}'}\{\|\boldsymbol{p},\boldsymbol{p}'\|\}\}. \quad (48)$$

Besides the Hausdorff distances measuring the maximum deviation between two given surfaces, we also define the *root mean square distance* d_{RMS} in order to characterize the averaged distances between two given surfaces as follows:

$$d_{RMS}(\mathcal{S},\mathcal{S}') = \sqrt{\frac{1}{\mathcal{S}}\int\int_{\boldsymbol{p}\in\mathcal{S}} d(\boldsymbol{p},\mathcal{S}')^2 d\mathcal{S}} \quad (49)$$

Note, that the distances d_H and d_{RMS} are in general not symmetric, so it is convenient to introduce symmetrical distances as $d_{symm} = \max[d(\mathcal{S},\mathcal{S}'), d(\mathcal{S}',\mathcal{S})]$.

To the best of our knowledge, there is unfortunately no method to analytically calculate the Hausdorff distance between two given implicit surfaces, so we had to develop the following strategy to calculate the Hausdorff distance approximately in order to compare two given implicit surfaces.

First, we polygonize the two implicit surfaces. A general overview of polygonizing implicit surfaces can be seen in Subsection 4.2.2 of Part II of this thesis, we decided to use the implementation proposed by Bloomenthal [24]. Then, we calculate the symmetrical Hausdorff distance $d_{H_{symm}}$ and $d_{RMS_{symm}}$ distances between the two resulting *polygonal meshes* using the MESH tool [10] that is a new and more efficient implementation of the METRO tool [38]. Of course, the polygonization only describes the implicit surface approximately, thus it introduces an error that decreases with an increasing precision of the polygonal mesh. Furthermore, polygonal meshes produced by the marching cubes algorithm are not rotation invariant since discrete samples are taken in regular axis-aligned spacings. Hence, to estimate the error that is introduced intrinsically by the polygonization process, we take the same implicit surface several times, rotate it arbitrarily and create various polygonal approximations in different precisions and compare the respective Hausdorff distances between them.

To this end, we reconstructed an implicit surface of the Stanford Bunny that was reduced aribitrarily to 600 points, and all connectivity information had been discarded. We polygonized the reconstructed implicit surface in different resolutions, i.e. in a grid size to be a factor ρ of the bounding cube of the model, and we created several different polygonizations at every resolution by rotating the implicit surface arbitrarily before. For every resolution, we calculated the maximum of the symmetrical Hausdorff distance $d_{H_{symm}}$ and the mean symmetrical Hausdorff distance $d_{RMS_{symm}}$ between all polygonizations. The quantitative results can be seen in Table 4, and the corresponding graphs are shown in Figure 8. Note that the unit of the Hausdorff distances are given in per cent of the bounding cube.

Since the difference between the Hausdorff distances of the polygonizations in different resolutions becomes smaller for an increasing resolution, we decided to polygonize all reconstructed implicit surfaces for the parameter evaluation using a grid size $\rho = 0.03$ of the model's bounding cube.

ρ	$d_{H_{symm}}$	$d_{RMS_{symm}}$
0.2	15.7050	0.5552
0.19	6.4054	0.5125
0.18	7.5797	0.4523
0.17	3.7464	0.3913
0.16	3.6180	0.1691
0.15	4.2926	0.3291
0.14	2.3921	0.2083
0.13	2.9503	0.2202
0.12	3.2918	0.1957
0.11	2.2491	0.1482
0.1	2,1903	0.1257
0.09	1.7888	0.1049
0.08	1.2429	0.0737
0.07	1.2005	0.0548
0.06	1.1299	0.0427
0.05	0.7787	0.0295
0.04	0.3768	0.0187
0.03	0.2197	0.0109
0.02	0.0998	0.0052
0.01	0.0348	0.0012

Table 4: Hausdorff distances in % of the bounding cube between polygonizations of arbitrarily rotated models at the same resolution with grid size factor ρ.

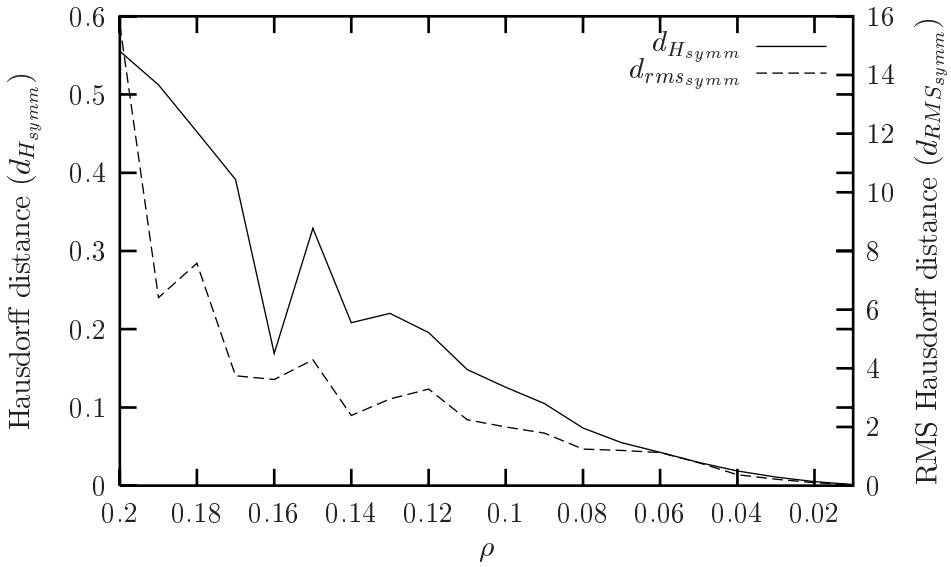


Figure 8: Hausdorff distances in % of the bounding cube between polygonizations of arbitrarily rotated models at the same resolution with grid size factor ρ.

2.7.3 Octree Domain Decomposition Parameters

In order to decompose the domain using an octree as introduced in Subsection 2.3.2, the parameters T_{min}, T_{max}, and β have to be chosen. Intuitively, higher values will lead to larger local domains and a higher surface reconstruction quality at the expense of a higher reconstruction time.

The surface reconstruction quality is compared by first reconstructing a reference surface using radial basis functions with global support according to Turk and O'Brien [156] but using the off-surface constraints of Subsection 2.4.2. Then, the difference in reconstruction quality using the partition of unity variational method with different parameters and the reference surface is determined.

First, we analyze the impact T_{min} and T_{max} parameters to the reconstruction quality. To this end, we compare a reference implicit surface that was reconstructed using one global radial basis function with the biharmonic basic function $\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|$ against implicit surfaces reconstructed by the partition of unity variational method using the same basic function, spherical subdomains Ω_i, $v_3(x) = -6x^5 + 15x^4 - 10x^3 + 1$, and $\beta = 1$. You can see the results for $T_{max} = 1.2T_{min}$, $T_{max} = 2T_{min}$, and $T_{max} = 5T_{min}$ in Table 5, Table 6, and Table 7, respectively. The number of resulting local subdomains is indicated in the column M, the required reconstruction time in column t_{rec}, and the Hausdorff distances between the reconstructed surface and the reference surface of the Stanford Bunny that was reduced aribitrarily to $N = 600$ points in columns $d_{H_{symm}}$ and $d_{RMS_{symm}}$. The results are also shown visually in Figures 9-11.

Let us now analyze the obtained results. A too small T_{min} may generate a too few number of points per subdomain, and the local reconstruction can lead to undesired results. On the other hand, a too high T_{max} may generate a too high number of points per subdomain resulting in an expensive reconstruction time for the local subdomain. For example, with $T_{max} = N$, only one subdomain is created ($M = 1$), and the reconstruction is global. We can see also, that when T_{max} approaches T_{min}, as in the example $T_{max} = 1.2T_{min}$, a larger number of subdomains that are highly overlapping is created since the domains are first subdivided and later enlarged according to Equation (35). This results in a higher reconstruction time. On the other hand, when T_{max} is much higher than T_{min}, as in the example $T_{max} = 5T_{min}$, the number of points in the local subdomains can be either too high, and this results also in a higher reconstruction time, or too low, and this results in a bad reconstruction quality. We found that $T_{max} = 2T_{min}$ is a good compromise between required reconstruction time and quality.

In general, higher values for T_{min} and hence T_{max} results in a higher reconstruction quality at the expense of a higher reconstruction time, and the reconstructed surfaces are more and more similar to globally reconstructed implicit surfaces.

Second, we tested the impact of the parameter β for fixed parameters of $T_{min} = 80$ and $T_{max} = 160$ on the Stanford Bunny that was reduced to 600 points. You can see the results for different values for β in Table 8 as well as the corresponding graphs in Figure 12. Note that the reconstruction quality is increased with a higher value for β, but the number of resulting local subdomains and the reconstruction time are increased as well.

We ran the same tests using cubical domains, and we did not notice any particular difference of the parameter impact.

T_{min}	T_{max}	M	t_{rec}	$d_{H_{symm}}$	$d_{RMS_{symm}}$
20	24	281	0.59	3.9176	0.1877
40	48	106	1.42	1.2971	0.0804
60	72	71	3.82	0.5670	0.0507
80	96	50	14.05	0.3138	0.0353
100	120	43	15.8	0.3024	0.0281
120	144	43	30.47	0.2133	0.0212
140	168	29	40.59	0.2084	0.0209
160	192	22	42.33	0.2227	0.0210
180	216	15	46.99	0.1969	0.0177
200	240	15	70.93	0.1983	0.0164
220	264	8	50.85	0.2114	0.0182
240	288	8	75.53	0.1978	0.0168
260	312	8	110.58	0.1970	0.0143
280	336	8	184.104	0.1991	0.0133
300	360	8	241.42	0.1967	0.0130
320	384	8	374.337	0.1975	0.0127
340	408	8	470.58	0.1995	0.0126
360	432	8	627.678	0.1995	0.0122
380	456	8	873.26	0.1952	0.0125
400	480	8	909.639	0.1952	0.0122
420	504	8	1263.24	0.1952	0.0120
440	528	8	1389.2	0.1952	0.0118
460	552	8	1539.33	0.1952	0.0116
480	476	8	1859.38	0.1952	0.0115
500	600	1	586.31	0.1952	0.0111

Table 5: Impact of T_{min} and T_{max} with $T_{max} = 1.2T_{min}$ on the number of subdomains M, the required reconstruction time in seconds, and the reconstruction quality compared to a globally reconstructed implicit surface.

T_{min}	T_{max}	M	t_{rec}	$d_{H_{symm}}$	$d_{RMS_{symm}}$
20	40	162	1.12	3.7958	0.1831
40	80	71	2.85	2.9071	0.1175
60	120	43	8.94	0.6015	0.0532
80	160	29	20.38	0.3574	0.0381
100	200	22	24.42	0.3046	0.0287
120	240	15	33.56	0.2330	0.0219
140	280	8	37.87	0.2340	0.0207
160	320	8	45.08	0.2323	0.0208
180	360	8	62.16	0.2313	0.0206
200	400	8	87.24	0.2219	0.0193
220	440	8	98.24	0.2167	0.0182
240	480	8	148.73	0.2016	0.0168
260	520	8	207.96	0.1195	0.0132
280	560	8	316.12	0.0793	0.0121
300	600	1	632.42	0.0584	0.0111

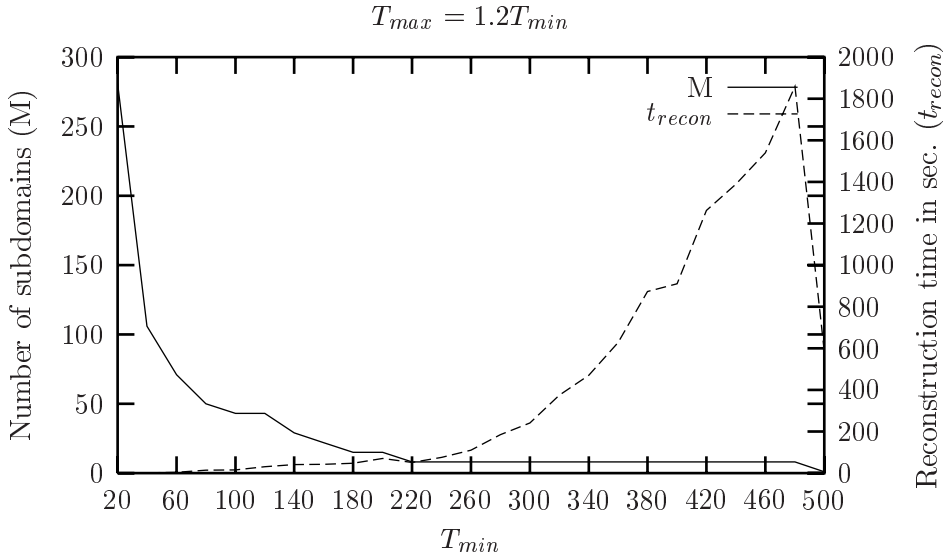
Table 6: Impact of T_{min} and T_{max} with $T_{max} = 2T_{min}$ on the number of subdomains M, the required reconstruction time in seconds, and the reconstruction quality compared to a globally reconstructed implicit surface.

T_{min}	T_{max}	M	t_{rec}	$d_{H_{symm}}$	$d_{RMS_{symm}}$
20	100	50	1.04	3.5731	0.1828
40	200	22	5.97	2.7512	0.1008
60	300	8	18.12	0.2453	0.0289
80	400	8	19.03	0.2450	0.0289
100	500	8	18.23	0.2383	0.0282
120	600	1	586.31	0.1964	0.0111

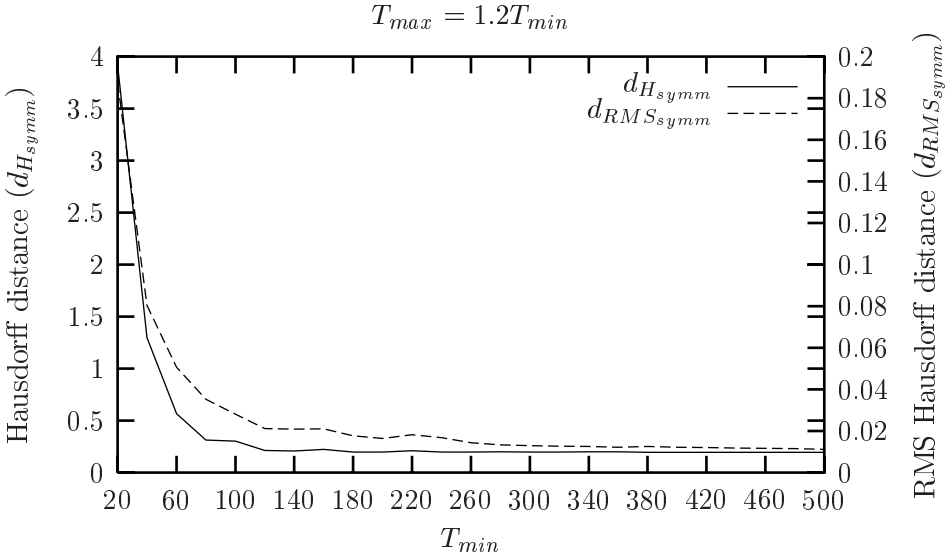
Table 7: Impact of T_{min} and T_{max} with $T_{max} = 5T_{min}$ on the number of subdomains M, the required reconstruction time in seconds, and the reconstruction quality compared to a globally reconstructed implicit surface.

β	M	t_{rec}	$d_{max_{symm}}$	$d_{RMS_{symm}}$
1	29	20.38	0.3574	0.0381
1.2	57	29.37	0.2032	0.0348
1.4	64	41.09	0.3068	0.0338
1.6	99	66.68	0.2072	0.0269
1.8	134	97.99	0.1979	0.0247
2	225	135.08	0.1476	0.0224

Table 8: Impact of β on the number of subdomains M, the required reconstruction time in seconds, and the reconstruction quality compared to a globally reconstructed implicit surface.

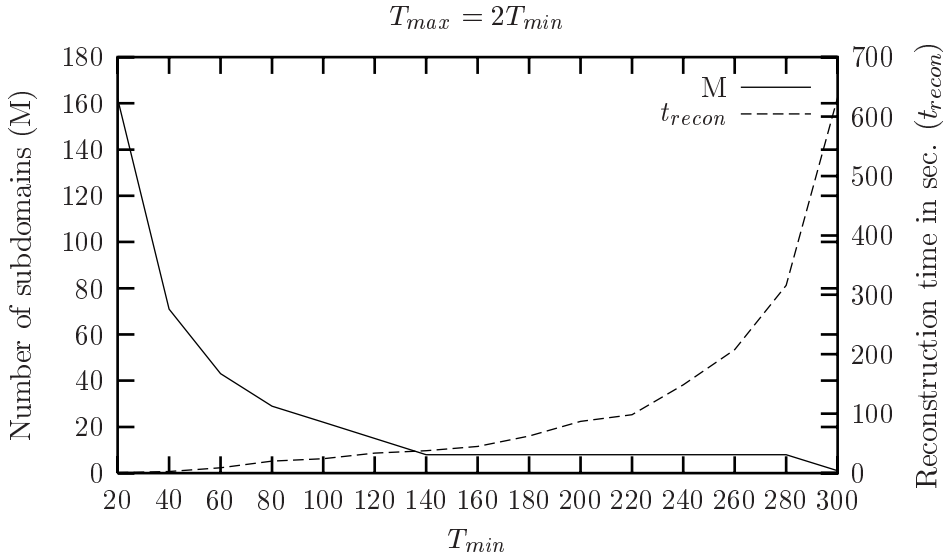


(a) Subdomains and reconstruction time.

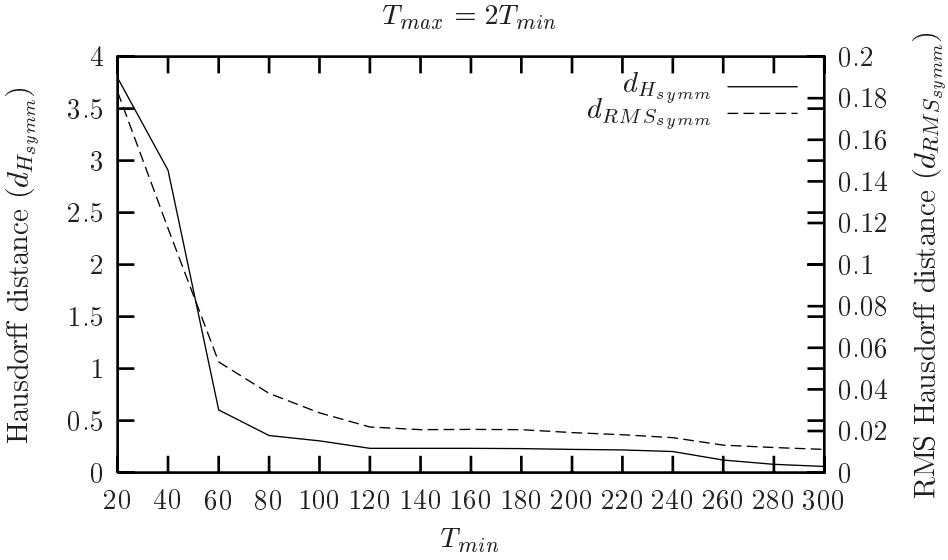


(b) Hausdorff distances.

Figure 9: Impact of T_{min} and T_{max} with $T_{max} = 1.2T_{min}$ on the number of subdomains M, the required reconstruction time in seconds, and the reconstruction quality compared to a globally reconstructed implicit surface.

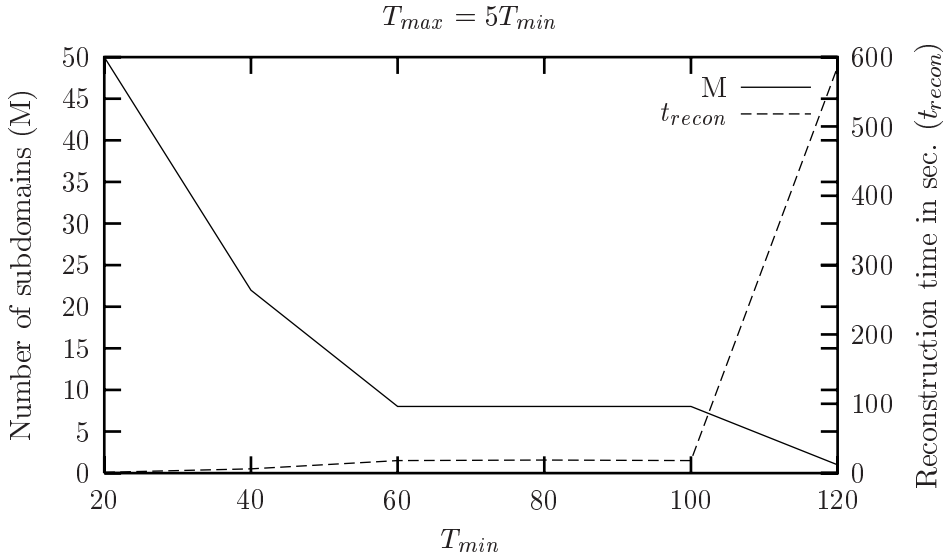


(a) Subdomains and reconstruction time.

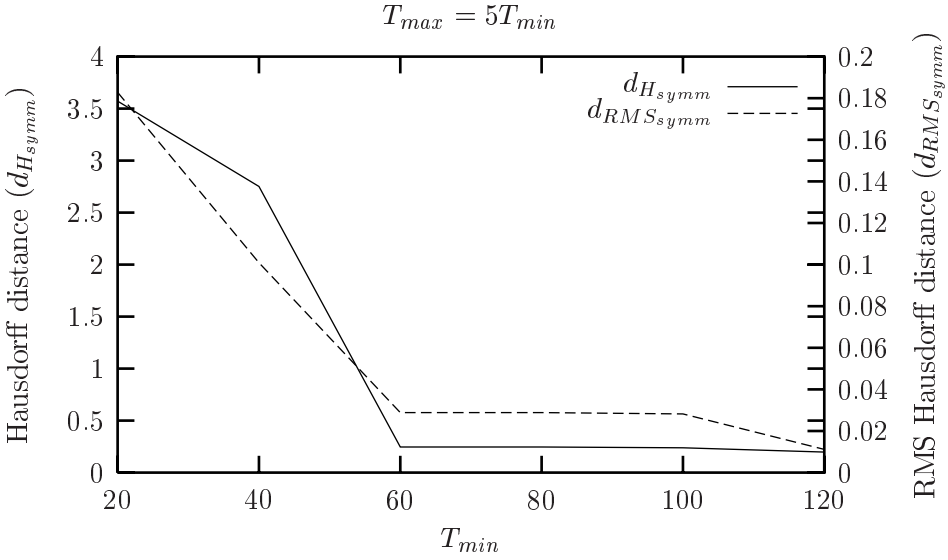


(b) Hausdorff distances.

Figure 10: Impact of T_{min} and T_{max} with $T_{max} = 2T_{min}$ on the number of subdomains M, the required reconstruction time in seconds, and the reconstruction quality compared to a globally reconstructed implicit surface.

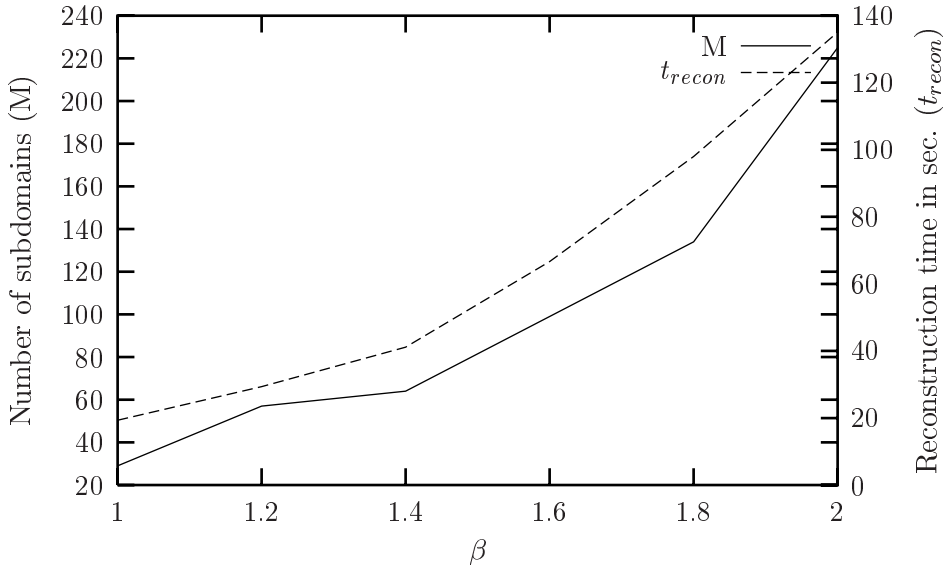


(a) Subdomains and reconstruction time.

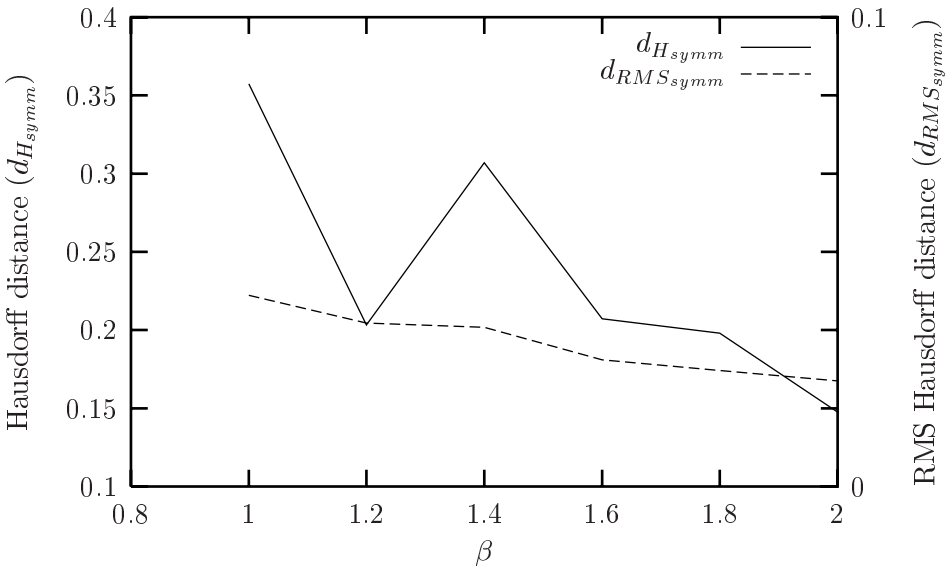


(b) Hausdorff distances.

Figure 11: Impact of T_{min} and T_{max} with $T_{max} = 5T_{min}$ on the number of subdomains M, the required reconstruction time in seconds, and the reconstruction quality compared to a globally reconstructed implicit surface.



(a) Subdomains and reconstruction time.

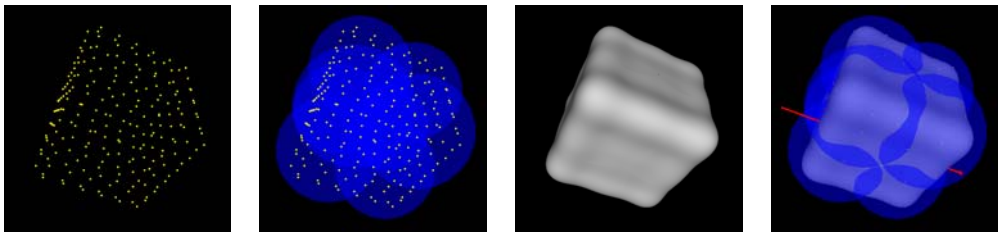


(b) Hausdorff distances.

Figure 12: Impact of β on the number of subdomains M, the required reconstruction time in seconds, and the reconstruction quality compared to a globally reconstructed implicit surface.

2.8 Example

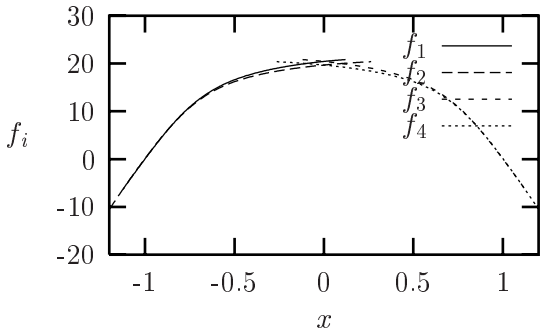
In order to illustrate the new reconstruction method based on the partition of unity, we discuss a very simple example in the following. Consider the point set of Figure 13(a) consisting of 200 points. Using the octree domain decomposition method of Subsection 2.3.2 with $T_{min} = 50$, $T_{max} = 150$, and $\beta = 1.5$, eight spherical domains are created (Figure 13(b)), and the reconstructed implicit surface using the triharmonic basic function $\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|^3$ is shown in Figure 13(c).



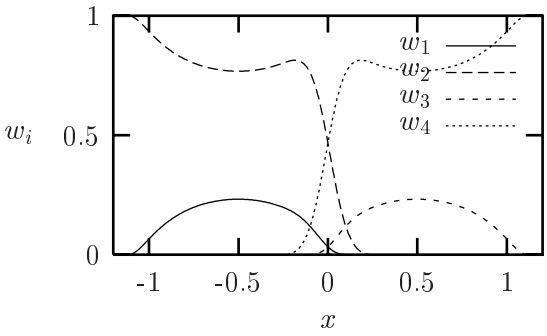
(a) Seed point set. (b) Octree domain decomposition. (c) Reconstructed surface. (d) Intersecting ray.

Figure 13: A simple example of the partition of unity variational method.

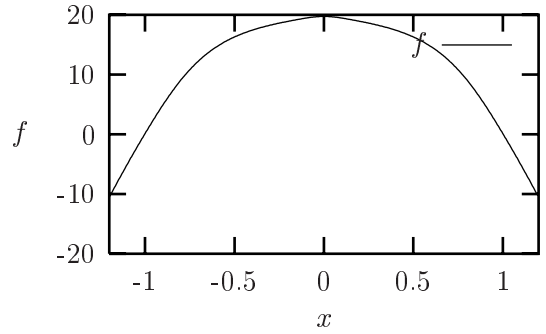
We will now analyze the behavior of some involved functions along a ray that passes through the implicit surface shown in Figure 13(d). The ray only passes through four of the eight spherical domains, and the four corresponding local reconstruction functions are shown in Figure 14(a). By using the decay function v_3 of Table 3 and the spherical distance function (43), the resulting normalized weighting functions after the normalization procedure of Equation (33) along the ray are shown in Figure 14(b). Finally, by putting together the local solutions using Equation (32), the global reconstruction function along the ray and its derivative are shown in Figures 14(c) and 14(d), respectively.



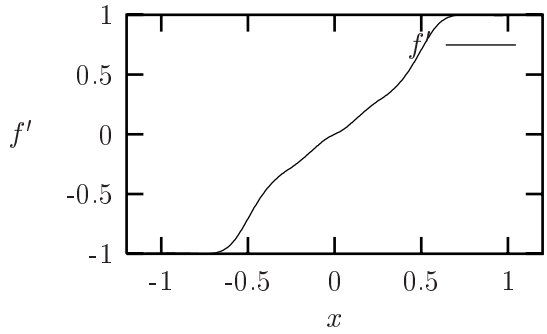
(a) Local reconstruction functions.



(b) Normalized weighting functions.



(c) Global reconstruction function.



(d) Derivative of the global reconstruction function.

Figure 14: Graphs of the values of the involved functions along a ray.

(a) Stanford Bunny. (b) Stanford Dragon. (c) Stanford Buddha.

Figure 15: Some visual results when reconstructing from the vertices of polygonal meshes.

2.9 Results

We tested the partition of unity variational method to reconstruct implicit surfaces from a variety of point sets coming from different sources, and in this section we show the obtained quantitative and qualitative results. All results were obtained on an Intel Pentium 1.7 GHz with 512 MB of RAM running Linux. To solve the linear systems involved in the local reconstruction processes, we used the linear solver from the GNU Scientific Library package [61] based on LU-decomposition. We decided to drive all the results by using the biharmonic basic function $\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|$ for the local reconstructions. Furthermore, we exclusively used the decay function $\upsilon_3(x) = -6x^5 + 15x^4 - 10x^3 + 1$.

As a first class of point sets, we take the vertices of polygonal meshes and discard all connectivity information. You can see some visual results of the reconstructed Stanford Bunny, the Stanford Dragon, and the Stanford Buddha in Figure 15.

Let us analyze the impact of T_{min} and T_{max} while reconstructing the Stanford Bunny from $N = 34,834$ points. Recall that we use two normal constraints per point, and the total number of constraints is $3N$. In table 9, the number of created subdomains is denoted by M and the required time is given by t_M. The required time by using the biharmonic basic function for the local reconstruction is given by t_{rec} and the resulting total reconstruction time by t_{total}. We also sketched the total reconstruction time t_{total} with respect to T_{min} and T_{max} in a graph in Figure 16.

Together with the obtained visual results, we found $T_{min} = 50$ and $T_{max} = 100$ to be a good compromise between locality of reconstruction and the required reconstruction time, and we will use these values for the further tests in this section.

Table 10 illustrates the scalability of the partition of unity variational method. We reduced the number of points of the original Stanford Bunny (34,834 points), the Stanford Dragon (437,645 points), and the Stanford Buddha (543,652 points) arbitrarily to N points. According to the results experienced above, we set $T_{min} = 50$ and $T_{max} = 100$ and use the biharmonic basic function $\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|$ for the local reconstructions. The number of domains after applying the domain decomposition method is denoted with M, and the required time is given by t_M.

We also show the required time for the local reconstructions in the subdomains (t_{rec}), and the accumulated total reconstruction time t_{total}. Consider the corresponding graphs of the required times with respect to the number of points in Figure 17.

The partition of unity variational method is particularly adapted to reconstruct implicit surfaces from non-uniformly distributed point sets. To illustrate the robustness against highly non-uniformly distributed point sets, we artificially created subsets of points of the Stanford Bunny and the Stanford Dragon with a sharp density variation and a smooth density variation, respectively (Figure 18(a) and 18(b)). The results of the reconstruction using the partition of unity variational method can be seen in Figures 18(c) and 18(d).

We will now show the impact of the regularization parameter λ smooothing out fine details to reconstruct implicit surfaces with noise in the point set. In order to elaborate qualitative results, consider the Stanford Bunny with its 34,834 points. Again, we discarded all connectivity information and reconstructed the implicit surface for different values of the regularization parameter λ. The higher the value for lambda, the more the model is approximated, and hence the more the concavities in the Stanford Bunny's fur and ear disappear as can be seen in the results shown in Figure 19. The Hausdorff distances of the different reconstructed models for $\lambda > 0$ to the reconstructed model with $\lambda = 0$ can be seen in Table 11.

As a second class of point sets, we take unorganized points from range scanners. We reconstructed the Cyberware Igea consisting of 134,346 points (Figure 20(a)) and the Max Planck head (Figure 20(b)) consisting of 52,809 points using the partition of unity variational method with $T_{min} = 50$, $T_{max} = 100$, and $\beta = 1$ in $5,205$ seconds and $1,205$ seconds, respectively. The resulting reconstructed implicit surfaces can be seen in Figure 20(c) and 20(d), respectively.

In order to show that the reconstructed implicit surfaces apply well for global illumination, we implemented a plugin for POVray [126] that allows to ray-trace implicit surfaces reconstructed by the partition of unity variational method. With this plugin, photorealistic images can be produced, and in Figure 21, you can see two examples of the ray-tracing of the Max Planck head and the Stanford Bunny.

T_{min}	T_{max}	M	t_M	t_{rec}	t_{total}
20	40	7960	14.07	138.67	152.74
20	60	6756	14.91	132.91	147.82
20	80	6028	13.47	135.43	148.9
20	100	5216	9.35	130.98	140.33
20	120	4320	9.50	200.57	210.07
20	140	3172	6.68	327.77	334.45
20	160	2052	3.24	537.29	540.53
20	180	1779	3.04	608.01	611.05
20	200	1667	3.1	659.57	662.67
40	60	6756	11.01	229.92	240.93
40	80	6028	9.52	217.18	226.7
40	100	5216	11	240.67	251.67
40	120	4320	6.83	281.8	288.63
40	140	3172	5.86	409.27	415.13
40	160	2052	3.62	575.8	579.42
40	180	1779	2.82	682.19	685.01
40	200	1667	2.44	720.14	722.58
40	220	1597	2.67	776.28	778.95
60	80	6028	10.71	567.55	578.26
60	100	5216	9.47	637.69	647.16
60	120	4320	9.1	579.04	588.14
60	140	3172	8.79	672.14	680.93
60	160	2052	4.65	754.72	759.37
60	180	1779	2.96	890.81	893.77
60	200	1667	3.13	981.19	984.32
60	220	1597	3.26	940.56	943.82
80	100	5216	8.56	1418.07	1426.63
80	120	4320	9.1	1486.33	1495.43
80	140	3172	6.17	1298.12	1304.29
80	160	2052	4.27	1206.97	1211.24
80	180	1779	3.80	1131.58	1135.38
80	200	1667	3.27	1050.3	1053.57
80	220	1597	2.98	1098.45	1101.43
100	120	4320	7.31	2034.27	2041.58
100	140	3172	5.78	1707.39	1713.17
100	160	2052	3.85	1422.45	1426.3
100	180	1779	2.58	1412.27	1414.85
100	200	1667	2.5	1421.36	1423.86
100	220	1597	3.19	1451.9	1455.09
120	140	3172	5.51	2648.92	2654.43
120	160	2052	3.93	2057.33	2061.26
120	180	1779	2.58	1938.65	1941.23
120	200	1667	3.22	1948.19	1951.41
120	220	1597	2.34	1979.07	1981.41
140	160	2052	3.9	2861.88	2865.78
140	180	1779	2.77	2786.25	2789.02
140	200	1667	2.91	2805.32	2808.23
140	220	1597	3.05	2813.77	2816.82
160	180	1779	2.58	3733.24	3735.82
160	200	1667	2.84	3840.42	3843.26
160	220	1597	2.71	4059.2	4061.91
180	200	1667	2.56	5029.62	5032.18
180	220	1597	2.37	5508.26	5510.63
200	220	1597	2.52	8016.18	8018.7

Table 9: Impact of T_{min} and T_{max} on the reconstruction time of the Stanford Bunny.

Model	P	M	t_M	t_{rec}	t_{total}
Dragon	1,000	134	0.02	5.52	5.54
	2,000	288	0.12	12.77	12.89
	4,000	561	1.23	33.08	34.31
	8,000	897	1.58	53.67	55.25
	16,000	1,940	6.45	146.44	152.89
	32,000	4,068	26.17	449.44	475.61
	64,000	8,009	98.38	1162.72	1261.1
	128,000	17,039	333.16	3448.41	3781.57
	256,000	32,467	589.82	7224.44	7814.26
	437,645	61,755	1216.65	12853.15	14069.8
Buddha	1,000	155	0.03	8.1	8.13
	2,000	253	0.16	21.74	21.9
	4,000	491	0.63	31.02	31.65
	8,000	932	1.93	75.81	77.74
	16,000	2,171	9.27	196.03	205.3
	32,000	3,893	28.9	458.77	487.67
	64,000	8,828	115.58	1328.42	1444
	128,000	15,856	343.09	4319.41	4662.5
	256,000	34,966	1018.9	8230.98	9249.88
	512,000	65,423	1555.6	14816.4	16372.1
	543,652	69,651	1686.67	15669.33	17356.9
Bunny	1,000	78	0.01	3.96	3.97
	2,000	239	0.09	9.47	9.56
	4,000	330	0.26	23.28	23.54
	8,000	1107	1.41	61.37	62.78
	16,000	1569	2.51	108.68	111.19
	32,000	4845	7.53	420.39	427.92
	34,834	5216	13.61	398.89	412.50

Table 10: Scalability of the reconstruction.

λ	$d_{max_{symm}}$	$d_{mean_{symm}}$	$d_{RMS_{symm}}$
0.00001	0.419367	0.0143101	0.0227934
0.00005	0.828604	0.0365094	0.0531872

Table 11: Hausdorff distance for different values for the regularization parameter λ compared to $\lambda = 0$.

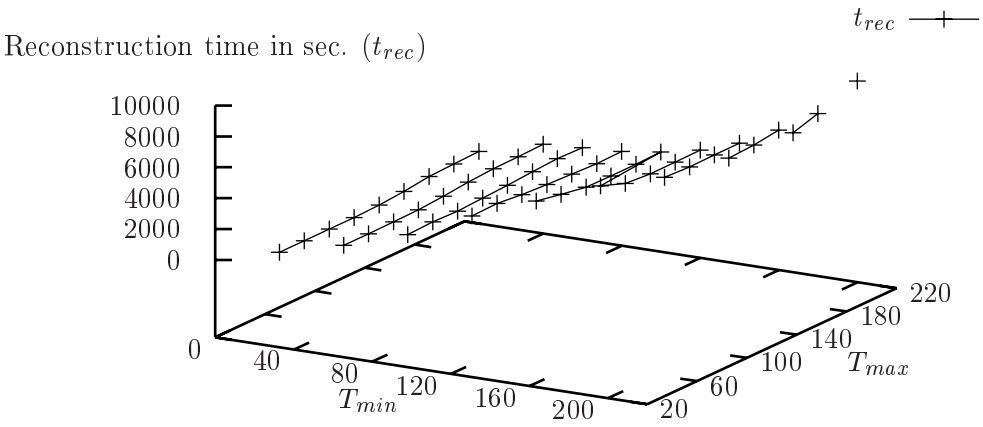
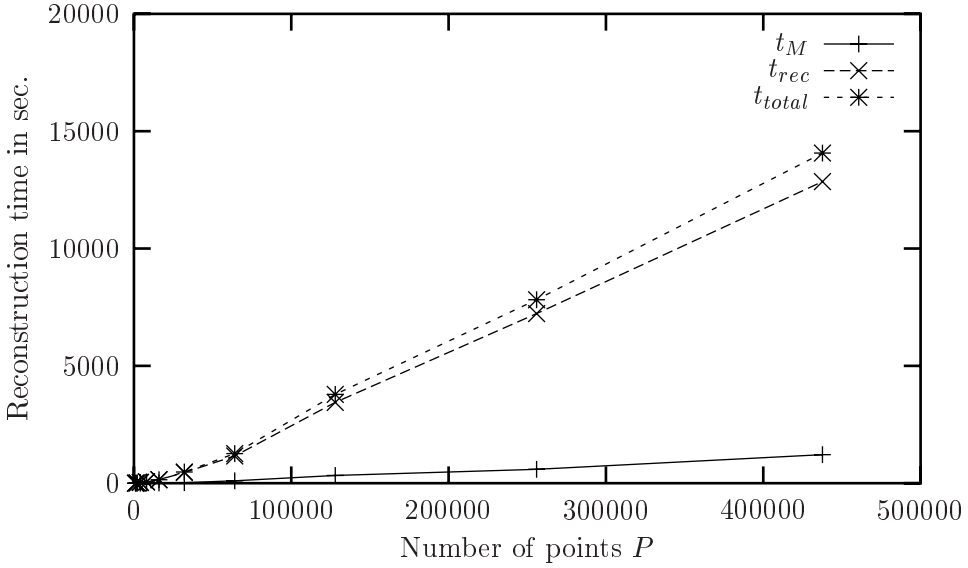
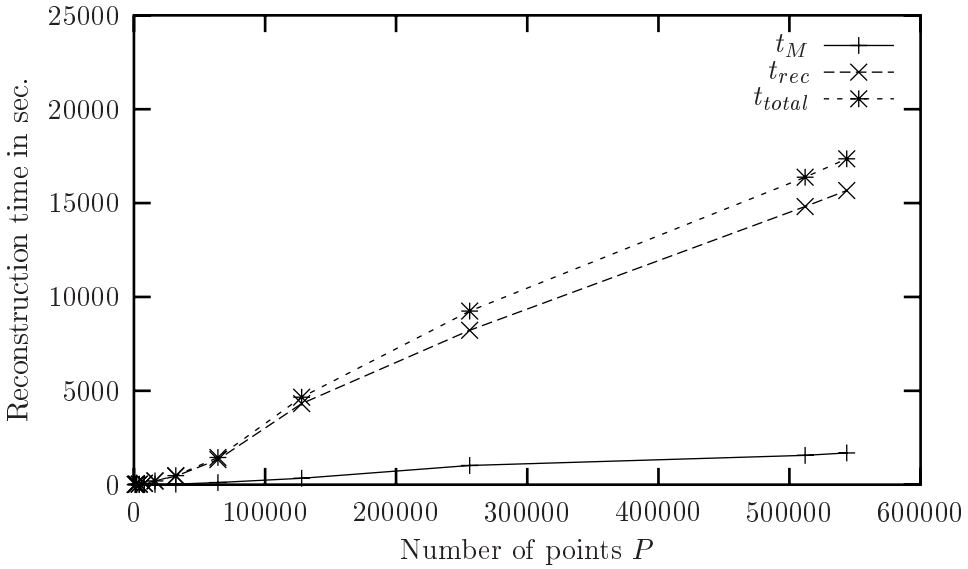


Figure 16: Impact of T_{min} and T_{max} on the reconstruction time of the Stanford Bunny.



(a) Dragon timings.

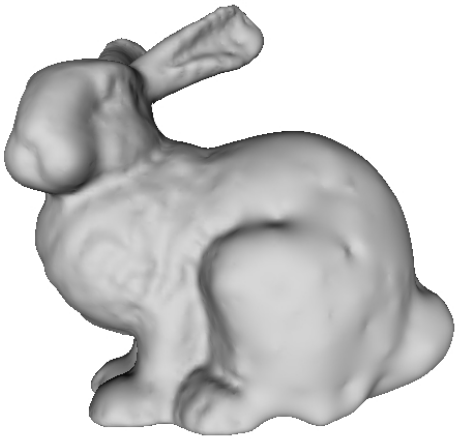


(b) Buddha timings.

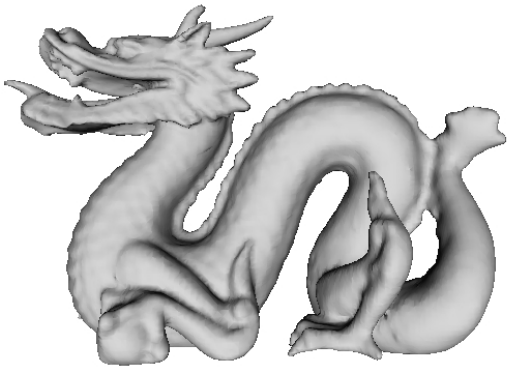
Figure 17: Scalability of the reconstruction time with respect to the number of points.

(a) Sharp density variation.

(b) Smooth density variation.

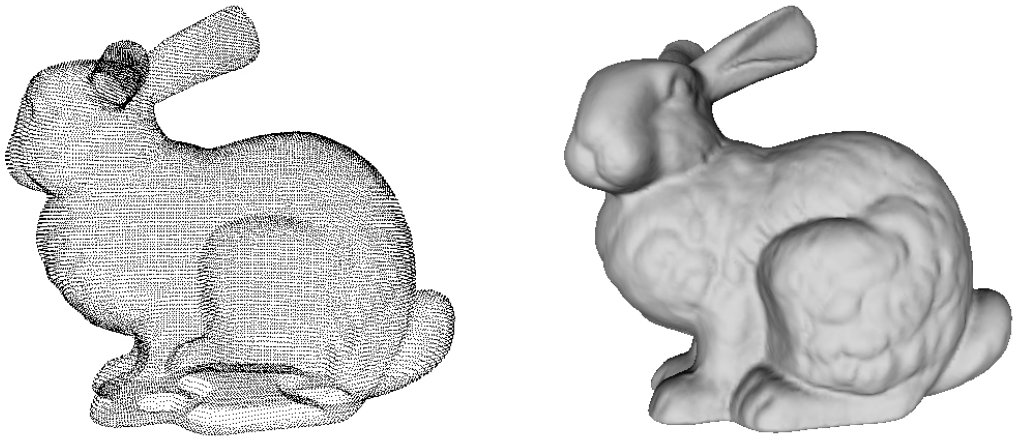


(c) Reconstructed surface.



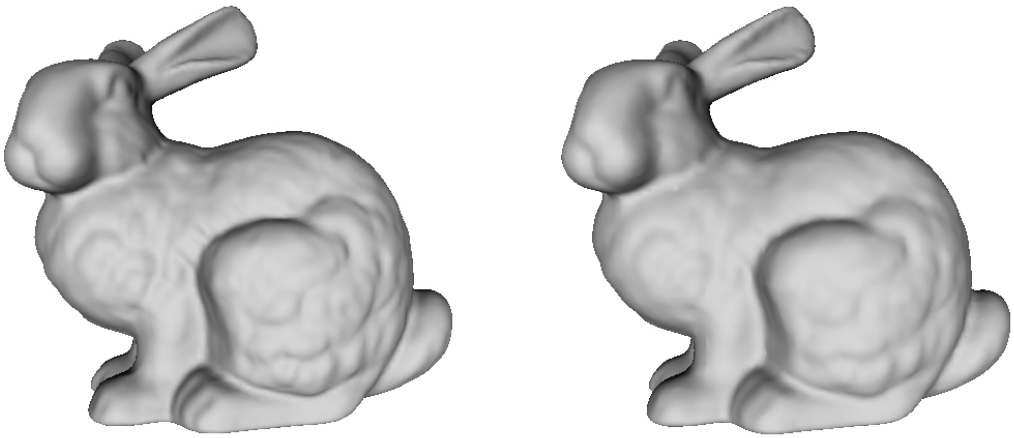
(d) Reconstructed surface.

Figure 18: Robustness of the partition of unity variational method against highly non-uniformly distributed point sets.



(a) 34,834 unorganized points.

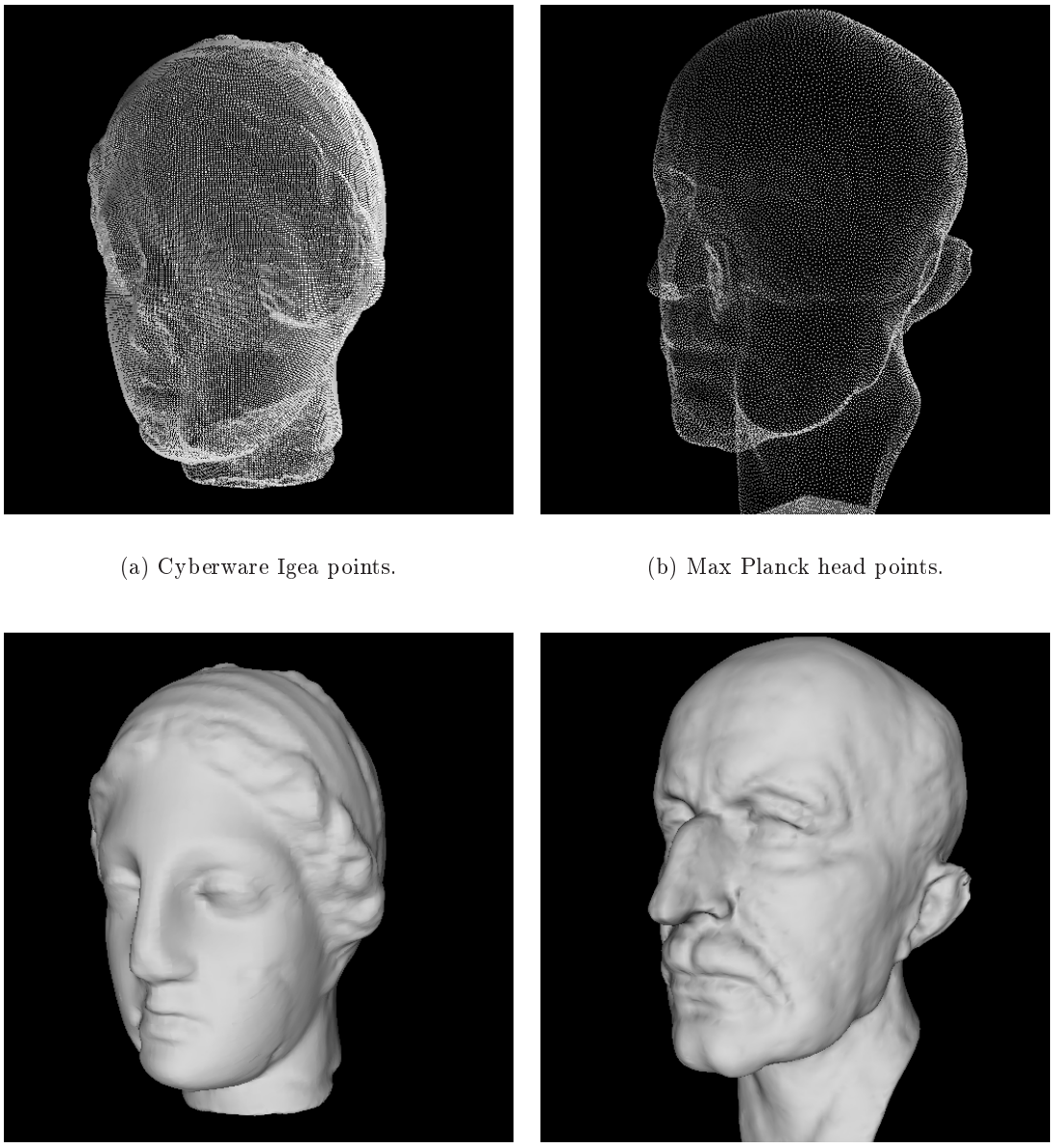
(b) $\lambda = 0$.



(c) $\lambda = 0.00001$.

(d) $\lambda = 0.00005$.

Figure 19: Approximating unorganized point sets with different values for the regularization parameter λ.



(a) Cyberware Igea points.

(b) Max Planck head points.

(c) Reconstructed surface.

(d) Reconstructed surface.

Figure 20: Reconstruction from range scanner data.

(a) Max Planck head.

(b) Stanford Bunny.

Figure 21: Ray-tracing using our POVray plugin.

2.10 Conclusions

In this chapter, we described the partition of unity variational method, a new method to reconstruct implicit surfaces from large unorganized point sets. This method divides the global reconstruction domain into smaller overlapping local subdomains using an adaptive domain decomposition method by using an octree, solves the reconstruction problems in the local subdomains using radial basis functions with global support, and blends the solutions together using the partition of unity method.

The method has a nice linear behavior of the required reconstruction time and memory usage with respect to the number of points in the point set, because the $O(N \log N)$ creation involved in the domain decomposition method is negligible compared to the $O(N)$ reconstruction of the local subdomains. Furthermore, the local reconstruction problems can be solved by various, non-communicating entities due to the independence of the local subdomains. We believe that this fact enables a straightforward out-of-core implementation of the new implicit surface reconstruction methods, where only the data of the local subdomain to reconstruct has to be kept in core.

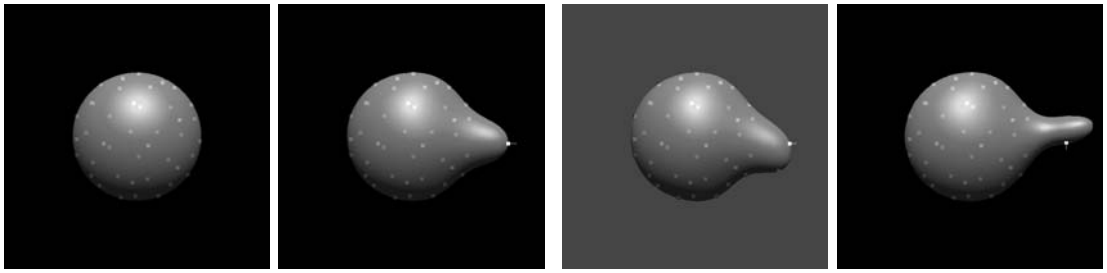
We showed the quality of the partition of unity variational method on a variety of point sets coming from different domains, and the quantitative results confirmed our anticipation of the linear complexity behavior. We think that the simplicity of the described process makes the new method highly accessible, compared for example to the method based on fast evaluation techniques of radial basis functions using Fast Multipole Methods of Carr et al. [33].

The partition of unity variational method distinguishs itself by the freedom it offers to decompose the global domain into local subdomains, even allowing to combine different subdomain shapes. This can be particularly useful when there is already some information about the unorganized point set to handle.

Implicit surfaces reconstructed by using the partition of unity variational method are composed of variational implicit surfaces in the local subdomains. This assumes that the reconstructed surface is everywhere locally symmetric, making it impossible to preserve sharp features along edges and corners. One elegant solution to overcome this limitation may be to use anisotropic basis functions [52] for the local reconstructions. But by sticking together the local reconstructions by using the partition of unity method, partition of unity blendings between sharp and smooth parts of the surface do not preserve sharp features either. We believe, that multi-level partition of unity implicits suffer from the same drawback, and that it is impossible to guarantee the preservation of sharp features without an extensive topology study.

In general, since we use the same type of local reonstructions in the subdomains, the variational implicit surfaces, we believe that our method is more stable against topological differences between blended subdomains than multi-level partition of unity implicits [118]. On the other hand, reconstructing variational implicit surfaces in the local subdomains is much slower than reconstructing quadratic surfaces, and consequently multi-level partition of unity implicits are much faster to reconstruct than reconstructing implicit surfaces by the partition of unity variational method.

Compared to reconstruction methods using compactly supported radial basis functions [113, 90], we can reconstruct implicit surfaces from unorganized point sets and not only quasi-uniformly distributed point sets.



(a) Starting points. (b) Moving a point. (c) Adding a point. (d) Changing a normal.

Figure 22: Simple low-level modelling demonstrated on a sphere.

Finally, since we use two normal constraints for the involved reconstructions of the local subdomains, we do not have to define a carrier solid that sometimes influences the shape of the reconstructed implicit surface [90] since it has to be bijective to it.

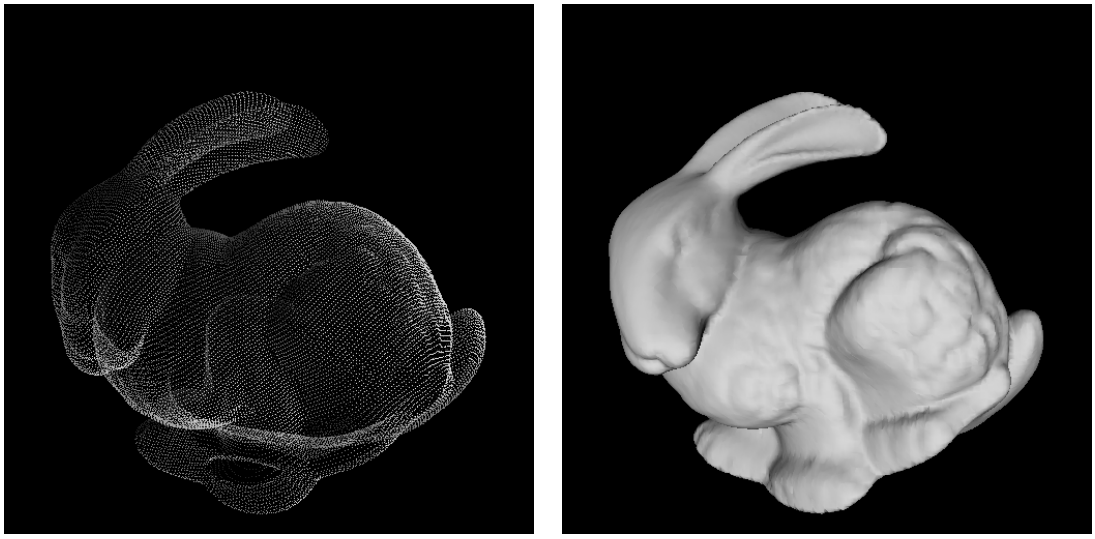
The use of the partition of unity method to reconstruct implicit surfaces from unorganized point sets enables to design *point-based modeling* environments, where points are used as modeling primitives, as an alternative to modeling techniques based on polygonal meshes. The pioneering work for point-based modeling can be attributed to Turk and O'Brien [156, 157], who interpolate global variational implicit surfaces from small unorganized point sets. Very recently, Pauly et al. have shown how to use the approximation power of moving least squares to define a point-based free-form shape modeling technique [122].

Implicit surfaces reconstructed using the partition of unity variational method is particularly adapted for free-form modeling with a large number of points as modeling primitives. After every change of the positions or normals of any of the points, an implicit surface is reconstructed efficiently by simply updating the reconstructions in the involved local subdomains. This can be done either by solving the linear systems in the involved subdomains from scratch, or by using the previous solutions of the local reconstruction problems to determine the new solutions with the Sherman-Morrison formula [128].

In Figure 22, we illustrate some simple examples of low-level modeling on a sphere. The initial implicit surface was reconstructed using the partition of unity variational method from a set of 42 seed points (Figure 22(a)). The results of some basic low-level operations such as changing a point's position, changing a point's normal, as well as adding a new point can be seen in Figures 22(b) - 22(d), respectively.

Since we allow a high number of unorganized points for modeling, we believe that it will be useful to define modeling operators on several modeling points at a time, where, for example, all modeling points are influenced with an impact according to a decay function.

High-level operators that act simultaneously on a large number of modeling points can be defined using *space deformation techniques* that define geometric deformations as a general transformation from $\mathbb{R}^3$ to $\mathbb{R}^3$, such as the warping operator by Barr, the free-form deformation operator by Sederberg and Parry, and others, see [18] for a survey. As our point-based modeling primitives are totally defined by a set of 3D points, all these techniques can be easily used,



(a) Twisted points. (b) Reconstructed surface.

Figure 23: Twisting the Stanford Bunny around the z-axis.

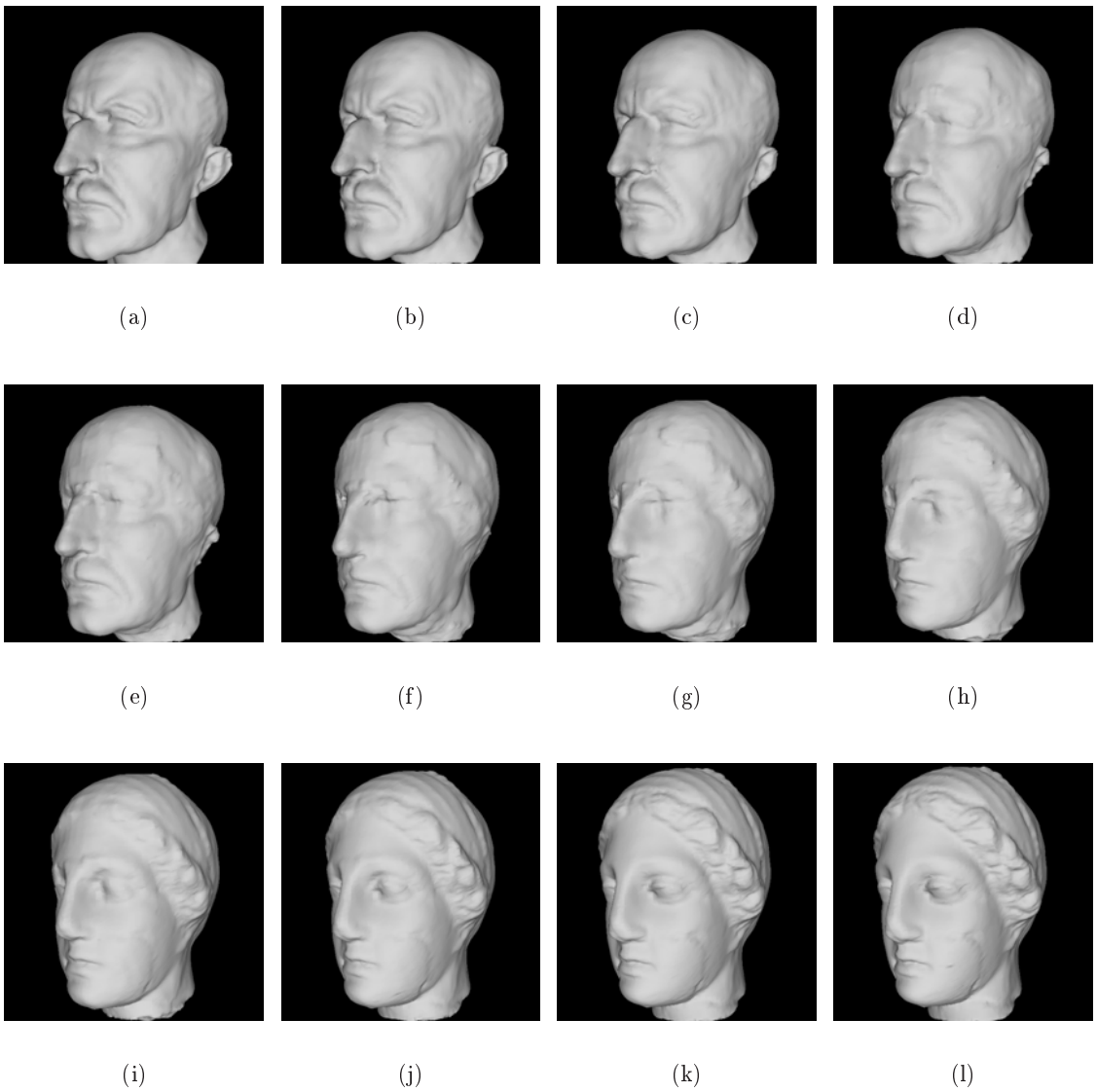
and by using either the partition of unity variational method or the hierarchical partition of unity variational method, topology issues do not have to be considered. For instance, Figure 23(a) presents the application of a classical squeezing operator applied to the unorganized points of the Stanford Bunny, and the resulting reconstructed surface using the partition of unity variational method is shown in Figure 23(b).

Since the defining function is at least C^0 continuous and has negative values outside the surface and positive values inside, the *function representation model* (FRep model), a powerful solution for point membership classification [121], can be applied to the partition of unity and hierarchical partition of unity variational methods. Function-based shape modeling operations are usually built in a constructive approach, resulting in a tree structure, with defining functions at the leaves and operations at the nodes.

Morphing between two implicit surfaces reconstructed by the partition of unity variational method or the hierarchical partition of unity variational method with defining functions f_1 and f_2 can simply be done by the following linear interpolation with $c \in [0, 1]$:

$$f_{morph}(\boldsymbol{x}) = (1 - c)f_1(\boldsymbol{x}) + cf_2(\boldsymbol{x}) \tag{50}$$

By starting with $c = 0$ and by gradually increasing c up to $c = 1$, the implicit surface defined by f_{morph} gradually metamorphoses the implicit surface defined by f_1 into the implicit surface defined by f_2. See Figure 24 for an example of the metamorphosis of the Max Planck head into the Cyberware Igea for 12 different values for c.



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

(e) (f) (g) (h)

(i) (j) (k) (l)

Figure 24: Morphing the Max Planck head to the Cyberware Igea.

Chapter 3

The Hierarchical Partition of Unity Variational Method

3.1 Overview

In this chapter, we present a hierachical way to use the partition of unity method for variational implicit surfaces in order to reconstruct implicit surfaces from a large unorganized point set $\mathcal{P}$. In contrast to the partition of unity variational method introduced in the preceding chapter, where the local reconstructions of all the leaf nodes of the domain decomposition are glued together using the partition of unity method, the *hierarchical partition of unity variational method* uses the partition of unity method also at the inner nodes of the hierarchy. The solutions are sticked together by pairs and propagated bottom-up, and the root node contains the global reconstruction function. Moreover, instead of fixed grid or octree domain decompositions, a binary tree is used to create the local subdomains. The particular charme of the new method is, that the local reconstruction is calculated from the same number of points in every local subdomain. Furthermore, the number of points in the *overlapping zone* can be specified, ensuring a useful constraint to stabilize the partition of unity blending of the local subdomains.

Again, we decompose the method into three principal steps. First, the global domain of interest is subdivided into overlapping subdomains using a binary tree, where every subdomain contains the same number of points (Section 3.2). Second, the local reconstructions of the subdomains in the leaf nodes of the binary tree are computed according to Section 2.4. Third, in order to evaluate the global defining function, the local reconstructions are glued together bottom-up by recursively applying the partition of unity method (Section 3.4).

After illustrating the hierarchical partition of unity variational method on a simple example in 2D in Section 3.5, we present the results of this new method in Section 3.6 and conclude in Section 3.7.

3.2 Binary Tree Domain Decomposition

Starting from the unorganized point set $\mathcal{P}$ and its bounding box being the entire domain $\Omega^{[0]}$, the binary tree domain decomposition method adaptively subdivides $\Omega^{[0]}$ using a binary

tree. The tree is built in a top-down recursive process starting from the root node, where the entire domain $\Omega^{[0]}$ is subdivided into two overlapping subdomains $\Omega_1^{[1]}$ and $\Omega_2^{[1]}$ containing an equal number of points $n^{[1]}$ in the respective point sets $\mathcal{P}_1^{[1]}$ and $\mathcal{P}_2^{[1]}$. All subdomains $\Omega^{[l]}$ at level l are themselves subdivided recursively into two overlapping subdomains $\Omega_1^{[l+1]}$ and $\Omega_2^{[l+1]}$ containing an equal number of points $n^{[l]}$ in the respective point sets $\mathcal{P}_1^{[l+1]}$ and $\mathcal{P}_2^{[l+1]}$. The recursion terminates, when the number of points in a subdomain falls below a threshold $2 * T_{leaf}$. In this way, a *perfect binary tree* of level L is established, i.e. all leaf nodes are at the same level L in the tree, and all internal nodes have the degree two. Furthermore, in all 2^L leaf nodes, the equal number of points $n^{[L]}$ is restricted to $T_{leaf} \leq n^{[L]} < 2T_{leaf}$.

Let us now give some insight how to subdivide a domain $\Omega^{[l]}$ into two overlapping subdomains $\Omega_1^{[l+1]}$ and $\Omega_2^{[l+1]}$ containing an equal number of points $n^{[l]}$ in the respective point sets $\mathcal{P}_1^{[l+1]}$ and $\mathcal{P}_2^{[l+1]}$.

In contrast to the octree domain decomposition method described in Section 2.3.2, the number of points $n_{overlap}^{[l]} = Card(\mathcal{P}_1^{[l+1]} \cap \mathcal{P}_2^{[l+1]}))$ in the overlapping zone $\Omega_1^{[l+1]} \cap \Omega_2^{[l+1]}$ can be specified explicitly as an *overlap quota* $q \in]0,1[$ of the number of points $n^{[l]}$:

$$n_{overlap}^{[l]} = qn^{[l]} \tag{51}$$

In order to avoid a too low or too high number of points in $\Omega_1^{[l+1]} \cap \Omega_2^{[l+1]}$ at the interior nodes of the binary tree resulting in either an undesired behavior when sticking together the functions of the local subdomains, or in a too high computational overhead, we clamp $n_{overlap}^{[l]}$ to $[T_{min}, T_{max}]$. Then, the number of points $n^{[l+1]}$ in the subdomains can be calculated as follows:

$$n^{[l+1]} = \left\lceil \frac{n_{overlap}^{[l]} + n^{[l]}}{2} \right\rceil \tag{52}$$

The extent of the two overlapping subdomains $\Omega_1^{[l+1]}$ and $\Omega_2^{[l+1]}$ is calculated by the following steps. First, the longest axis of $\Omega^{[l]}$ is determined. Second, we collect the point sets $\mathcal{P}_1^{[l+1]}$ (respectively $\mathcal{P}_2^{[l+1]}$) containing the $n^{[l+1]}$ points with the lowest (respectively highest) values with respect to the longest axis. In practice, we rearrange the points $\mathcal{P}^{[l]}$ with respect to their values of the longest axis. By setting $i_1 = n^{[l+1]}$ and $i_2 = n^{[l]} - n^{[l+1]} + 1$, we rearrange the points so that $\boldsymbol{p}_i < \boldsymbol{p}_{i_1}$ for $1 \leq i \leq i_1$ and $\boldsymbol{p}_i > \boldsymbol{p}_{i_2}$ for $i_2 \leq i \leq n^{[l]}$:

$$\mathcal{P}^{[l]} = \overbrace{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{i_2-1}, \underbrace{\boldsymbol{p}_{i_2}, \ldots, \boldsymbol{p}_{i_1}}_{\mathcal{P}_2^{[l+1]}}}^{\mathcal{P}_1^{[l+1]}}, \boldsymbol{p}_{i_1+1}, \ldots, \boldsymbol{p}_{n^{[l]}}}$$

Finally, $\Omega_1^{[l+1]}$ and $\Omega_2^{[l+1]}$ are defined by the bounding boxes of $\mathcal{P}_1^{[l+1]}$ and $\mathcal{P}_2^{[l+1]}$, respectively. Consider the resulting recursive algorithm for the binary tree domain decomposition in Algorithm 2 that has to be called using `decompose(`$\mathcal{P}$`, 0)`.

In the leaf nodes, the local reconstructions are calculated according to the partition of unity variational method resulting in defining functions $f^{[L]}$. See Section 2.4 for more details about the local reconstructions.

Algorithm 2 decompose($\mathcal{P}$, l)

Require: points $\mathcal{P}^{[l]}$, level l
Ensure: binary tree
  set $\Omega^{[l]}$ be the bounding box of $\mathcal{P}$
  set $n^{[l]} = \text{Card}(\mathcal{P}^{[l]})$
  if $n^{[l]} >= 2T_{leaf}$ **then**
    set $n^{[l]}_{overlap} = qn^{[l]}$
    if $n^{[l]}_{overlap} < T_{min}$ **then**
      $n^{[l]}_{overlap} = T_{min}$
    else if $n^{[l]}_{overlap} > T_{max}$ **then**
      $n^{[l]}_{overlap} = T_{max}$
    end if
    determine the longest axis of $\Omega^{[l]}$
    rearrange the points in $\mathcal{P}^{[l]}$ according to the longest axis
    determine the points $\mathcal{P}_1^{[l+1]}$ and $\mathcal{P}_2^{[l+1]}$
    decompose($\mathcal{P}_1^{[l+1]}$, l+1)
    decompose($\mathcal{P}_2^{[l+1]}$, l+1)
  else
    Calculate the local reconstruction for $\mathcal{P}^{[l]}$
  end if

3.3 Sticking Solutions Together

In order to evaluate the global defining function f at a point $\boldsymbol{x}$, the local reconstructions are glued together by recursively applying the partition of unity method bottom-up in the binary tree. More precisely, at every *internal node* of level l, the defining function $f^{[l]}$ is given by a partition of unity blending of the defining functions of the two child nodes $f_1^{[l+1]}$ and $f_2^{[l+1]}$:

$$f^{[l]}(\boldsymbol{x}) = \frac{f_1^{[l+1]}(\boldsymbol{x})\hat{w}_1^{[l+1]}(\boldsymbol{x}) + f_1^{[l+1]}(\boldsymbol{x})\hat{w}_2^{[l+1]}(\boldsymbol{x})}{\hat{w}_1^{[l+1]}(\boldsymbol{x}) + \hat{w}_2^{[l+1]}(\boldsymbol{x})} \tag{53}$$

Recall that the weighting functions $\hat{w}_1^{[l+1]}$ (and $\hat{w}_2^{[l+1]}$) at level $l+1$ are constructed as the composition of a distance function $d_1^{[l+1]}$) (respectively $d_2^{[l+1]}$) and a decay function υ, i.e. $\hat{w}_1^{[l+1]}(\boldsymbol{x}) = \upsilon \circ d_1^{[l+1]}(\boldsymbol{x})$ (respectively $\hat{w}_2^{[l+1]}(\boldsymbol{x}) = \upsilon \circ d_2^{[l+1]}(\boldsymbol{x})$), as already explained in detail in Section 2.5.

Of course, since the domains $\Omega_1^{[l+1]}$ and $\Omega_2^{[l+1]}$ are defined as the bounding boxes of the point sets $\mathcal{P}^{[l]}$ spanned from $\boldsymbol{a}$ and $\boldsymbol{b}$, the distance function to use is a box distance function:

$$d_i^{box}(x,y,z) = 1 - \prod_{v \in x,y,z} \frac{4(v-a_v)(b_v-v)}{(b_v-a_v)^2} \tag{54}$$

To illustrate the evaluation of the global defining function f at a point $\boldsymbol{x}$, consider the recursive Algorithm 3 that has to be called with `eval(`$\boldsymbol{x}$`, root node)`.

Algorithm 3 eval($\boldsymbol{x}$, node `node`)

Require: point $\boldsymbol{x}$, node `node` at level l
Ensure: defining function value of $\boldsymbol{x}$ at node `node` of level l

```
if p ∈ Ω^[l] then
  if node is a leaf node then
    return f^[L](x)
  else
    set f_1^[l+1] = eval(x, node->child[0])
    set f_2^[l+1] = eval(x, node->child[0])
    calculate ŵ_1^[l+1] from Ω_1^[l+1]
    calculate ŵ_2^[l+1] from Ω_2^[l+1]
    return (f_1^[l+1](x) ŵ_1^[l+1](x) + f_1^[l+1](x) ŵ_2^[l+1](x)) / (ŵ_1^[l+1](x) + ŵ_2^[l+1](x))
  end if
else
  if node is the root node then
    return some negative value
  else
    return 0
  end if
end if
```

3.4 Scalability

In this section, we analyze the scalability of the hierarchical partition of unity variational method in terms of computational complexity. We are interested in the scalability of both the reconstruction of the global defining function f starting from a point set $\mathcal{P}$ consisting of N points, and the complexity analysis of the evaluation of the global defining function f at a point $\boldsymbol{x} \in \mathbb{R}^3$.

Concerning the reconstruction of the global defining funtion f, the following principal steps are involved:

1. Determine the domain $\Omega^{[l]}$ from the bounding box of the point set $\mathcal{P}^{[l]}$ and recursively subdivide it into overlapping domains after collecting the point sets $\mathcal{P}_1^{[l+1]}$ and $\mathcal{P}_2^{[l+1]}$.

2. Compute the local defining functions $f^{[L]}$ for all local subdomains $\Omega^{[L]}$ in the 2^L leaf nodes.

The first step is the recursive domain decomposition process. In order to analyze the scalability, we need to determine the number of points per level as well as the number of levels L.

Of course, the number of points in the root node is $N = Card(\mathcal{P})$. Using equations (51) and (52), the number of points for a node at level l is $n^{[l]} = \frac{N(1+q)^l}{2^l}$, and since there are 2^l nodes per level, the number of points per level is $N(1+q)^l$. Since the number of points in the overlapping zone is bounded above by T_{max}, the number of points per node at level $0 \leq l \leq L_1$ is bounded above by $\frac{N}{2^l} + \frac{2^l-1}{2^{l-1}} T_{max}$ with $L_1 = \lceil \log_2 N \rceil$. Furthermore, at level L_1, the number

of points per node is $n^{[L_1]} \leq 2T_{max} + 1$. By applying Equation (52) for all levels $l > L_1$, there are at most $\frac{2T_{max}(1+q)^{l-L_1}}{2^{l-L_1}}$ points per node for $L_1 < l \leq L$. Since the recursion terminates for all nodes $n^{[l]} < 2T_{leaf}$, the number of levels L is bounded above, and hence the number of points $n^{[L]}$ at level L is bounded above as well, with the constants α and β:

$$L \leq \lceil \log N \rceil + \left\lceil \frac{\log \frac{T_{leaf}}{2T_{max}}}{\frac{1+q}{2}} \right\rceil = \alpha + \lceil \log N \rceil \tag{55}$$

$$n^{[L]} \leq 2^{\left\lceil \frac{\log \frac{T_{leaf}}{2T_{max}}}{\frac{1+q}{2}} \right\rceil} * N = \beta N \tag{56}$$

Since we know that for $l_1 > l_2$ we have $n^{[l_1]} > n^{[l_2]}$, the total number of points in all nodes of the binary tree is bounded by $\beta N \alpha \lceil \log N \rceil$ and is hence in $O(N \log N)$.

At every node of the binary tree, it is understood that the bounding box as well as the longest axis can be determined in linear time with respect to the number of points in the node. According to Sedgewick [142], the points can also be rearranged in linear time with respect to the longest axis. Since the total number of points of all nodes in the binary tree is bounded by $O(N \log N)$, the binary tree can be created in $O(N \log N)$.

The second step involves the computation of the local reconstructions in all 2^L leaf nodes. Finding the local reconstructions in a leaf node is in $O(1)$, since the number of points $n^{[L]}$ per leaf node is bounded by the constant T_{leaf}. Moreover, since the number of leaf nodes is proportional to N, all local reconstructions can be resolved in $O(N)$ time.

Concerning the evaluation of the global defining function f in a point $\boldsymbol{x}$, the following three steps are involved:

1. Find all local subdomains $\Omega^{[L]}$ with $\boldsymbol{x} \in \Omega^{[L]}$.
2. Evaluate all local defining functions $f^{[L]}$ and the corresponding weighting functions $\hat{w}^{[L]}$.
3. Propagate the evaluations of the local defining functions bottom-up in the binary tree by using a partition of unity blending by pairs.

Due to the fact that there is only a constant number of regions including the point $\boldsymbol{x}$, the first and third step, i.e. traversing the binary tree from the root node and propagating the evaluations bottom up, require $O(\log N)$ time. The evaluation of one local defining function can be done in constant time because the number of points is bounded by T_{leaf}, and since there is a constant number of concerned subdomains for $\boldsymbol{x}$, step two can be done in $O(1)$.

Summing up, the implicit surface can be reconstructed in $\mu_{rec} O(N \log N) + \nu_{rec} O(N) = O(N \log N)$ time, and the global defining function f can be evaluated in $\mu_{eval} O(\log N) + \nu_{eval} O(1) = O(\log N)$ time. Consequently, the hierarchical partition of unity variational method is efficient according to the definition of Schaback [139].

Similar to the partition of unity variational method, we experienced a better scalability in the hierarchical partition of unity variational method in practice. During the surface reconstruction and evaluation, the constants μ_{rec} and μ_{eval} are very small compared to the constants ν_{rec} and nu_{eval}, and the results in Section 3.6 show an overall linear reconstruction and constant evaluation time with respect to the number of points N.

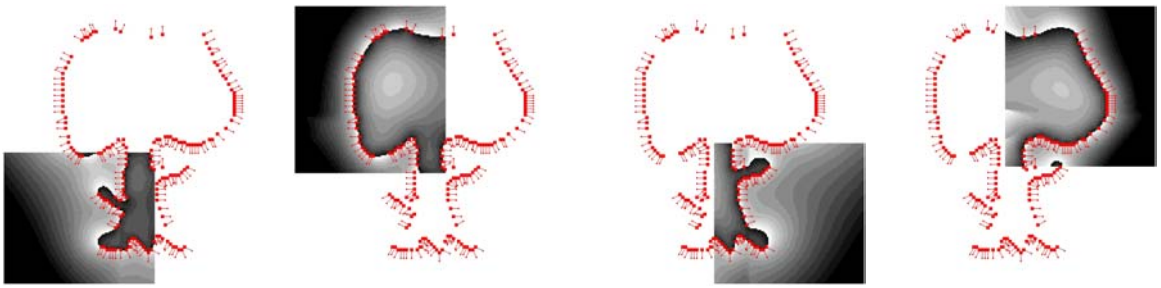
3.5 Example

In this section, we show two examples of the hierarchical partition of unity variational method to illustrate the domain decomposition method using the binary tree as well as the stability against differences in topology of the subdomains.

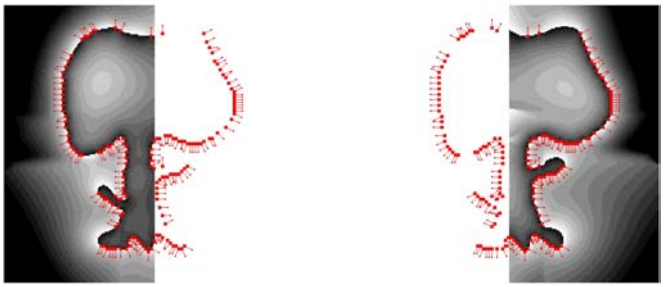
First, a simple example illustrates the domain decomposition method using the binary tree in the hierarchical partition of unity method in 2D (Figure 25). All function values are represented by grayscale values according to the convention $\left[\begin{smallmatrix}-\infty\\ black\end{smallmatrix} \dashrightarrow \begin{smallmatrix}0^-\\ white\end{smallmatrix} \middle| \begin{smallmatrix}0^+\\ black\end{smallmatrix} \dashrightarrow \begin{smallmatrix}+\infty\\ white\end{smallmatrix}\right]$.

The local reconstructions in the four leaf nodes of the binary tree can be seen in Figure 25(a). The result of blending together the leaf nodes using the partition of unity method can be seen in Figure 25(b). The global defining functions of the root node of the binary tree and a straightforward global variational implicit surface are sketched in Figure 25(c) and Figure 25(d), respectively. Note that the results are similar near the zero-set of the defining functions where we are interested in. They may differ elsewhere without having an impact on the reconstructed zero-set.

A second example in 2D illustrates the stability of the hierarchical partition of unity variational method when sticking together two subdomains with radically different topologies. To this end, consider the unorganized point set in 2D in Figure 26(a). The local reconstructions in the two subdomains are shown in Figures 26(b) and 26(c). The resulting partition of unity blending is shown in Figure 26(d), and no noticeable differences can be seen compared to the global reconstruction shown in Figure 26(e).



(a) Local reconstructions in the leaf nodes.

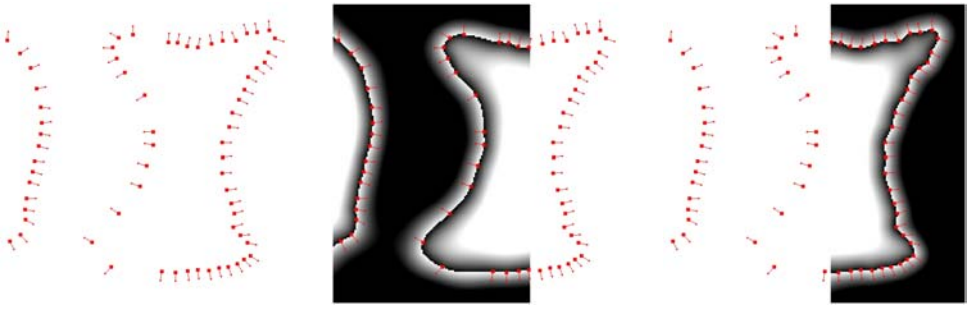


(b) Partition of unity blending of the leafs' local reconstructions.

(c) Global defining function in the root node.

(d) Defining function of the global variational implicit surface.

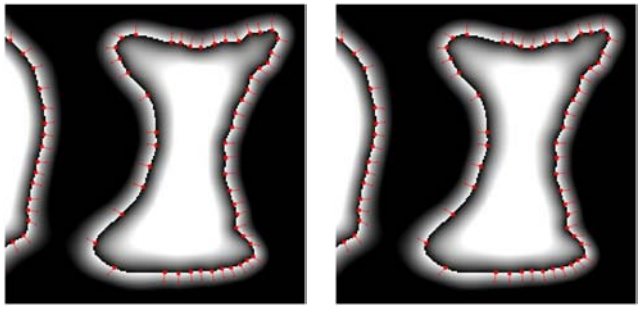
Figure 25: 2D example of the hierarchical partition of unity variational method compared to a global variational implicit surface (all function values are represented by grayscale values according to the convention $\left[\begin{smallmatrix} -\infty \\ black \end{smallmatrix} \dashrightarrow \begin{smallmatrix} 0^- \\ white \end{smallmatrix} \middle| \begin{smallmatrix} 0^+ \\ black \end{smallmatrix} \dashrightarrow \begin{smallmatrix} +\infty \\ white \end{smallmatrix} \right]$).



(a) Seed point set in 2D.

(b) Local reconstruction on the left side.

(c) Local reconstruction on the right side.



(d) Partition of unity blending of the left and right local reconstructions.

(e) Global reconstruction.

Figure 26: Stability against topological differences (all function values are represented by grayscale values according to the convention $\left[\begin{smallmatrix} -\infty \\ black \end{smallmatrix} \dashrightarrow \begin{smallmatrix} 0^- \\ white \end{smallmatrix} \middle| \begin{smallmatrix} 0^+ \\ black \end{smallmatrix} \dashrightarrow \begin{smallmatrix} +\infty \\ white \end{smallmatrix}\right]$).

3.6 Results

We reconstructed implicit surfaces using the hierarchical partition of unity variational method from various point sets from different sources, and we show some results in this section. As for the partition of unity variational method, all results were obtained on an Intel Pentium 1.7 GHz with 512 MB of RAM running Linux. Again, the linear systems involved in the local reconstructions were solved using the linear solver from the GNU Scientific Library package [61] based on LU-decomposition. For the local reconstructions, we exlusively used biharmonic basic functions $\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|$ since we experienced that they are more stable ingainst numerical instabilities.

For a first type of point sets, we take the vertices of a polygonal mesh and discard all connectivity information. In order to show the scalability of the hierarchical partition of unity variational method, we reduced the Cyberware Isis (187,645 points) and the original Stanford Dragon (437,645 points) arbitrarily to N points. In Table 12, we show the required reconstruction time t_{rec} as the total of the binary tree creation time t_{tree} and the required time t_{rec} for the local reconstructions with two normal constraints per point. We experienced the best quality-speed trade-off by setting $T_{min} = 30$, $T_{max} = \infty$, and $T_{leaf} = 50$ as well as an overlap quota $q = 0.05$, and the resulting number of leaf nodes is given by 2^L and the equal number of points per leaf node is given by $n^{[L]}$. We also sketched the graph of the total reconstruction time t_{total} with respect to the number of points N in Figure 27. It can be seen, that the given total reconstruction times of the Cyberware Isis increase by leaps and bounds with respect to the number of points N. This is due to the fact, that the total reconstruction time can be inferior even for a higher number of points N since the number of points per leaf node $n^{[L]}$ can decrease, and the complexity of the local reconstructions is in $O((n^{[L]})^3)$. But in a general view, there is a linear behavior with respect to the number of points N, since the $O(N \log N)$ creation of the binary tree is negligible compared to the $O(N)$ solving of the involved local reconstructions as can be seen in Figure 27(a) and 27(b).

Some visual results of the reconstructed implicit surfaces can be seen in Figure 28.

In order to illustrate the impact of the overlap quota q as well as the minimum number of leaf nodes T_{leaf} on the total reconstruction time t_{total}, see Table 13 for some different values on the Stanford Bunny (34,834 points). For a too low overlap quota $q = 0.01$, artefacts appear as can be seen in Figure 29.

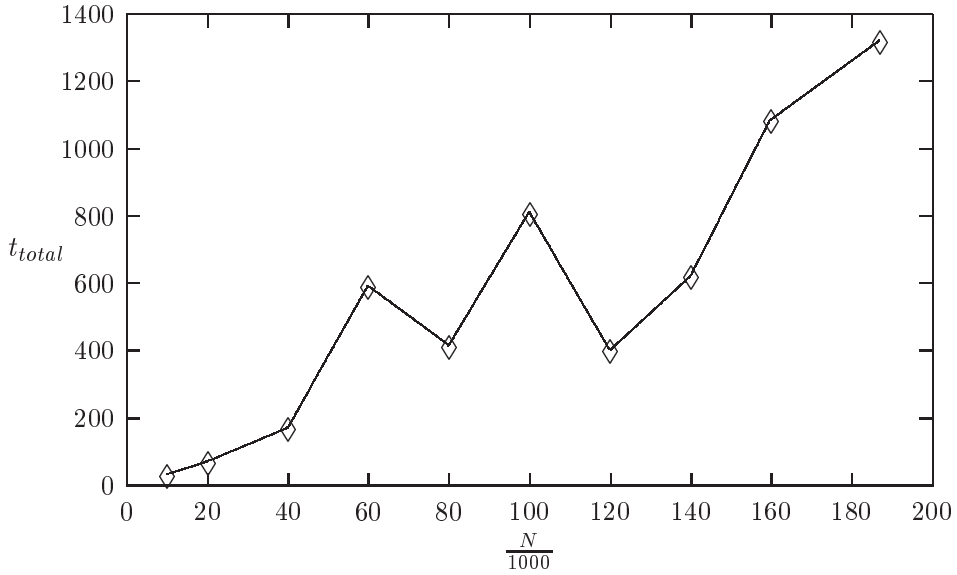
As the partition of unity variational method, the hierarchical partition of unity variational method is particularly adapted to reconstruct implicit surfaces from non-uniformly distributed point sets. To illustrate the robustness against highly non-uniformly distributed point sets, we reconstructed implicit surfaces from the same artificially created density varying subsets of points of the Stanford Bunny and the Stanford Dragon as in Chapter 2. The results can be seen in Figures 30(a) and 30(b).

Model	N	2^L	$n^{[L]}$	t_{tree}	t_{total}
Isis	10,000	256	60	0.4	34
	20,000	512	63	1.0	72
	40,000	1,024	66	2.5	173
	60 000	1,024	96	4.1	593
	80,000	2,048	69	6.0	416
	100 000	2,048	85	8.1	812
	120 000	4,096	56	10.5	403
	140 000	4,096	64	12.8	623
	160,000	4,096	72	14.4	1087
	187,645	4,096	84	16.0	1323
Dragon	50,000	1,024	81	3.3	362
	100,000	2,048	85	8.6	935
	200,000	4,096	89	18.7	2114
	400,000	8,192	93	43.2	4360

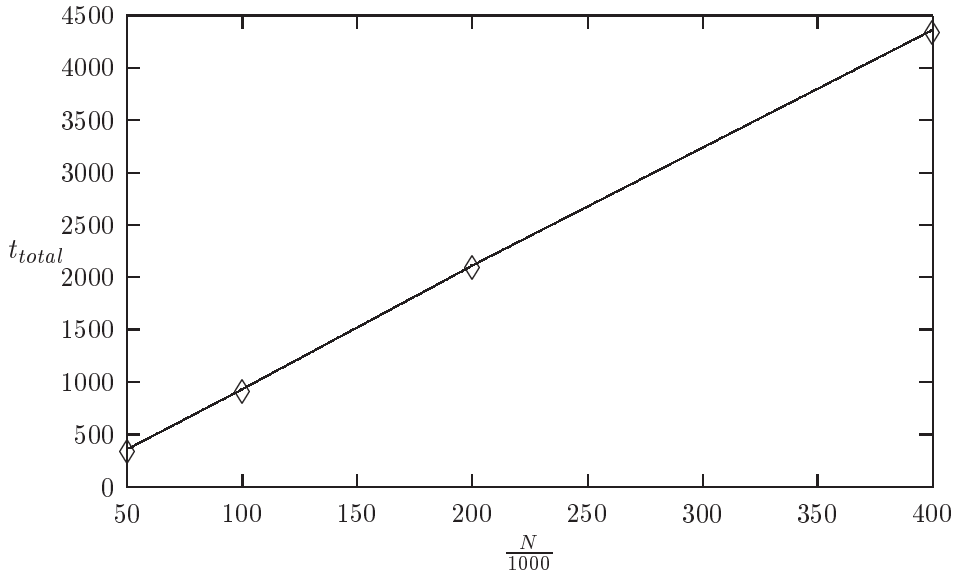
Table 12: Scalability of the hierarchical partition of unity variational method (timings in seconds).

Model	T_{leaf}	q	2^L	$n^{[L]}$	t_{total}
Bunny	50	0.01	1,024	46	54
		0.02	1,024	48	59
		0.05	2,048	34	46
		0.1	4,096	30	66
	100	0.01	512	81	171
		0.02	512	86.	211
		0.05	1,024	58	109
		0.1	1,024	89	474
	200	0.01	256	152	633
		0.02	256	162	799
		0.05	512	106	417

Table 13: Total reconstruction time with varying overlap quota q and minimum number of points per leaf node T_{leaf}.

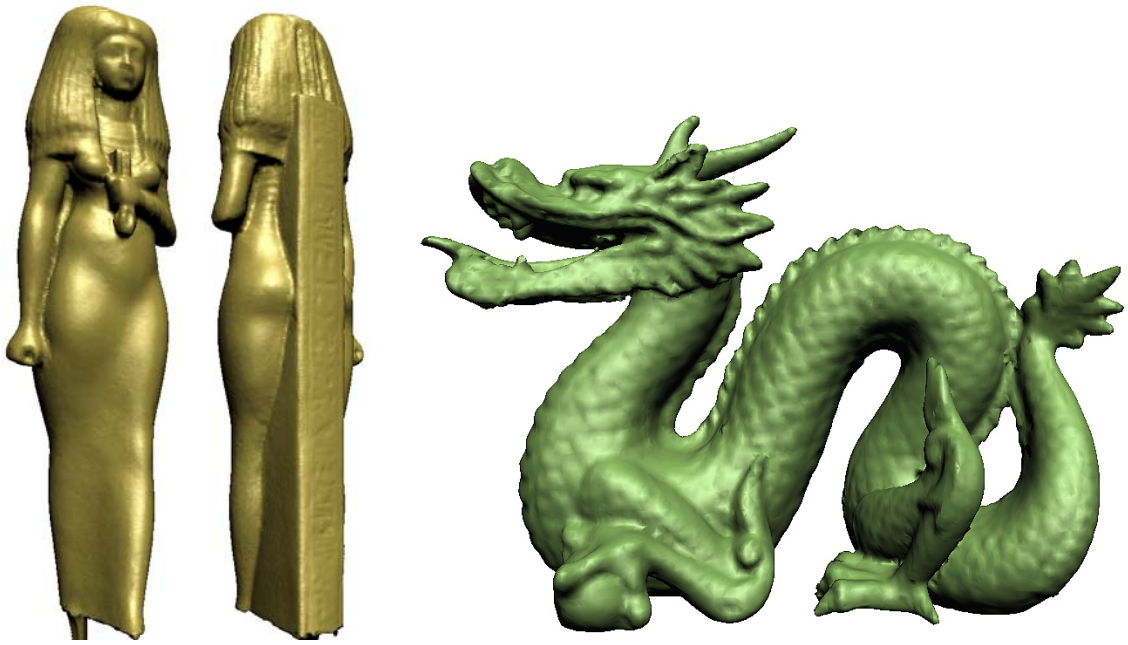


(a) Isis



(b) Stanford Dragon

Figure 27: Reconstruction time in seconds with respect to the number of points.



(a) Isis - 160,000 points.

(b) Dragon - 200,000 points.

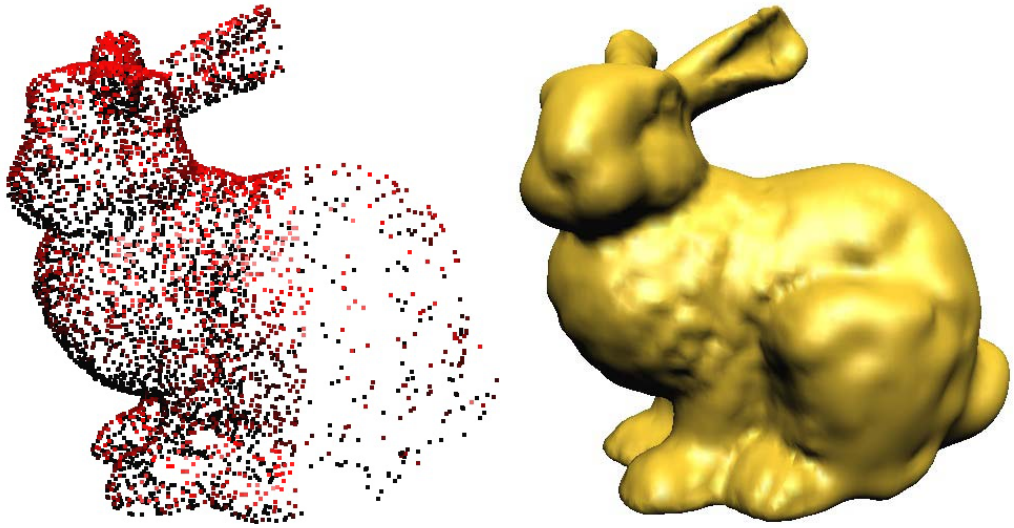
Figure 28: Visual results of the hierarchical partition of unity variational method.

(a) $q = 0.01$.

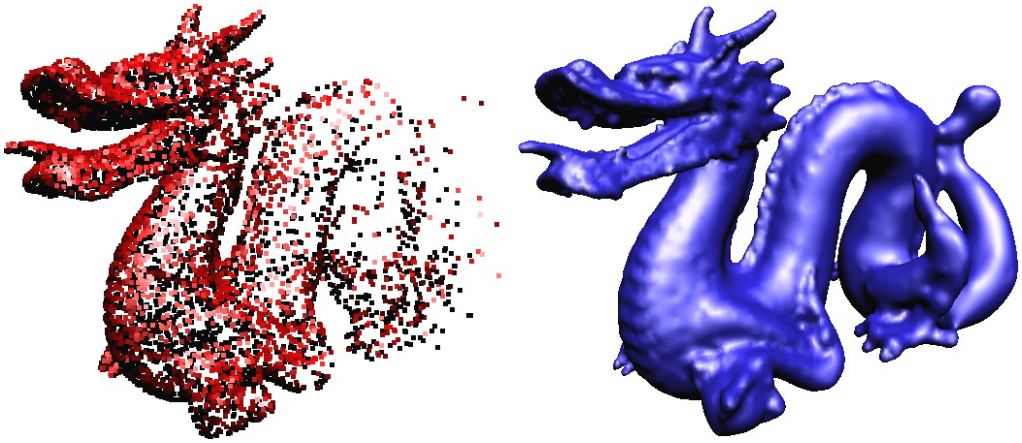
(b) $q = 0.02$.

(c) $q = 0.05$.

Figure 29: Reconstruction quality for different values of the overlap quota q.



(a) Non-uniform point set with hard density variation.



(b) Non-uniform point set with smooth density variation.

Figure 30: Robustness of the hierarchical partition of unity variational method.

3.7 Conclusions

In this chapter, we described the hierarchical partition of unity variational method, another new method to reconstruct implicit surfaces from large unorganized point sets. This method divides the global reconstruction domain into smaller overlapping local subdomains using a binary tree, solves the reconstruction problems in the local subdomains using radial basis functions with global support, and blends the solutions together by hierarchically applying the partition of unity method at all inner nodes of the hierarchy. Consequently, the local reconstructions are blended together by pairs and propagated bottom-up, and the root node contains the global reconstruction function.

Of course, the hierarchical partition of unity variational method has many similarities with the partition of unity variational method presented in Chapter 2. The method also has a nice linear behavior of the required reconstruction time and memory usage with respect to the number of points in the point set, enables an out-of-core implementation due to the independence of the local subdomains, is robust against topological differences between blended subdomains, and does not require a carrier solid.

However, an important advantage of the hierarchical partition of unity variational method compared to the partition of unity variational method is, that the local reconstructions in the leaf nodes are computed from an equal number of points. Furthermore, the number of points in the overlapping zone can be specified explicitly while further increasing the stability of the implicit surface reconstruction. This makes the hierarchical partition of unity variational method even more robust against highly non-uniformly distributed and topologically complex point sets.

Part II

Rendering of Implicit Surfaces from Large Unorganized Point Sets

After the acquisition of the unorganized point set $\mathcal{P}$, there are different techniques to render it on the screen in order to get a visual feedback. In this part, we will discuss different rendering techniques for implicit surfaces reconstructed from unorganized points. We start by reviewing some previous work in Chapter 4 that we divide into three categories.

The first category of previous work describes techniques that are using the unorganized point set $\mathcal{P}$ directly for visualization. The two major techniques of this first category are called *point-based rendering*, where the points are mapped by *forward warping* from 3D space to 2D image locations, and *point-based ray-tracing*, where rays passing through the 2D image locations and the viewpoint are intersected with the points, also called *backward warping*.

The second category of previous work to be described are techniques to visualize implicit surfaces in general, since we have shown how to reconstruct an implicit surface $\mathcal{S}$ from the unorganized point set $\mathcal{P}$ in Part I of this thesis.

Finally, the third category of previous work are techniques to visualize the unorganized point set $\mathcal{P}$ that are using information from both the unorganized point set $\mathcal{P}$ as well as its reconstructed implicit surface $\mathcal{S}$.

In Chapter 5, we present a new point-based rendering technique for an unorganized point set that uses a reconstructed implicit surface as well. The implicit surface is rendered view-dependently in an output-sensitive multiresolution manner using points as rendering primitive without the creation of a polygonal mesh representation.

In Chapter 6, we present another point-based rendering technique for implicit surfaces reconstructed from unorganized points that is running on programmable graphics hardware. The local differential geometry for every point in the unorganized point set is extracted in a preprocess, and then every point is rendered as a fragment-shaded rectangle.

Chapter 4

Previous Work

4.1 Point-based Rendering and Ray-Tracing

4.1.1 Point-based Rendering

Motivation

Point-based rendering has become very popular in the last years due to the evolution of the rendering power of modern 3D graphics hardware and due to the fact, that extremely large models from laser range scanners become available [99, 42, 108]. In contrast to traditional rendering techniques developed as far back as the middle of the 1970's, where 3D objects are approximated by sets of polygons (*polygonal meshes*), rendered using a depth buffer [35], and shaded during projection using a model according to Gouraud [65] or Phong [125], point-based rendering techniques are rendering unorganized points without explicit connectivity. There are several reasons for the success of point-based rendering techniques. First, the huge memory available on modern graphics hardware has made it possible to treat scenes with dramatically more polygons, and as a consequence, by optimally using the rendering pipeline, projected polygons are mapped to less than one screen pixel. So the main advantage of scan-line rendering, i.e. the incremental calculation of a polygon's inner points, has mostly vanished for scenes of high complexity. Second, representing a curved surface connected planar polygons requires information about connectivity and remains only an approximation of the curved surfaces. During point-based rendering the connectivity has no longer to be managed, and some common operations such as level-of-details, geometrical deformation, and topology modifications are much easier to implement.

History of Origins

Let us now give some insight into the evolution of point-based rendering techniques. The first use of 3D discrete points as rendering primitives in computer graphics can be attributed to Reeves in 1983, with his famous concept of particle systems [129]. According to Reeves, a particle is simply a point in the 3D Euclidean space, combined with some additional information, such as color, density, shading or scattering coefficients. The main advantage of particle systems is that the rendering step becomes trivial: projecting each particle on the screen, checking it for visibility with the depth buffer, and finally shading the corresponding

pixel using the color stored in the particle. Particle systems have mainly been used as a convenient modeling primitive to generate and animate specific objects that are hard to manage by conventional geometric models: fire and explosion [129], smoke [45], waterfalls [146], clouds [22], and fluids [45, 110]. Original particle systems, based on what we propose to call *isotropic particles*, were developed for modeling and rendering volumetric models. Szelisky and Tonnesen [149] adapted particle systems to surface models, by introducing the concept of *oriented particles* in 1992, based on work done by Reynolds [134]. In this technique, each particle represents a sample of an underlying surface, and is associated to a 3D frame that represents the local behavior by the normal vector and the tangent plane of this surface.

An early approach exploiting the possibility to use particles to render solid objects was first investigated by Levoy and Whitted in 1985 [100]. Their idea is to sample a solid object into a set of 3D points with a sufficient density to give the visual impression of a solid object when individual points are projected on the screen. A similar idea was proposed by Max [106] to render trees.

At the time of all these early approaches, existing computer resources were not feasible to render a large amount of points in *real-time*, i.e. to be able to render them in a *frame rate* keeping up with the human eye perception. But when Shade et al. presented layered depth images [143] and Grossman and Dally presented an efficient point-based rendering technique called point sample rendering [69, 68] simultaneously in 1998, the first scenes using points as rendering primitive were rendered in real-time, resulting in a highly growing interest towards point-based rendering recently.

We classify the point-rendering techniques according to the characteristic, whether there is knowledge about the surface where the points are sampled from or not. First, we discuss point-rendering techniques, where there is knowledge about the surface and the sampling can be surface-driven, and second, we discuss point-rendering techniques that do not require any other underlying surface representation.

Knowledge about the Surface

When there is some knowledge about the underlying surface, we consider that discrete points are used because they can be rendered more efficiently than other primitives gained from an underlying surface representation. We will first review some previous work about point rendering completely implemented in software before reviewing work that is making use of current graphics hardware pipelines.

Shade et al. generate a *layered depth image (LDI)* [143], i.e. a view of a scene from a single viewpoint with multiple pixels along each line of sight, from any other surface representation. The front element of a *layered depth pixel* samples the first surface along the line of sight, and the following pixels sample the next surfaces. To render the layered depth image from arbitrary viewpoints, the layered depth pixels are warped to image space efficiently using an incremental calculation, and they are rendered as rectangular gaussian *splats*, i.e. an associated image raster parallel to the viewport. The name splat comes from the colorful analogy of throwing a snowball against a wall, with the spreading energy analogous to the splatting snow. The splat size is calculated efficiently by using the camera coordinates of the layered depth image and the new camera coordinates, and no holes appear in the output image.

Note that the fixed resolution of the LDI does not provide an appropriate sampling for all camera coordinates resulting in blurry artefacts. Chang et al. [36] overcome this problem by a hierarchical space partitioning method using an octree where in each cell a bounding box and an LDI containing the samples of this bounding box is stored. All LDI in the tree have the same resolution, but when traversing the so-called *LDI tree* from the root, the bounding boxes become smaller and hence the object is described more accurately. The LDI tree is rendered by traversing from the root and checking in each cell whether it provides enough detail when it is warped to the output image. If a current cell does not provide enough detail, i.e. the warped bounding box covers more than one output pixel, its children are traversed. Of course, the highest resolution is still limited by the resolution of the LDI in the leaf nodes of the octree.

Grossman and Dally's *point sample rendering* [69, 68] generates discrete surface points associated with normal and color information (*point samples*) by sampling orthographic views on an equilateral triangle lattice from any other surface representation. For rendering, the point samples are simply projected to screen space. By knowing in advance the target resolution and magnification at which the 3D object will be viewed, the side length of the triangles in the triangle lattice is chosen that no holes will appear in the final image. Grossman and Dally call this technique *adequate sampling*. However, magnifying an adequately sampled 3D object to a higher resolution results in a target image with *holes*, i.e. several pixels are not addressed although they should be. But as the object is adequately sampled at the original magnification, decreasing the object resolution by the same amount that the magnification is increased results again in an adequately sampled object [69]. Instead of using this brute force approach and rendering the point samples in the lower resolution resulting in blocky artefacts, Grossman and Dally use the *pull-push algorithm* primarily introduced by Gortler et al. [64]. In the *pull phase*, a succession of lower resolution approximations is calculated, and in the *push phase*, these lower resolution images are used to fill in the holes in the target resolution image.

Pfister et al. [124] generate discrete surface points associated with normals and material attributes called *surfels* by sampling an octree hierarchy of layered depth cubes [102], i.e. three orthogonal LDI, of any geometrically defined surface representation. For rendering, the octree is traversed starting from the root until the desired resolution is obtained, and the surfels are projected to image space. As the distance of adjacent rays during sampling is known in advance, the surfels are adequately sampled at a fixed resolution. Similar to Grossman and Dally [69, 68], when magnifying, holes are filled using the pull-push algorithm.

Botsch et al. [29] use a pure software implementation to render discrete surface points associated with normals and material attributes, that are generated by sampling a locally parameterized surface described by a parameterization function $\Pi \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$. The points are compactly stored in a hierarchical representation based on an octree with extremely low memory requirements, and this hierarchical representation is exploited to perform fast rendering by an incremental calculation of the projection of the surface points.

In contrast to all point rendering approaches using pure software implementations, Grisoni et al. [67, 152] make use of the simple OpenGL point primitive [116]. They sample surface points of a geometrically defined geometric object associated with normal and color information, also called surfels, on 3D grids of various resolutions. By knowing the resolution

of the grid, the object is adequately sampled in spirit of Grossman and Dally [69, 68]. For rendering, the appropriate grid is chosen regarding the viewing parameters, and each surfel is projected using the OpenGL point primitive. When magnifying, holes are filled using circular or rectangular splats, and the size can be determined by the viewing parameters.

Rusinkiewicz and Levoy [136] presented the QSplat algorithm that generates a hierarchy of spheres of different radii from a polygonal mesh in a preprocess that is stored efficiently in a tree. This is done by splitting the set of all vertices from the mesh along the longest axis of its bounding box. The two subtrees are computed recursively until a single vertex is reached. At the highest resolution in the leave nodes of the tree, small spheres are associated to the vertices of the mesh, and at intermediate resolutions in the interior nodes of the tree, bounding spheres are created that include the spheres of the corresponding subtrees. During rendering, the hierarchy traversal is adjusted in order to guarantee a given frame rate. The spheres are rendered by graphics hardware as splats, Rusinkiewicz and Levoy use circular, ellipsoidal, or rectangular splats that are either opaque, resulting in blocky artefacts, or have a gaussian transparency distribution, resulting in blobby artefacts. However, the QSplat algorithm adapts nicely to level-of-detail streaming over networks [137]. Recently, Dachsbacher et al. [46] showed, that the computational cost to traverse the multiresolution hierarchy can be shifted from the CPU to the graphics hardware by rearranging the nodes of the hierarchy into a sequential list.

Wand et al. use an output-sensitive rendering algorithm called the randomized z-buffer algorithm [160] that dynamically chooses random surface points from surfaces described by triangular meshes. The rendering time grows only logarithmically with the number of triangles, and Wand et al. state, that they can render surfaces consisting of up to 10^{14} triangles at interactive frame rates. A similar approach was developed independently by Stamminger and Drettakis [147], they also generate discrete points to be sufficiently dense in the final image using a hierarchical sampling method of triangular meshes, procedural objects, terrains, or volumetric objects. Their algorithm adapts particularly nice to simple modifications of the geometries.

Whereas all latter approaches render high-quality images by using a huge amount of statically or dynamically sampled surface points, Kalaiah and Varshney [85, 86] use less surface points by embedding local differential geometry for every point resulting in *differential points*differential point that are created in a preprocess. Starting from surfaces described by polygonal meshes or *NURBS* (parametric surfaces based on nonuniform Rational B-Splines), an initial super-sampled point-based representation is generated and then simplified to differential points, by generating less differential points in regions of low curvature and more differential points in regions of high curvature. The differential points are quantized into 256 different types and then rendered as normal-mapped rectangles. Thanks to the small number of required differential points to represent surfaces, the storage requirements and the traffic bandwidth from the CPU to the graphics hardware are significantly reduced.

Besides these pure point-based rendering techniques, various approaches have been proposed recently combining point-based and polygon-based rendering in one framework using some hybrid hierarchy [37, 41, 51, 40]. Points are rendered in regions where the polygonal mesh has a small screen-space projected area, but again the polygonal mesh has to be known a priori. Hybrid point-based and polygon-based rendering approaches have proven to

be particularly successful for the rendering of plant ecosystems [50].

No Knowledge about the Surface

All previous approaches render discrete surface points starting from 3D objects, where another surface representation is available. As our initial goal is to render an unorganized point set $\mathcal{P}$ coming from any source, we cannot rely on a given surface representation a priori, and we will now discuss some previous work about point rendering of an unorganized point set $\mathcal{P}$.

Zwicker et al.'s *surface splatting* [175] builds on work of Pfister et al. [124], but they can render an unorganized point set $\mathcal{P}$ associated with normals and color information by using a screen space formulation of the *Elliptical Weighted Average (EWA)* filter, that is still successfully used for traditional texture mapping of polygonal meshes [79, 78]. In a small neighborhood of a point $\boldsymbol{p}_i \in \mathcal{P}$, a local 2D parameterization of the surface is constructed by weighting the points in the small neighborhood projected on the tangent plane of $\boldsymbol{p}_i$ using truncated radially symmetric Gaussian functions. The variance matrix of the Gaussian function is chosen in order to match the local density of the points around $\boldsymbol{p}_i$. The mapping of the truncated local parameterizations to screen space (*EWA splats*) can be regarded as splatting ellipsoids, and the contributions of the mappings of all points of $\mathcal{P}$ convolved with a low-pass filter are accumulated in screen space. In order to avoid the accumulation with hidden surface parts, a modified A-buffer [32] is used.

Surface splatting provides high-quality anisotropic texture filtering with visually stunning results, but the pure software implementation is fill-limited and can only render a limited number of surface points in real-time. To overcome this problem, Ren et al. [130] formulated an object space EWA filter that renders EWA splats as textured rectangles on programmable graphics hardware [101] resulting in a significantly higher performance by transforming four vertices for an EWA splat using a textured rectangle. Botsch and Kobbelt [28] as well as Guennebaud and Paulin [71] obtain an even higher performance on programmable graphics hardware since they render EWA splats by transforming only one vertex through the rendering pipeline using the *point sprite* extension on modern graphics hardware.

The power of surface splatting has recently been extended to support depth-of-field point-based rendering [94]. However, since the surface is only approximated by local 2D parameterizations that are drawn as elliptical splats parallel to the viewport, artefacts can be noticed along the silhouette.

Efficiency

Almost all presented point-based rendering techniques have two components to further accelerate memory and rendering efficiency, and we will present some examples in the following.

First, for memory efficiency, the discrete surface points as well as its normals and material attributes are quantized in order to fit a huge amount of points in the memory. The point locations and material attributes are often quantized in an incremental method [136, 29] or by simply neglecting the least significant bits [67, 152]. The normals are often quantized by considering a volume divided into discrete patches and defining the quantized rays from the center of the volume to the patches. For example, a cube where the faces are subdivided into rectangular patches according to Zhang and Hoff [170] is used by Grossman and Dally [69, 68]

and Rusinkiewicz and Levoy [136], and a recursively subdivided octahedron is used by Grisoni et al. [67, 152] and Botsch et al. [29].

Second, for rendering efficiency, when a hierarchy is used for levels-of-detail, commonly known culling techniques are used to cull away fully invisible interior nodes in order to avoid the traversal of its subtrees. For *hierarchical view frustum culling* [39], the bounding sphere hierarchy is exploited for example by Rusinkiewicz and Levoy [136] as well as Alexa et al. [7, 6], and for *hierarchical backface culling* [95], they use normal cones [145] to represent the orientation of the interior nodes of the hierarchy. We [131] proposed to use a hierarchical bit encoding of the normal cones based on work of Zhang and Hoff [170].

All these point-based rendering techniques using forward warping account for local illumination, and in the following subsection we will present point-based ray-tracing techniques that also account for global illumination.

4.1.2 Point-based Ray-Tracing

Of course, ray-tracing of an unorganized point set $\mathcal{P}$ with an additional underlying surface representation can simply be done by ray-tracing the underlying surface. Various techniques exist for polygonal meshes, parametric surfaces [83] and especially NURBS surfaces [104], procedurally defined objects [84], and implicit surfaces as discussed in detail in Subsection 4.2.1, for a general survey see Hanrahan [73].

Schaufler and Jensen [140] presented a point-based ray-tracing technique where no other underlying surface representation is required. Schaufler and Jensen assume, that for an unorganized point set $\mathcal{P}$ the maximum size g of the largest gap between two points is known, and that the point set has associated normals $\boldsymbol{n}_i$ and eventually material attributes. Given a ray $R(t) = \boldsymbol{o} + t\boldsymbol{d}$ with origin $\boldsymbol{o}$ and direction $\boldsymbol{d}$, an intersection between the ray and the point set $\mathcal{P}$ is reported, when there is at least one point $\boldsymbol{p}_i$ in a cylinder C surrounding the ray and capped off by a plane perpendicular to the ray at the origin and the maximum length of the ray. Then, all C points inside the cylinder C and with a distance d_j from the ray R smaller than g are collected, and the intersection point $\boldsymbol{p}_{is}$ is calculated by interpolating the ray parameters t_j weighted by $(g - d_j)$ and inserted into R:

$$\boldsymbol{p}_{is} = R\left(\frac{\sum_{j=1}^{C} t_j(g - d_j)}{\sum_{j=1}^{C}(g - d_j)}\right) \tag{57}$$

The normal of the intersection point is simply interpolated and weighted by $g - d_i$ as well,

$$\boldsymbol{n}_{is} = \frac{\sum_{j=1}^{C} \boldsymbol{n}_j(g - d_j)}{\sum_{j=1}^{C}(g - d_j)}, \tag{58}$$

and any other associated material attribute is interpolated similarly. Note that the distance between a point and the ray is measured orthogonal to a point's normal and not orthogonal to the ray. For efficiency reasons, an octree is used to accelerate the decision whether a point $\boldsymbol{p}_i$ is inside the cylinder.

Unfortunately, the point-based ray-tracing technique is view-dependent, since the surface points to be taken into account during the interpolation of equations (57) and (58) depend on the view direction. Moreover, the technique suffers from noticeable artefacts along the

silhouette. Since the maximum size g of the largest gap between two points has to be known in advance, the point set $\mathcal{P}$ has to be quasi-uniformly distributed, and a high variation between the spacing of neighboring surface points can lead to strong artefacts.

Recently, Wand and Strasser [161] used the point-based ray-tracing technique to derive a multiresolution point-based ray-tracing technique. But in contrast to Schaufler and Jensen [140], an underlying surface defined by a triangular mesh is required. In a preprocess, a multiresolution hierarchy is set up from the triangular mesh, and it is stored in an octree. Discrete surface points are only used to approximate the geometry of the inner nodes, whereas the leaf nodes contain the original triangles of the mesh. In order to intersect a ray with the surface, the hierarchy is traversed, and when the points in the inner nodes are dense enough, the intersection is calculated according to Schaufler and Jensen [140], otherwise the ray is intersected with the original triangles of the mesh.

4.2 Visualization of Implicit Surfaces

4.2.1 Ray-Tracing

Ray-tracing enables to directly visualize implicit surfaces without any additional storage overhead while creating high-quality images. For every pixel in the output image, a single ray is cast from the viewpoint, and the intersection with the implicit surface is determined. By following secondary rays, ray-tracing also accounts for global illumination.

In this subsection, we will discuss various methods to intersect a ray $R(t) = \boldsymbol{o} + t\boldsymbol{d}$ with origin $\boldsymbol{o}$ and direction $\boldsymbol{d}$ with the implicit surface $\mathcal{S}$ defined by the function $f : \mathbb{R}^3 \to \mathbb{R}$. Whereas analytical solutions exist for low-degree *algebraic implicit surfaces*, where the defining function f can be written as a polynomial equation, we will concentrate here on the more general case for the defining function f.

Interval analysis was first used by Mitchell [111] to intersect a ray with an implicit surface. First, the zero-values of the defining function are isolated by finding intervals $t \in [t_1, t_2]$ that are known to contain one and only one zero-value of the defining function $f(R(t) = \boldsymbol{o} + t\boldsymbol{d}) = 0$. Secondly, a refinement technique based on Newton's method and regula falsi is applied to those intervals until the ray surface intersection is located as accurately as needed.

Sphere Tracing is a ray-tracing technique for implicit surfaces that marches stepwise along the ray R starting from $\boldsymbol{o}$ towards the first intersection, primarily introduced by Hart [75]. The steps are calculated guaranteed not to penetrate the implicit surface, and the only requirement is that the defining function f is *Lipschitz*, i.e. there exists a *Lipschitz bound* L such that for all arguments $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ the following inequality holds:

$$|f(\boldsymbol{x}_1) - f(\boldsymbol{x}_2)| < L|\boldsymbol{x}_1 - \boldsymbol{x}_2| \tag{59}$$

The Lipschitz bound L bounds above the rate of change of the defining function f between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, and this means that $\frac{f(\boldsymbol{x})}{L}$ is an upper bound for the distance between $\boldsymbol{x}$ and the implicit surface $\mathcal{S}$.

Sphere tracing starts with $t = 0$ and marches along the ray $R(t) = \boldsymbol{o}+t\boldsymbol{d}$ in irregular steps $\frac{f(R(t))}{L}$ guaranteed not to intersect the implicit surface. Nevertheless, an intersection is reported when the marching along the ray approaches the zero-value of the defining function within an error tolerance ϵ, i.e. $|f(R(t))| < \epsilon$. No intersection is reported, when t takes a value greater than a maximum ray traversal distance.

LG-Surfaces introduced by Kalra and Barr [87] are another class of implicit surfaces that can be ray-traced reliably. Besides the existence of a Lipschitz bound L of the defining function, LG-Surfaces require the existence of a directional bound G of its gradient along the ray R:

$$|\frac{\partial f(R(t_1))}{\partial t} - \frac{\partial f(R(t_2))}{\partial t_2}| < G|t_1 - t_2| \tag{60}$$

Kalra and Barr's ray-tracing algorithm relies on the determination of an interval $[t_1, t_2]$ nearest to the ray's origin $\boldsymbol{o}$ with exactly one intersection of the ray and the implicit surface, that can then be refined using Newton iterations or regula falsi. Starting from the intersections of a bounding box of the implicit surface $\mathcal{S}$ at ray parameters t_1 and t_2, the interval $[t_1, t_2]$ is subdivided recursively, and an intersection is reported if an interval with exactly one intersection is found.

In order to find the number of intersections of an interval $[t_1, t_2]$ with $t_m = (t_1 + t_2)/2$ and $d = (t_2 - t_1)/2$, the directional bound G is used: if $f(R(t_1))$ and $f(R(t_2))$ are of opposite signs, there is at least one intersection in the interval, and there is exactly one intersection, when $\|f(R(t_m))\| > Gd$, i.e. when the defining function is monotone in the interval. Otherwise, if $f(R(t_1))$ and $f(R(t_2))$ have the same sign and $\|f(R(t_m))\| > Gd$, it is guaranteed that there is no intersection in the interval.

Note, that Kalra and Barr use a preprocess to accelerate the ray-tracing, where regions in space that do not contain any intersection with the implicit surface are pruned away using the Lipschitz bound L of the defining function.

4.2.2 Polygonization

In this subsection, we present different methods to represent a continuous implicit surface $\mathcal{S}$ defined by an defining function $f : \mathbb{R}^3 \to \mathbb{R}$ by discrete polygonal meshes, and according to Bloomenthal [25], we call this conversion *polygonization*. Polygonal meshes have received much attention in computer graphics since conventional graphics hardware is optimized for displaying polygons, and today's graphics hardware is capable of rendering 350 million triangles per second [117] by using a depth buffer [35] and shading model according to Gouraud [65] or Phong [125]. A major advantage compared to ray-tracing of implicit surfaces is, that the polygonization has to be done only once for a static surface, regardless of the viewing parameters. On the other hand, the polygonization may require a considerable amount of time and induces storage overhead. In contrast to ray-tracing implicit surfaces, rendering the polygonal mesh by graphics hardware does not account for global illumination.

Of course, the polygonal mesh obtained by the polygonization should accurately describe the implicit surface and should be *topologically correct*, i.e. it should be homeomorphic to

the implicit surface. Furthermore, the polygonal mesh should be *topologically consistent*, i.e. without disconnected vertices, edges, or faces, and distorted faces should be avoided.

According to Akkouche and Galin [5], we divide the polygonization techniques into three categories.

Spatial sampling techniques subdivide the 3D space into cells, commonly either cubes or tetrahedra, and search for the cells that intersect the implicit surface.

One of the most commonly known spatial sampling techniques is the marching cubes algorithm [169, 103], that divides the 3D space into cubic cells. At each vertex of each cube, the defining function f is evaluated and depending on the signs, triangles are created for each of the cubes. When there are at least two vertices $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ of the same cube with $f(\boldsymbol{x}_1)f(\boldsymbol{x}_2) \leq 0$, the surface $\mathcal{S}$ must cross the edge between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, and the intersection $\boldsymbol{x}'$ with $f(\boldsymbol{x}') = 0$ of the surface and the edge is computed by linear interpolation. There are $2^8 = 256$ different configurations depending on the signs of the 8 vertices of the cube, and by exploiting symmetry, this reduces to 15 essential configurations. But as pointed out by Dürst [54], 5 of these essential configurations are ambiguous, and different triangles can be created for the cube leading to topological inconsistencies for the wrong choice. Several disambiguation strategies have been addressed, see Van Gelder and Wilhelms [58].

Marching tetrahedra algorithms, e.g. by Shirley and Tuchman [144] or Hall and Warren [72], further divide the cubic cells into tetrahedra, and for each tetrahedra, there are only 8 essential configurations without ambiguous cases resulting in topologically consistent polygonal meshes [8]. Nevertheless, marching tetrahedra algorithms create numerous, often over distorted triangles.

Since the cells are of constant size, all these techniques may miss small features and do not adapt to the local geometry of the implicit surface, and an adequate cell size has to be determined. When the size is chosen too small, an excessive number of polygons may be produced, and when it is too large, detail may be obscured. To overcome this drawback, adaptive subdivision techniques converge to the surface recursively. Bloomenthal [23] uses a conventional octree decomposition, but *cracks* may occur between triangles of adjacent cells of different size.

Surface fitting techniques create a seed mesh that roughly approximates the implicit surface and progressively adapt and deform it to better fit the implicit surface. For example, Velho [158, 159] starts with a coarse polygonal approximation of the surface according to [63], and subdivides each polygon recursively according to the local curvature. Care must be taken, that the coarse polygonal approximation captures the correct topology of the implicit surface, otherwise the resulting polygonal mesh may be topologically incorrect.

Other surface fitting techniques either assume special classes of implicit surfaces created from skeletal elements [49, 43], or rely on a search for critical points [30, 148, 167] suffering from inefficiency for complex implicit surfaces.

Surface tracking techniques also known as *continuation techniques*, start from a seed element on the surface and iteratively grow a polygonal mesh that approximates the implicit

surface. *Cellular surface tracking techniques* [8, 169, 24] start from a cell that intersects the implicit surface and iteratively find all intersecting cells among its neighbors. Since the cells are of constant size, cellular surface tracking techniques suffer from the same drawback as non-adaptive spatial sampling techniques, and furthermore, in the general case it can be difficult to determine a seed cell.

Alternatively, a particle system can be used to evenly distribute samples over the implicit surface [154, 149, 166] and a *Delaunay triangulation* [48] is computed for polygonization [47]. Again, care must be taken that the particles are dense enough to create a topologically correct polygonal mesh.

Hilton [80] defines a surface tracking technique that directly creates triangles during polygonization called *marching triangles*. Starting from a seed triangle, new triangles are created iteratively from the boundary edges by using a relaxed Delaunay constraint that locates new vertices that are linked to the mesh. Hilton's initial marching triangles algorithm was further improved by others [5, 77, 88].

After the polygonization of the implicit surfaces, the resulting polygonal mesh can further be optimized in a postprocess, e.g. by efficiently creating triangle strips [132]. In order to represent sharp features more precisely, see for example Ohtake and Belyaev [120] as well as Kobbelt et al. [89].

4.3 Rendering and Ray-Tracing of Implicit Surfaces reconstructed from Points

4.3.1 Point-based Rendering

Alexa et al. [7, 6] reconstruct an implicit surface starting from the unorganized point set $\mathcal{P}$ called *point set surface*. The surface reconstruction is based on the moving least squares approximation that we already explained in Subsection 1.3 in Part I of this thesis. The unorganized point set is organized in a bounding sphere hierarchy similar to QSplat [136], but since the radius of the spheres in the leaf nodes cannot be estimated from an underlying polygonal representation, it is simply set to the feature size h, i.e. the anticipated spacing between neighboring points. For rendering, the hierarchy is traversed recursively starting from the root, and the size of the bounding spheres in pixel space is calculated similar to QSplat. If the bounding sphere projects to less than a pixel, its center is simply projected to screen space without regarding the subtree. Otherwise, when the traversal reaches a leaf node with a bounding sphere that projects to more than one pixel, additional points are generated on a sufficiently dense grid parallel to the reference plane H of the leaf node's representation point in order to cover the extent of the projected leaf node's bounding sphere. Alexa et al. sample the polynomial associated with the representation point on the grid rather than projecting the grid points on the surface using the projection Ψ introduced in Section 1.3, since the cost would be too high for interactive applications. Consequently, a non-continuous surface is rendered, but Alexa et al. show that the Hausdorff error of this approximation can be bounded and is not worse than the error of the surface computed using the projection Ψ.

4.3.2 Polygonization

Crespin uses a polygonization technique that is exclusively designed for variational implicit surfaces [44]. The technique takes benefit of the fact, that the unorganized point set $\mathcal{P}$ provides direct information about the geometry and topology of the surface.

In a first step, Crespin creates an initial Delaunay tetrahedralization from $\mathcal{P}$ and the vertices of its bounding box stretched by a factor of two, assuming that the variational implicit surface entirely lies in this stretched bounding box. The Delaunay tetrahedralization is calculated incrementally according to Boissonnat et al. [27]. In a second step, Crespin collects the tetrahedra that cross the implicit surface, i.e. the tetrahedra where the defining function f has both positive and negative values at the vertices. Then, in an iterative refinement procedure, the tetrahedra are refined until they all have a circumsphere smaller than a predefined threshold. The refinement is done by inserting new vertices at the centers of the tetrahedra into the Delaunay tetrahedralization, that is updated incrementally [27], and all tetrahedra that cross the implicit surface are kept. Finally, the tetrahedra are polygonized according to Bloomenthal [24].

Crespin's algorithm can be understood as a surface fitting technique, since a seed mesh that roughly approximates the surface is created using the initial Delaunay tetrahedralization, and the seed mesh is then adapted to better fit the surface. Unfortunately, Crespin's algorithm was designed to polygonize variational implicit surfaces reconstructed from small unorganized point sets, and no results for implicit surfaces reconstructed from large unorganized point sets are given.

4.3.3 Ray-Tracing of Point Set Surfaces

Adamson and Alexa [2] describe a method to ray-trace surfaces represented by point set surfaces that are implicitly defined by the projection $\Psi : \mathbb{R}^3 \to \mathbb{R}^3$ as the points that project onto themselves as explained in Section 1.3 of Part I.

In a preprocess, a sphere structure enclosing the surface is computed as the union of spheres around all $\boldsymbol{p}_i \in \mathcal{P}$ with a radius slightly smaller than the feature size h. Starting from these spheres, a bounding sphere hierarchy is built bottom-up.

For a given ray $R(t) = \boldsymbol{o} + t\boldsymbol{d}$ with origin $\boldsymbol{o}$ and direction $\boldsymbol{d}$, the ray-surface intersection is calculated as follows. First, all spheres intersected by the ray are collected using the bounding sphere hierarchy, and the spheres are sorted with respect to their distance from the ray origin $\boldsymbol{o}$ and stored in a priority queue. Then, the spheres from the priority queue are inspected one at a time, until an intersection with the surface is reported.

In order to calculate an intersection between the ray and the surface in a given sphere, the sphere's center is projected onto the surface by the projection Ψ providing a local polynomial approximation. When there is no intersection between the ray and this local polynomial approximation, no intersection inside the sphere is reported. Otherwise, let $\boldsymbol{p}_{is}$ be the closest intersection point to the ray origin $\boldsymbol{o}$. If the projected distance between $\boldsymbol{p}_{is}$ and the surface is below an error tolerance ϵ , i.e. $\|\Psi(\boldsymbol{p}_{is}) - \boldsymbol{p}_{is}\| < \epsilon$, the intersection is found. Otherwise, in an iterative refinement procedure, the projection $\Psi(\boldsymbol{p}_{is})$ and the intersection between the resulting new local polynomial approximation and the ray yielding a new intersection point is repeated, until the projected distance to the surface falls below the error tolerance ϵ. Adamson and

Alexa experienced, that 2 to 3 projections are sufficient to accurately intersect a ray with the surface [2]. Note, that Adamson et al. propose to use degree $d = 2$ polynomial approximations in order to avoid iterative intersection computations [2]. For efficiency reasons, the algorithm exploits coherence by storing the local polynomial approximation resulting from the projection of the sphere's center for further rays.

Adamson and Alexa are currently working on a new surface approximation technique for unorganized point sets that is inspired by the moving least squares approximation, but the ray intersection computations can be calculated more efficiently [1].

4.4 Other Work

In this chapter, we presented an extensive study of previous work on rendering techniques for unorganized point sets, rendering techniques for implicit surfaces, and rendering techniques that use both these surface representations. Nevertheless, there are several other rendering techniques that we did not discuss in detail. For example, Witkin and Heckbert [166] introduced a totally different rendering technique for implicit surfaces that was further improved by Hart et al. [76]. The implicit surface is sampled by a physically based particle system and the particles are rendered as discs perpendicular to the particle's normal. As an other rendering technique for implicit surfaces, Mora et al. [112] use an efficient implementation of parametric cubes.

We have seen in Section 4.3, that the only rendering techniques using both the unorganized point set and the reconstructed implicit surface were presented for point set surfaces [7, 6, 2] and for variational implicit surfaces reconstructed from small unorganized point sets [44]. This fact motivated us to define two new rendering techniques that use both these surface representations, and we will present them in the next two chapters.

Chapter 5

Point-based Rendering of Implicit Surfaces from Unorganized Points

5.1 Overview

In this chapter, we present a new point-based rendering technique for an unorganized point set $\mathcal{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_N\}$ with associated normals and material attributes using the reconstruction of an implicit surface $\mathcal{S}$ [133]. The implicit surface is rendered view-dependently in an output-sensitive multiresolution manner using points as rendering primitive without the creation of a polygonal mesh representation.

The technique can be divided into a preprocessing phase and a rendering phase. During preprocessing, an implicit surface is reconstructed from the unorganized point set $\mathcal{P}$, and a bounding sphere hierarchy is set up (Section 5.2). During rendering (Section 5.3), the bounding sphere hierarchy is traversed starting from the root, invisible nodes are culled away, and the spheres are either projected directly to the screen when their projected screen-space size is smaller than a threshold, or additional surface points are generated by sampling the reconstructed implicit surface $\mathcal{S}$ through local ray-casting within the sphere. We show some examples to illustrate the point-based rendering technique and present qualitative and quantitative results in Section 5.4 before we conclude in Section 5.5.

5.2 Preprocessing Phase

5.2.1 Implicit Surface Reconstruction

In a first step of the preprocessing phase, an implicit surface $\mathcal{S}$ is reconstructed from a given unorganized point set $\mathcal{P}$. Any surface reconstruction method of implicit surfaces of Part I can be used, as long as a defining function f is explicitly given. Furthermore, the implicit surface should interpolate the point set $\mathcal{P}$ or pass nearby the points $\boldsymbol{p}_i$. In order to enable the reconstruction from a large point set $\mathcal{P}$, we initially proposed [133] to use variational implicit surfaces defined by radial basis functions with either compact support (Subsection 1.4.4) or global support with an associated fast evaluation technique (Subsection 1.4.3). However, the rendering algorithm is even more efficient by exploiting recent advances in reconstructing implicit surfaces from large unorganized point sets $\mathcal{P}$, such as the partition of unity variational

method (Chapter 2), the hierarchical partition of unity variational method (Chapter 3), and multi-level partition of unity implicits (Section 1.5), due to the locality of these techniques and hence the faster evaluation of the defining function f.

5.2.2 Construction of the Bounding Sphere Hierarchy

In the second step of the preprocessing phase, a bounding sphere hierarchy is constructed from the unorganized point set $\mathcal{P}$ in spirit of the QSplat algorithm [136]. In general, bounding sphere hierarchies are considered as time and space efficient [135], and are typically stored as trees, whose inner nodes consist of spheres that bound the volume of all children nodes, and whose leaf nodes consist of simple primitives. This results in a level-of-detail representation allowing view-dependent refinement and the use of several speed-up techniques, especially visibility culling.

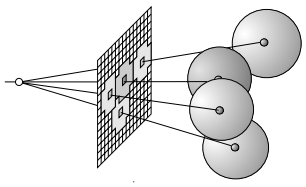


Figure 31: The projection of the bounding spheres to the screen forms a closed region.

We create the bounding sphere hierarchy in a top down recursive process and store it in a *binary space partitioning (BSP)* tree. The root node consists of the bounding sphere of the point set $\mathcal{P}$. At each step of the recursion, the axis aligned bounding box of the node's points is computed, and two children nodes are created by splitting the points along the longest axis of the bounding box into two disjunct sets with an equal number of points (or one set with only one point more than the other). In each node except the leaf nodes, a bounding sphere of the points is stored. The recursion terminates at nodes with only one point of the point set $\mathcal{P}$. The bounding spheres associated to the leaf nodes are calculated, so that they are slightly overlapping and enclose the entire reconstructed implicit surface in order to form a closed region when projected to the screen (Figure 31). In every node in addition to the bounding sphere, we store a material attribute and a normal by averaging the respective values of the children nodes. Note that this can smooth out sharp edges and high-frequency materials, nevertheless, we found averaging the best approximation in the inner nodes. Furthermore, in every inner node, we store a normal cone [145] for hierarchical backface culling:

```
struct Node {
   Node* left;
   Node* right;
   Sphere boundingSphere;
   Material someMaterialAttribute;
   Normal normal;
   NormalCone normalCone;
}
```

The algorithm that generates the bounding sphere hierarchy is outlined in Algorithm 4.

Algorithm 4 generateHierarchy(`points[1..n]`)

Require: `n` points `points[1..n]` with associated normals `points[1..n].normal` and material attributes `points[1..n].material`

Ensure: returns a node of the bounding sphere hierarchy

```
if n>1 then
  determine the bounding box bbox of the points[1..n]
  determine the longest axis of bbox
  determine points[median] and rearrange points
  node.left = generateHierarchy(points[1..med])
  node.right = generateHierarchy(points[med+1..n])
  node.boundingSphere = calculateBoundingSphere(bbox)
  node.material = (node.left.material+node.right.material)/2
  node.normal = (node.left.normal+node.right.normal)/2
  node.normalCone = calculateNormalCone(node.left.normalCone,
  node.right.normalCone)
  return node
else
  node.left = 0
  node.right = 0
  node.boundingSphere = calculateBoundingSphere(node)
  node.material = points[1].material
  node.normal = points[1].normal
  node.normalCone = calculateNormalCone(points[1].normal)
  return node
end if
```

In order to illustrate the bounding sphere hierarchy construction on a simple example in 2D, see Figure 32 for the BSP created from 10 unorganized points, as well as its corresponding BSP tree in Figure 33. The bounding spheres of three different levels of the hierarchy created from 437,645 points of the Stanford Dragon model can be seen in Figure 34.

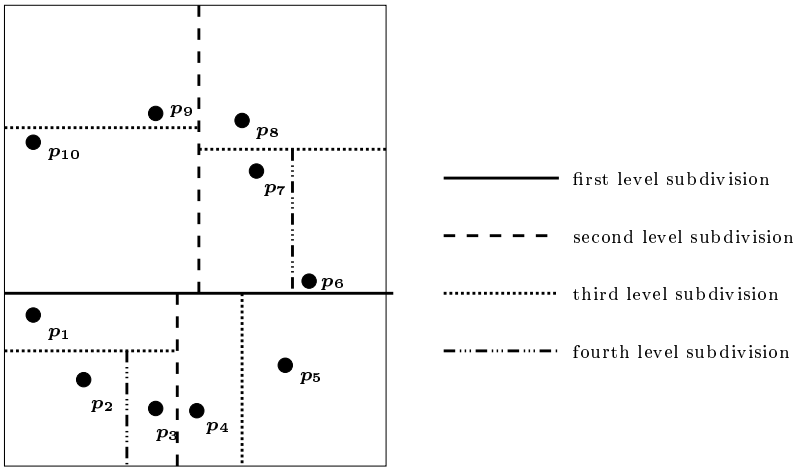


Figure 32: A BSP from 10 unorganized points.

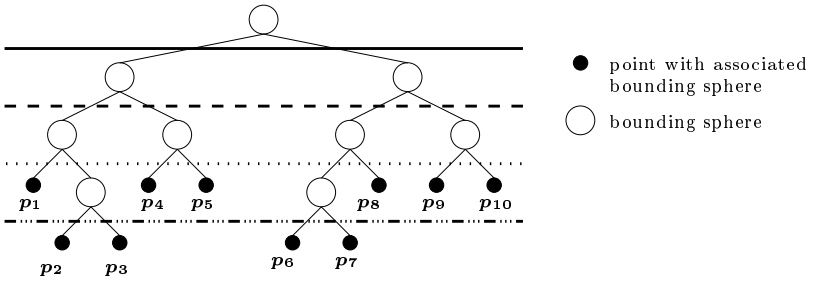
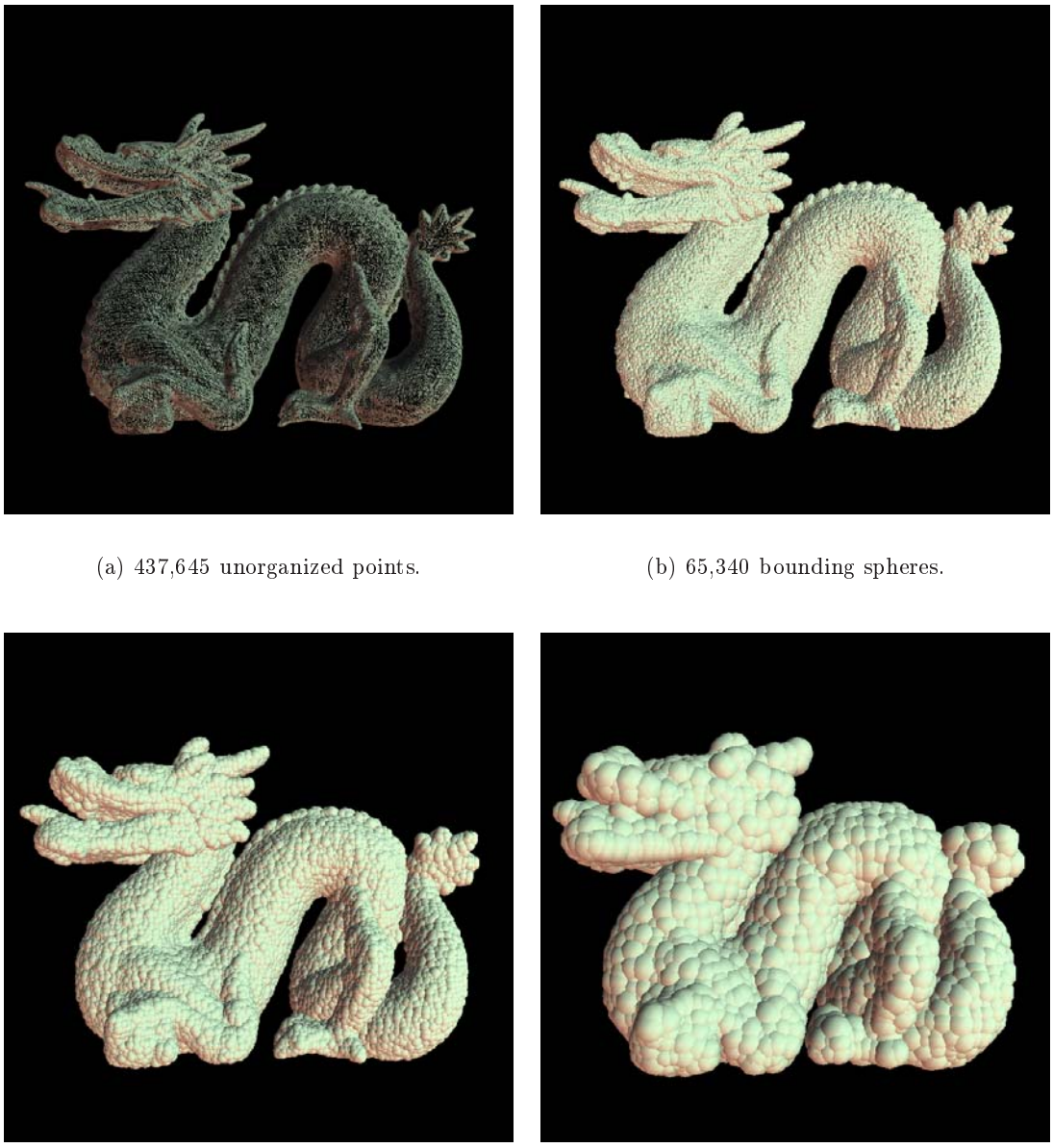


Figure 33: The corresponding BSP tree.



(a) 437,645 unorganized points.

(b) 65,340 bounding spheres.

(c) 15,284 bounding spheres.

(d) 3,803 bounding spheres.

Figure 34: Some levels of the Stanford Dragon's bounding sphere hierarchy.

5.3 Rendering Phase

5.3.1 Traversing the Hierarchy

Once the bounding sphere hierarchy is set up during the preprocessing phase, it is used for the view-dependent multiresolution point-based rendering algorithm that is described in the following.

For every rendering frame, the hierarchy is traversed starting from the root node. When a node is entirely invisible, the corresponding branch is skipped from processing as explained in Subsection 5.3.2. The hierarchy is traversed until the screen space projected size of a node's bounding sphere in pixels falls below a threshold τ, and a splat is drawn as explained in Subsection 5.3.3. The threshold τ determines the level-of-detail for the multiresolution rendering, i.e. the maximum splat size for each projected bounding sphere on the screen. Too high values for τ can lead to blocky or blurry artefacts, ideally τ should be set to one pixel so that each projected bounding sphere falls onto one pixel on the screen.

When reaching a leaf node of the hierarchy with a screen-space projected size above the threshold τ, additional points on the surface have to be generated as can be seen in Subsection 5.3.4, in order to avoid splats with a size greater than the threshold τ and hence blocky or blurry artefacts. An overview of the rendering algorithm is outlined in Algorithm 5.

Algorithm 5 traverseHierarchy(`node`)

Require: node `node` of the bounding sphere hierarchy
Ensure: render the node
  if `node` invisible **then**
    skip this branch of the tree from processing
  else if `node` is a leaf node **then**
    if projectedSceenSize(`node.boundingSphere`) $< \tau$ **then**
      draw a splat
    else
      generate additional points
      draw splats of the additional points
    end if
  else if projectedScreenSize(`node.boundingSphere`) $< \tau$ **then**
    draw a splat
  else
    traverseHierarchy(`node.left`)
    traverseHierarchy(`node.right`)
  end if

5.3.2 Culling

We use a *conservative* culling technique to cull away nodes that are entirely invisible during the hierarchy traversal. Recall, that every node of the bounding sphere hierarchy stores a bounding sphere and a normal cone, that are used for hierarchical view frustum culling and hierarchical backface culling, respectively.

Hierarchical view frustum culling is done by using the bounding sphere of a node to cull away nodes, that are entirely not in the view frustum. The center of the bounding sphere is checked against all six planes that define the view frustum. When the center of the bounding sphere is outside of at least one of the view frustum planes with a distance greater than the bounding sphere's radius, it is completely out of the view frustum, and the branch of the node is skipped from further processing.

In order to further optimize the hierarchical view frustum culling, we adopted two speed-up techniques from Assarsson and Möller [11].

First, we use *masking* during the hierarchy traversal to mask off all view frustum planes where the bounding sphere of a node is completely inside, in order to exclude these view frustum planes from the view frustum culling test for all children nodes.

Second, we exploit temporal coherence between consecutive rendering frames using the *plane-coherence test*: when a node of the bounding sphere is outside a view frustum plane in a rendering frame, it is probable that it is outside the same view frustum plane in the next frame. Hence, for every view frustum culled node, we store the corresponding plane against the node was culled away, and this will be the first plane to test the node's bounding sphere in the next frame.

Hierarchical backface culling is done by using the normal cones stored in each node to cull away nodes, that are entirely backfacing. When the normal cone of a node is entirely backfacing, the node is completely invisible, and the branch of the node is skipped from the hierarchy traversal. Furthermore, in spirit of the masking performed during hierarchical view frustum culling, we mask off all children nodes with entirely frontfacing normal cones, in order to exclude them from the backface culling test.

5.3.3 Drawing Splats

In our new rendering algorithm, both the projected bounding spheres and the additionally generated points are drawn as splats. After calculating the screen-space projected size of the bounding sphere or the additionally generated point, respectively, a splat is drawn. Rendering a square or circular splat parallel to the viewport can be done very efficiently using the graphics hardware, but the higher the splat size, the blockier is the resulting image. This is why we also support high-quality EWA splats introduced by Zwicker et al. [175], where the splat is not parallel to the viewport but perpendicular to a point's associated normal, and the splats are accumulated in screen space using an elliptical weighted average filter. But again, larger splats lead to visible artefacts, and for EWA splats, the higher the splat size, the blurrier is the resulting image.

5.3.4 Generating Additional Points

In order to avoid to render splats of a size greater than the threshold τ when reaching a leaf node of the bounding sphere hierarchy, additional surface points are generated. To this end, the reconstructed implicit surface $\mathcal{S}$ defined by the defining function f is sampled within the leaf's bounding sphere through local *ray-casting*. We prefer to use the term ray-casting

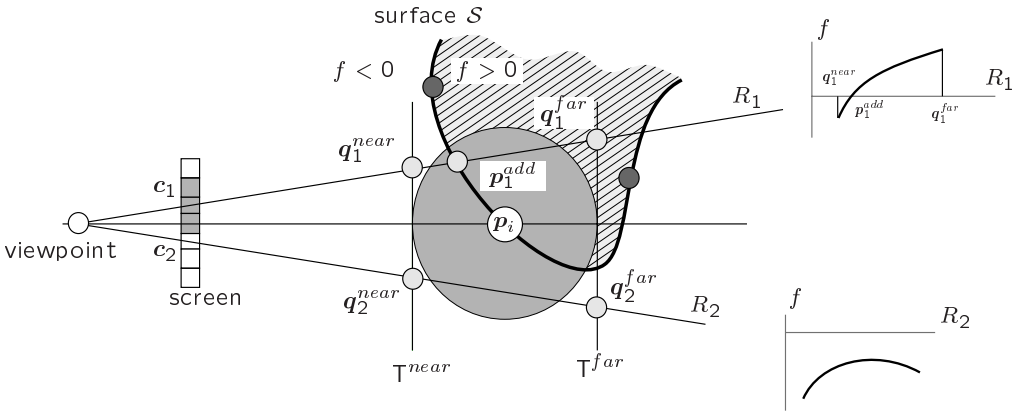


Figure 35: The projection of a leaf's bounding sphere to the screen (in 2D).

instead of the term ray-tracing, since we cannot account for secondary rays in our rendering technique.

Let us suppose that $\tau = 1$. In a first step, the screen space projected area of the leaf's bounding sphere is determined using the viewing parameters. This area consists of all screen pixels $\boldsymbol{c}_j$. Then, in a second step, for each pixel an additional surface point $\boldsymbol{p}_j^{add} \in \mathcal{S}$ that projects to the screen pixel $\boldsymbol{c}_j$ is determined. This is done by casting a ray R from the viewpoint through the pixel $\boldsymbol{c}_j$ and intersecting it with the reconstructed implicit surface $\mathcal{S}$ inside the leaf's bounding sphere. Since we consider implicit surfaces with no further assumption on the defining function f, and we suppose that there is either exactly one intersection or no intersection with the implicit surface inside the leaf's bounding sphere (we will be more precise on this fact below), we determine the additional surface point $\boldsymbol{p}_j^{add}$ by a local ray-casting through interval analysis inside the leaf's bounding sphere for efficiency reasons.

Consider the projection of the leaf's bounding sphere to the output image in Figure 35 to the pixels $\boldsymbol{c}_j$. The plane T^{near} (respectively T^{far}) is the nearest (respectively farthest) tangent plane of the leaf's bounding sphere that is parallel to the viewport. For each pixel $\boldsymbol{c}_j$, the ray R from the viewpoint to $\boldsymbol{c}_j$ intersects the planes T^{near} (respectively T^{far}) in the points $\boldsymbol{q}_j^{near}$ (respectively $\boldsymbol{q}_j^{far}$).

Then, the values of the defining function f of the implicit surface $\mathcal{S}$ in $\boldsymbol{q}_j^{near}$ and $\boldsymbol{q}_j^{far}$ are calculated. By supposing that the leaf's bounding sphere is sufficiently small so that there is at most one intersection with the surface between $\boldsymbol{q}_j^{near}$ and $\boldsymbol{q}_j^{far}$, we only have to consider the two following cases.

- If the values of the defining function $f(\boldsymbol{q}_j^{near})$ and $f(\boldsymbol{q}_j^{far})$ are of different sign (see $\boldsymbol{q}_1^{near}$ and $\boldsymbol{q}_1^{far}$ in Figure 35), we calculate the surface intersection point $\boldsymbol{p}_j^{add}$ by interpolating between $\boldsymbol{q}_j^{near}$ and $\boldsymbol{q}_j^{far}$ using the regula falsi algorithm. Tests have shown, that two iterations are enough to get a sufficient approximation. The resulting surface intersection point $\boldsymbol{p}_j^{add}$ will be projected to the pixel $\boldsymbol{c}_j$, and its normal is given by the gradient of the defining function f.
- If $f(\boldsymbol{q}_j^{near})$ and $f(\boldsymbol{q}_j^{far})$ are of the same sign (see $\boldsymbol{q}_2^{near}$ and $\boldsymbol{q}_2^{far}$ in Figure 35), we consider that the ray is not intersecting the surface.

For $\tau > 1$, we sample only a regularly sampled subset of the pixels $\boldsymbol{c}_j$ and generate a subset of the additional points $\boldsymbol{p}_j^{add}$, so that all pixels are covered when splatting the subset of the additionally generated points.

Let us now discuss some details about the rendering algorithm.

First, why do we expect that there will be only exactly one intersection or no intersection at all inside the leaf's bounding sphere? We admit that we cannot prove this to be true. Especially along the silhouette of the implicit surface, there might either be more than one intersection with the implicit surface, or the interval analysis might report no intersection although there is one. But recall that we reconstruct the implicit surface $\mathcal{S}$ from the point set $\mathcal{P}$, with all points $\boldsymbol{p}_i$ being the centers of the leafs' bounding spheres, and that the bounding spheres are just small enough to be slightly overlapping. Since we presume to reconstruct a smooth implicit surface from the point set $\mathcal{P}$, we exclude high frequency surfaces between neighboring points in the point set $\mathcal{P}$ at the outset, especially when reconstructing variational implicit surfaces, multi-level partition of unity implicits, or implicit surfaces by using the partition of unity variational or hierarchical partition of unity variational methods. Therefore, we trade speed for quality in the rendering algorithm by using interval analysis. Of course, we can also use more sophisticated and expensive algorithms as discussed in Subsection 4.2.1 to find the ray surface intersection inside the sphere. But when considering the difference between using interval analysis in the leaf's bounding spheres (Figure 36(a)) against using Hart's sphere tracing in the leaves' bounding spheres (Figure 36(b)) on a rendering example of a reconstructed Stanford Bunny using the partition of unity variational method where all pixels result from additionally generated points, we did not find any significant visual difference as can be noticed in the difference image in Figure 36(d). The extent of the projected leaves' bounding spheres is illustrated in Figure 36(e).

Second, in order to determine $f(\boldsymbol{q}_j^{near})$ and $f(\boldsymbol{q}_j^{far})$, we do not intersect the ray R with the sphere but with the tangent planes T^{near} and T^{far}. Again, we trade speed for quality, since intersecting the ray $R(t)$ with the tangent plane does not require the calculation of a square root compared to intersecting the ray $R(t)$ with the sphere. In order to defend the choices made by our algorithm, consider the difference on the rendering example between using Hart's sphere tracing in the tangent planes of the leaf's bounding spheres (Figure 36(b)) against the straightforward ray-tracing of the Stanford bunny (Figure 36(c)) shown in Figure 36(f).

(a) Interval analysis in the leaves.

(b) Sphere tracing in the leaves.

(c) Straightforward sphere tracing.

(d) Difference image between (a) and (b).

(e) Illustration of the extent of the leaves with a different color for each leaf.

(f) Difference image between (b) and (c).

Figure 36: Quality comparison when generating additional points.

5.4 Results

In this section, we present and discuss some obtained results using our new point-based rendering technique. All results were obtained on an Intel Pentium 1.7 GHz with 512 MB of RAM and a GeForce 3 graphics board running Linux.

First, we present the results of the preprocessing phase. Of course, the required time for the reconstruction of the implicit surface depends on the chosen reconstruction technique. In this section, we exclusively use the partition of unity variational method presented in Chapter 2 of Part I, and we refer to the obtained results therein. For the construction of the bounding sphere hierarchy, see Table 14 for the required time $t_{hierarchy}$ and number of levels L, that as a matter of course depends on the number of points N of a given 3D object.

Model	N	L	$t_{hierarchy}$
Buddha	543,652	19	5.75
Dragon	437,645	18	4.04
Male	148,138	17	1.48
Bunny	34,834	15	0.29

Table 14: Required time in seconds for the construction of the bounding sphere hierarchy and the number of levels L.

Second, the results of the rendering phase are presented. Since we implemented the bounding sphere hierarchy in a queue data structure, no pointers have to be followed and hence traversing the hierarchy is extremely fast. The hierarchical view frustum culling and the hierarchical backface culling can be processed very fast as well, and we do not give any timings here since the bottleneck in the rendering algorithm is the rendering of the splats and the generation of the additional points.

Consider Figures 37 for an illustration of the bounding sphere hierarchy traversal to render the Stanford Dragon. When the hierarchy is not traversed until the leaf node because the screen space projected size of an interior node's bounding sphere falls below the threshold $\tau = 2$, a red-colored splat is drawn. Otherwise, when reaching a leaf node, a green-colored splat is drawn for nodes with a screen-space projected size below the threshold τ, and a blue-colored splat is drawn when additional points have to be generated. See Figures 37(a) (10,352 splats), Figure 37(c) (53,211 splats), and Figure 37(e) (248,645 splats) to illustrate what type of node was used for three different viewing parameters, and consider the corresponding rendered images using hardware-accelerated splats without the generation of additional points in Figure 37(b) ($>$ 400 *frames per second (fps)*), Figure 37(d) ($>$ 90 fps), and Figure 37(f) ($>$ 20 fps).

The rendering quality is much higher when 14,517 additional points are generated at the mouth of the Stanford Dragon using a reconstructed implicit surface as can be seen in Figure 37(g). Note especially the set of teeth and the tongue in a close-up of Figure 37(g) shown in Figure 37(i) compared to the close-up of Figure 37(f) shown in Figure 37(h), where no additional points were generated.

Another example of a similar image sequence can be seen in Figure 38, where especially the mouth and the nose of the Cyberware Igea are rendered much smoother by generating

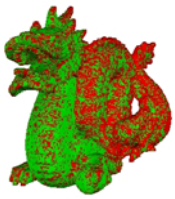
17,212 additional points using a reconstructed implicit surface as can be seen in Figure 38(g) and its close-up in Figure 38(i) compared to the close up of Figure 38(f) shown in Figure 38(h), where no additional points were generated.

Let us now present some timings for the rendering. When no additional points have to be generated, by using our GeForce 3 graphics board, we can render up to 5M hardware-accelerated splats. Our output-sensitive software implementation of the high-quality EWA splatting technique can render up to 300k high quality EWA splats per second in a 512x512 window. Note that the authors of the EWA splatting technique claim to render up to 500k EWA splats per second using a better optimized implementation on a similar hardware configuration [175]. In order to show the visual difference with respect to the type of splats, consider the $38,221$ unorganized points (after view-frustum and backface culling from $148,138$ points) representing a male in Figure 39(a) and the hardware-accelerated rendering using square splats (Figure 39(b)), circular splats (Figure 39(c)), and high-quality EWA splats (Figure 39(d)).

The most time-consuming process in our rendering algorithm is the generation of additional points. Of course, the cost of the rendering algorithm is linearly proportional to the number of additional points to generate. This directly shows that our algorithm is highly output-sensitive. It is self-evident that the cost to generate one additional point also depends on the type of implicit surface that was reconstructed from the unorganized point set. For example, we can inspect about $2,000$ points per second for implicit surfaces reconstructed by the partition of unity variational method presented in Chapter 2 with the parameters $T_{min} = 50$, $T_{max} = 100$, $\beta = 1$, $\upsilon_3(x) = -6x^5 + 15x^4 - 10x^3 + 1$, and a biharmonic basic function. Recall, that the number of points per second that can be inspected does not depend on the number of points in the unorganized point set $\mathcal{P}$ due to the constant evaluation time of the defining function (see Section 2.6 of Part I). However, for multi-level partition of unity implicits [118], the generation of additional points is much faster, since the local defining functions in the local subdomains are quadratic polynomials that are much faster to evaluate than the radial basis functions when using the partition of unity variational method.

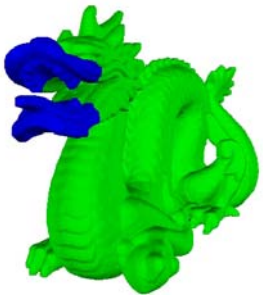
(a) 10,352 splats.

(b) $>$ 400 fps.



(c) 53,211 splats.

(d) $>$ 90 fps.



(e) 248,645 splats.

(f) $>$ 20 fps.

(g) 14,517 additional points at $>$ 0.1 fps.

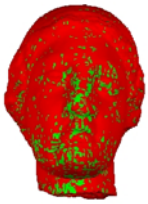
(h) Close-up of (f).

(i) Close-up of (g).

Figure 37: Traversing the hierarchy and generating additional points where required for the Stanford Dragon.

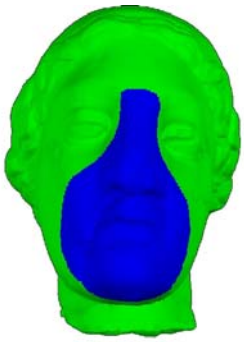
(a) 13,193 splats.

(b) $>$ 400 fps.



(c) 25,211 splats.

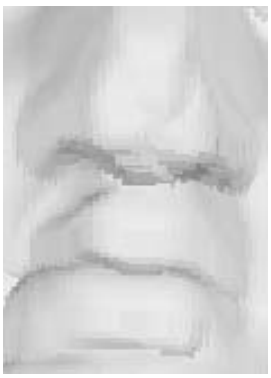
(d) $>$ 200 fps.



(e) 83,711 splats.

(f) $>$ 50 fps.

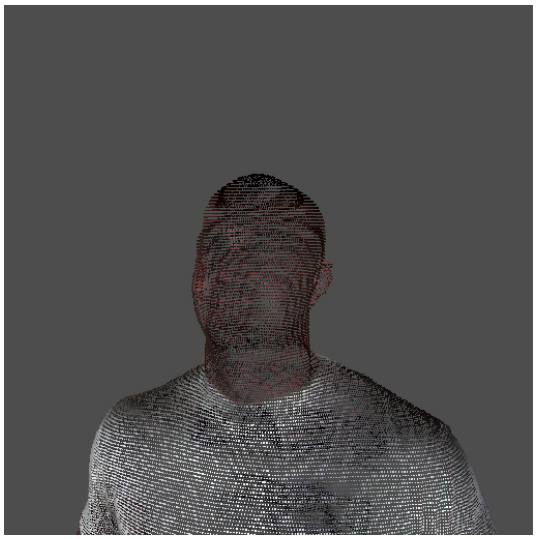
(g) 17,212 additional points at $>$ 0.1 fps.



(h) Close-up of (f).

(i) Close-up of (g).

Figure 38: Traversing the hierarchy and generating additional points where required for the Cyberware Igea.



(a) 38,221 unorganized points.

(b) Square splats at > 100 frames per second.

(c) Circular splats at > 100 frames per second.

(d) EWA splats at > 8 frames per second.

Figure 39: Different types of splats.

5.5 Conclusions

In this chapter, we presented a new point-based rendering technique where the implicit surface is rendered view-dependently in an output-sensitive multiresolution manner using points as rendering primitive. The initial unorganized point set is used to establish a multiresolution representation as a bounding sphere hierarchy, and a reconstructed implicit surface is used to generate additional points through local ray-casting when the initial unorganized point set does not provide enough detail for rendering. In our point-based rendering technique, different splatting techniques can be used for rendering, such as hardware splats or high-quality EWA splats.

Note the conceptual difference of our new point-based rendering technique compared to standard acceleration techniques for ray-tracing and ray-casting: we apply forward warping in regions where the unorganized point set is sufficiently dense and does not create holes in the output image. In the other regions, no ray needs to be cast in order to identify the sphere which is associated to the corresponding point as this is a prerequisite. Rays are only cast locally inside the sphere in order to generate additional points when the projected size of the sphere on the screen exceeds a threshold.

The required time to generate the additional points is strongly related to the type of the reconstructed implicit surface. The faster the evaluation of the defining function of the implicit surface, the faster is the generation of the additional points. Consequently, we believe that the new point-based rendering technique is particularly well adapted for multi-level partition of unity implicits [118], since the defining function can be evaluated very quickly. Furthermore, the point-based rendering technique is better suited for large unorganized point sets, since the more points are contained in the initial unorganized point set, the fewer additional points have to be generated.

The most time-consuming process of our rendering algorithm is still the generation of additional points. Hence, when the viewing parameters are changed, a high threshold τ should be used in order to quickly provide a visual feedback. On the other hand, when the viewing parameters remain unchanged, the threshold should be decreased in order to increase the visual quality by generating additional points. We intend to automatically adapt the threshold τ to ensure a given frame rate. In order to further accelerate our rendering algorithm, we strive to integrate the additionally generated points in the bounding sphere hierarchy for the following frames with different viewing parameters in order to exploit frame-to-frame coherence.

Note that the additional surface points can be generated in parallel since the bounding spheres in the leaf nodes can be considered independently from each other, and since no connectivity problems have to be resolved in our point-based rendering technique. Note also the particular importance of the culling techniques that are used in our technique: no additional points have to be generated in areas that are culled away anyway.

Chapter 6

Differential Point Rendering of Implicit Surfaces

6.1 Overview

In this chapter, we present another point-based rendering technique for a point set $\mathcal{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_N\}$ that uses the reconstruction of an implicit surface $\mathcal{S}$. This point-based rendering technique is still under development, and at the moment it is only appropriate for quasi-uniformly distributed point sets. Nevertheless, we felt that the information given in this chapter is useful and interesting.

The rendering technique is based on differential point rendering by Kalaiah and Varshney [85, 86]. For every point in the point set $\mathcal{P}$, we create a *differential point* by adding local differential geometry using the reconstructed implicit surface $\mathcal{S}$. More precisely, by local differential geometry for a point $\boldsymbol{p}_i \in \mathcal{S}$, we understand the *principal directions* $\boldsymbol{u}_{\boldsymbol{p}_i}$ and $\boldsymbol{v}_{\boldsymbol{p}_i}$ as well as the associated *principal curvatures* $u_{\boldsymbol{p}_i}$ and $v_{\boldsymbol{p}_i}$. Recall, that the principal directions are the directions of the greatest and least curvatures, and the associated principal curvatures are the respective curvature amounts in these directions. Since the differential points are rendered as fragment-shaded rectangles that lie on the tangent plane of the point $\boldsymbol{p}$, we use the principal directions and principal curvatures to determine the rectangle's extent. In flat regions or regions of low curvature of the surface, a differential point approximates larger vicinities than in regions of high curvature, and thus the rectangle has a larger extent. The differential points are rendered as fragment-shaded rectangles being perpendicular to the differential point's normal. Consequently, the implicit surface is rendered efficiently as a collection of local neighborhoods without requiring any connectivity information.

This chapter is organized as follows. First, in Section 6.2, we present the preprocessing phase. After an implicit surface $\mathcal{S}$ is reconstructed from the unorganized point set $\mathcal{P}$, we show how the implicit surface $\mathcal{S}$ is used to extract the principal directions and principal curvatures for a point $\boldsymbol{p} \in \mathcal{S}$, and how this is used to create the differential points from a point set $\mathcal{P}$. Then, in Section 6.3, we present the rendering phase where we show how the differential points can be rendered on programmable graphics hardware. Finally, we present an illustrative example in Section 6.4 as well as qualitative and quantitative results in Section 6.5 before we conclude in Section 6.6.

6.2 Preprocessing Phase

6.2.1 Implicit Surface Reconstruction

In the first step of the preprocessing phase, an implicit surface $\mathcal{S}$ is reconstructed, so that it is faithful to the point set $\mathcal{P}$. Any surface reconstruction method of Part I that creates at least C^2 continuous surfaces with an explicitly given defining function f can be used. The implicit surface has to be at least C^2 continuous, because the extraction of the principal directions and principal curvatures for a point $\boldsymbol{p} \in \mathcal{S}$ requires the second derivatives of the defining function f of the implicit surface $\mathcal{S}$ as we will see in the following subsection.

For all the examples in this chapter, we reconstructed an implicit surface using the partition of unity variational method presented in Chapter 2 of Part I. In order to guarantee a C^2 continuous implicit surface, we used the triharmonic basic function $\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|^3$ for the local reconstructions as well as the decay function $v_3(x) = -6x^5 + 15x^4 - 10x^3 + 1$ involved in the partition of unity blending. Furthermore, we used only one interior normal constraint per point since we experienced the second derivatives (and hence the principal directions and curvatures) to be less oscillatory. The derivation of the first and second derivatives of the defining function for implicit surfaces created by the partition of unity variational method can be found in Appendix A.

6.2.2 Extracting the Principal Directions and Curvatures of an Implicit Surface

In the initial differential point rendering technique [85, 86], Kalaiah and Varshney extract the principal directions and curvatures from parametric surfaces [31], triangular meshes [151], or NURBS surfaces that are fit to triangular meshes [93]. In this section, we show how to extract the principal directions and curvatures for a point $\boldsymbol{p} = [x, y, z]^T$ on the implicit surface $\mathcal{S}$, i.e. $\boldsymbol{p} \in \mathcal{S}$. To this end, consider the defining function $f : \mathbb{R}^3 \to \mathbb{R}$ of the implicit surface $\mathcal{S} = \{\boldsymbol{x} \in \mathbb{R}^3 : f(\boldsymbol{x}) = 0\}$. Recall, that the normal $\boldsymbol{n}$ of a point $\boldsymbol{p}$ is defined by the gradient of the defining function

$$\boldsymbol{n} = \nabla f(\boldsymbol{p}) = \left[\frac{\partial f}{\partial x}(\boldsymbol{p}), \frac{\partial f}{\partial y}(\boldsymbol{p}), \frac{\partial f}{\partial z}(\boldsymbol{p})\right]. \tag{61}$$

In order to derive second-order local geometry for the determination of the principal directions and curvatures, we require the *Hessian matrix* $\boldsymbol{H}$ of the second derivatives of the defining function f:

$$\boldsymbol{H} = \nabla \boldsymbol{n} = \begin{pmatrix} \frac{\partial \boldsymbol{n}}{\partial x}(\boldsymbol{p}) \\ \frac{\partial \boldsymbol{n}}{\partial y}(\boldsymbol{p}) \\ \frac{\partial \boldsymbol{n}}{\partial z}(\boldsymbol{p}) \end{pmatrix} = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2}(\boldsymbol{p}) & \frac{\partial^2 f}{\partial x \partial y}(\boldsymbol{p}) & \frac{\partial^2 f}{\partial x \partial z}(\boldsymbol{p}) \\ \frac{\partial^2 f}{\partial x \partial y}(\boldsymbol{p}) & \frac{\partial^2 f}{\partial y^2}(\boldsymbol{p}) & \frac{\partial^2 f}{\partial y \partial z}(\boldsymbol{p}) \\ \frac{\partial^2 f}{\partial x \partial z}(\boldsymbol{p}) & \frac{\partial^2 f}{\partial y \partial z}(\boldsymbol{p}) & \frac{\partial^2 f}{\partial z^2}(\boldsymbol{p}) \end{pmatrix} \tag{62}$$

We use a local parameterization to extract the principal directions and curvatures on a point $\boldsymbol{p} \in \mathcal{S}$ with an associated normal $\boldsymbol{n}$ and a Hessian Matrix $\boldsymbol{H}$. For the illustration of the following calculations, consider Figure 40.

Let us now approximate the defining function f of the implicit surface $\mathcal{S}$ in a small vicinity of $\boldsymbol{p}$ by using a small vector $\boldsymbol{w}$ for a second degree Taylor expansion with an approximation

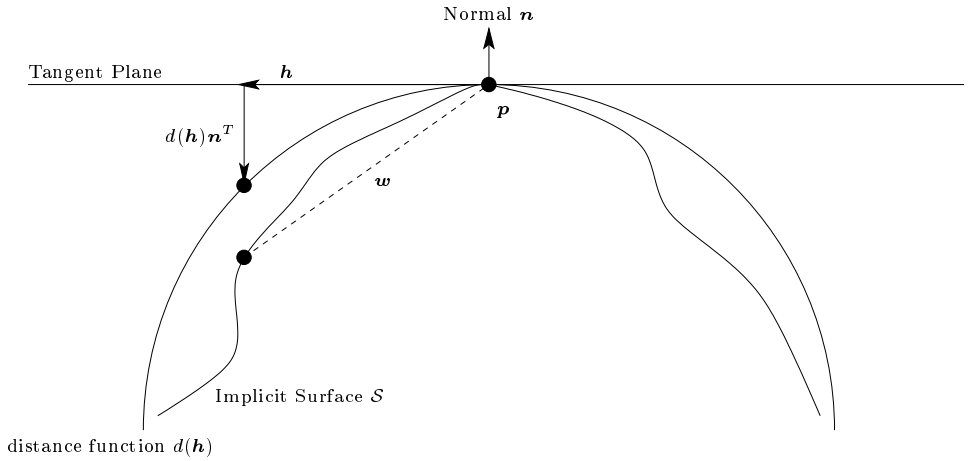


Figure 40: Calculating the curvature of an implicit surface $\mathcal{S}$ with defining function f.

error $o(\|\boldsymbol{w}\|^2)$:

$$f(\boldsymbol{p}+\boldsymbol{w}) = f(\boldsymbol{p}) + \boldsymbol{n}^T \bullet \boldsymbol{w} + \frac{1}{2}\boldsymbol{w} \bullet (\boldsymbol{H}\boldsymbol{w}) + o(\|\boldsymbol{w}\|^2) \tag{63}$$

Since $\boldsymbol{p} \in \mathcal{S}$, the defining function of the implicit surface $\mathcal{S}$ in $\boldsymbol{p}$ is $f(\boldsymbol{p}) = 0$, and we find

$$f(\boldsymbol{p}+\boldsymbol{w}) = \boldsymbol{n}^T \bullet \boldsymbol{w} + \frac{1}{2}\boldsymbol{w} \bullet (\boldsymbol{H}\boldsymbol{w}) + o(\|\boldsymbol{w}\|^2). \tag{64}$$

Now, we split the small vector $\boldsymbol{w}$ into a vector $\boldsymbol{h}$ that is orthogonal to $\boldsymbol{n}$, i.e. $\boldsymbol{n}^T \bullet \boldsymbol{h} = 0$, and a distance function $d : \mathbb{R}^3 \to \mathbb{R}$ to the tangent plane in $\boldsymbol{p}$:

$$\boldsymbol{w} = \boldsymbol{h} + d(\boldsymbol{h})\boldsymbol{n}^T \tag{65}$$

We want to determine the distance function d so that $f(\boldsymbol{p} + \boldsymbol{h} + d(\boldsymbol{h})\boldsymbol{n}^T) = 0$. By inserting $\boldsymbol{w}$ into equation (64) and setting $d(\boldsymbol{h}) = o(\boldsymbol{h})$ since per definition $d'(\boldsymbol{0}) = 0$, we find

$$\boldsymbol{n}^T \bullet (\boldsymbol{h} + d(\boldsymbol{h})\boldsymbol{n}^T) + \frac{1}{2}\Big((\boldsymbol{h} + d(\boldsymbol{h})\boldsymbol{n}^T) \bullet \big(\boldsymbol{H}(\boldsymbol{h} + d(\boldsymbol{h})\boldsymbol{n}^T)\big)\Big) = o(\|\boldsymbol{h}\|^2), \tag{66}$$

that we develop to

$$\begin{aligned} &\boldsymbol{n}^T \bullet (\boldsymbol{h} + d(\boldsymbol{h})\|\boldsymbol{n}\|^2 + \frac{1}{2}\Big(\boldsymbol{h} \bullet (\boldsymbol{H}\boldsymbol{h})\Big) + \frac{1}{2}\Big(\boldsymbol{h} \bullet \big(\boldsymbol{H}d(\boldsymbol{h})\boldsymbol{n}^T\big)\Big) \\ &+\frac{1}{2}\Big(d(\boldsymbol{h})\boldsymbol{n}^T \bullet (\boldsymbol{H}\boldsymbol{h})\Big) + \frac{1}{2}\Big(d(\boldsymbol{h})\boldsymbol{n}^T \bullet (\boldsymbol{H}d(\boldsymbol{h})\boldsymbol{n}^T)\Big) = o(\|\boldsymbol{h}\|^2). \end{aligned} \tag{67}$$

Since $\boldsymbol{n}^T \bullet \boldsymbol{h} = 0$, and since we can neglect the constant term with respect to $\boldsymbol{h}$, we find

$$d(\boldsymbol{h})\|\boldsymbol{n}\|^2 + \frac{1}{2}\Big(\boldsymbol{h} \bullet (\boldsymbol{H}\boldsymbol{h})\Big) + d(\boldsymbol{h})\Big(\boldsymbol{h} \bullet (\boldsymbol{H}\boldsymbol{n}^T)\Big) = o(\|\boldsymbol{h}\|^2), \tag{68}$$

and the second order approximation of d is

$$d(\boldsymbol{h}) = -\frac{\boldsymbol{h} \bullet (\boldsymbol{H}\boldsymbol{h})}{2\|\boldsymbol{n}\|^2}. \tag{69}$$

Now, we want to find the maximum and minimum of $\frac{\boldsymbol{h}\bullet(\boldsymbol{H}\boldsymbol{h})}{2\|\boldsymbol{h}\|^2}$ for $\boldsymbol{h}$ orthogonal to $\boldsymbol{n}$. This implies that the derivative of $\frac{\boldsymbol{h}\bullet(\boldsymbol{H}\boldsymbol{h})}{2\|\boldsymbol{h}\|^2}$ has a component orthogonal to $\boldsymbol{n}$ with the value 0, and hence

$$\boldsymbol{H}\boldsymbol{h} = \mu\boldsymbol{h} + \lambda\boldsymbol{n}^T. \tag{70}$$

In order to determine the principal directions and curvatures, we have to find the eigenvectors with associated non-zero eigenvalues of

$$\boldsymbol{H}\boldsymbol{h} - \frac{(\boldsymbol{H}\boldsymbol{h}) \bullet \boldsymbol{n}^T}{\|\boldsymbol{n}\|^2}\boldsymbol{n}^T, \tag{71}$$

that can be written more compactly as

$$\boldsymbol{H}\left(\boldsymbol{I} - \frac{\boldsymbol{n}^T \bullet \boldsymbol{n}}{\|\boldsymbol{n}\|^2}\right). \tag{72}$$

Summing up, the principal directions $\boldsymbol{u_p}$ and $\boldsymbol{v_p}$ are the eigenvectors of the matrix (72) with associated non-zero eigenvalues, and the principal curvatures $u_{\boldsymbol{p}}$ and $v_{\boldsymbol{p}}$ are the corresponding non-zero eigenvalues.

6.2.3 Creating the Differential Points

In our differential point rendering technique, we create one differential point for every point $\boldsymbol{p}_i$ of the point set $\mathcal{P}$. To this end, we first calculate the principal directions $\boldsymbol{u}_{\boldsymbol{p}_i}$ and $\boldsymbol{v}_{\boldsymbol{p}_i}$ as well as the principal curvatures $u_{\boldsymbol{p}_i}$ and $v_{\boldsymbol{p}_i}$ for every point $\boldsymbol{p}_i$ by using the reconstructed implicit surface $\mathcal{S}$ as explained in the preceding subsections. The principal directions are used to determine the orientation of the rectangle of the differential point to be rendered, and the principal curvatures are used to determine the extents of the rectangle in the principal directions. In order to avoid disproportional aspect ratios of the rectangle, we delimit the extents of the rectangle to $[E_{min}, E_{max}]$ with $E_{min} < E_{max}$. Using a maximum principal width χ with $0 \leq \chi \leq 1$ according to Kalaiah and Varshney, the extents $u^{extent}_{\boldsymbol{p}_i}$ and $v^{extent}_{\boldsymbol{p}_i}$ in the principal directions $\boldsymbol{u}_{\boldsymbol{p}_i}$ and $\boldsymbol{v}_{\boldsymbol{p}_i}$, respectively, are calculated as follows:

$$u^{extent}_{\boldsymbol{p}_i} = \max\left(E_{min}, \min\left(\frac{\sqrt{2\chi - \chi^2}}{|u_{\boldsymbol{p}_i}|}, E_{max}\right)\right) \tag{73}$$

$$v^{extent}_{\boldsymbol{p}_i} = \max\left(E_{min}, \min\left(\frac{\sqrt{2\chi - \chi^2}}{|v_{\boldsymbol{p}_i}|}, E_{max}\right)\right) \tag{74}$$

Then, we define the positions of the four corners of the rectangle as follows:

$$\boldsymbol{r}_{i_1} = \boldsymbol{p}_i + u_{\boldsymbol{p}_i}^{extent}\boldsymbol{u}_{\boldsymbol{p}_i} + v_{\boldsymbol{p}_i}^{extent}\boldsymbol{v}_{\boldsymbol{p}_i} \tag{75}$$

$$\boldsymbol{r}_{i_2} = \boldsymbol{p}_i + u_{\boldsymbol{p}_i}^{extent}\boldsymbol{u}_{\boldsymbol{p}_i} - v_{\boldsymbol{p}_i}^{extent}\boldsymbol{v}_{\boldsymbol{p}_i} \tag{76}$$

$$\boldsymbol{r}_{i_3} = \boldsymbol{p}_i - u_{\boldsymbol{p}_i}^{extent}\boldsymbol{u}_{\boldsymbol{p}_i} - v_{\boldsymbol{p}_i}^{extent}\boldsymbol{v}_{\boldsymbol{p}_i} \tag{77}$$

$$\boldsymbol{r}_{i_4} = \boldsymbol{p}_i - u_{\boldsymbol{p}_i}^{extent}\boldsymbol{u}_{\boldsymbol{p}_i} + v_{\boldsymbol{p}_i}^{extent}\boldsymbol{v}_{\boldsymbol{p}_i} \tag{78}$$

Finally, we determine the normals in the four corners of the rectangle to be rendered by simply using the gradient of the reconstructed implicit surface $\mathcal{S}$ resulting in the normals $\boldsymbol{n}_{\boldsymbol{r}_{i_1}}$, $\boldsymbol{n}_{\boldsymbol{r}_{i_2}}$, $\boldsymbol{n}_{\boldsymbol{r}_{i_3}}$, and $\boldsymbol{n}_{\boldsymbol{r}_{i_4}}$. Summing up, a differential point is simply represented by the four corners of the rectangle as well as the corresponding normals.

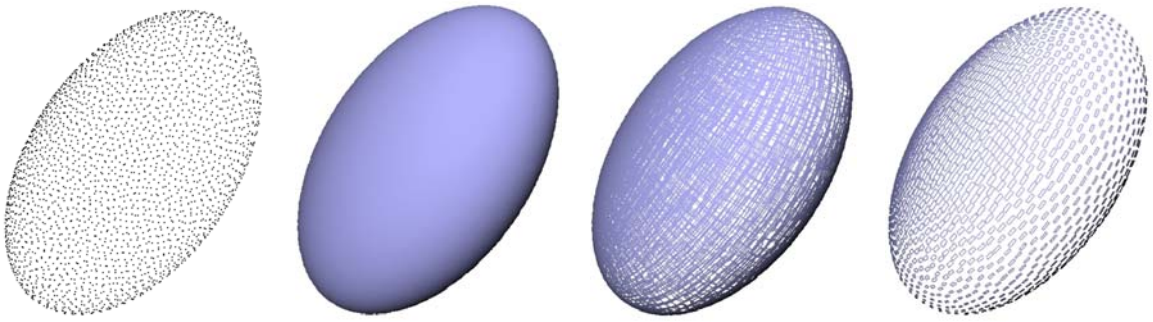
When creating the differential points, we choose the maximum principal width χ so that the rectangles of all the differential points overlap sufficiently without leaving holes in the surface coverage. Note that even though this results in a complete surface representation, there are redundant differential points that are completely covered by neighboring differential points. Consequently, redundant differential points can be pruned away in a simplification process according to Kalaiah and Varshney [86]. Note also that a too high maximum principal width χ creates too large rectangles for the differential points, and artefacts appear especially at the silhouette and at surface parts with a high curvature represented by too few points in the point set $\mathcal{P}$. We believe that a curvature-driven sampling of the differential points might overcome this drawback.

6.3 Rendering Phase

After the differential points for all points $\boldsymbol{p}_i$ have been created, we render the differential points as fragment-shaded rectangles on programmable graphics hardware since curved primitives are not supported by current graphics hardware. In contrast to Kalaiah and Varshney, who select the best fitting normal map out of 256 precomputed normal maps according to the principal curvatures, we interpolate the normals at the corners of the differential point's rectangle using *vertex shaders* and *fragment shaders*. More precisely, for each vertex at the rectangle's corners, the vertex shader copies the normal at the vertex into a texture coordinate. Then, the fragment shader interpolates the normals represented by the texture coordinates for each fragment and normalizes them by using a *cube map texture*. Note that even though we render the differential points as overlapping fragment-shaded rectangles, no information about connectivity is required.

6.4 Example

In this section, we discuss a very simple example in order to show how implicit surfaces can be rendered by differential points. For a more explicit illustration, consider the $2,616$ uniformly distributed points on an ellipsoid in Figure 41(a). Starting from these points, we reconstructed an implicit surface by using the partition of unity variational method presented in Chapter 2 of Part I. The differential point rendering of the reconstructed implicit surface is shown in Figure 41(b), where the differential points were rendered as fragment-shaded rectangles, and



(a) 2,616 uniformly distributed points on an ellipse.

(b) Differential point rendering.

(c) Wireframe rendering.

(d) Wireframe rendering with scaled rectangles.

Figure 41: Differential point rendering of a reconstructed implicit surface from uniformly distributed points.

the corresponding *wireframe* rendering is shown in Figure 41(c). In order to illustrate the principle directions and principal curvatures of the differential points, we scaled the rectangles representing the differential points, and the results of the wireframe rendering can be seen in Figure 41(d). In this simple example, it can be observed that the extents of the rectangles are larger in regions of low curvature and smaller in regions of high curvature. Note also, that the energy minimizing reconstructed surface using the partition of unity variational method is not a mathematically regular ellipsoid, since it can be seen that the principal directions are not parallel to the coordinate axes.

6.5 Results

In this section, we present some quantitative and qualitative results that we obtained on an Intel Pentium 1.7 GHz with 512 MB of RAM and a GeForce3 graphics board running Linux. For the computation of the eigenvectors and eigenvalues involved in the determination of the principal curvatures and directions, we used the GNU Scientific Library package [61].

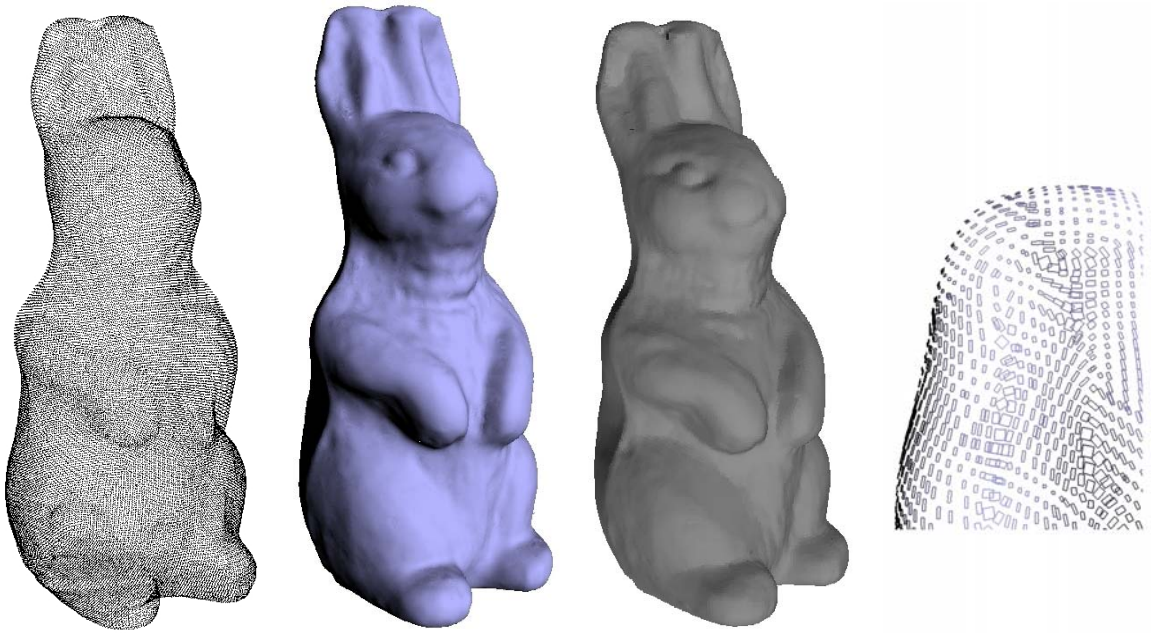
First, we present the results of the preprocessing phase. Of course, the required time for the reconstruction of the implicit surface depends on the chosen reconstruction technique. In this chapter, we exclusively use the partition of unity variational method presented in Chapter 2 of Part I. We refer to the obtained results therein, but note that the number of constraints is decreased by a third since we only use one normal constraint per point.

Of course, the cost to extract the differential points is linearly proportional to the number of differential points to create. The cost to determine the Hessian matrix and the normal in a point depends on the type of implicit surface that was reconstructed from the point set, whereas the cost to determine the principal directions and principal curvatures is independent of the type of implicit surface. For an implicit surface reconstructed by the partition of unity variational method with the parameters $T_{min} = 50$, $T_{max} = 100$, $\beta = 1.2$, $\upsilon_3(x) = -6x^5+15x^4-10x^3+1$, and a triharmonic basic function, we can create about $5,000$ differential

points per second. Recall, that the number of differential points that can be created per second does not depend on the number of points in the point set due to the constant evaluation time of the defining function (see Section 2.6 of Part I).

Second, the results of the rendering phase are presented. Since the involved vertex shaders and fragment shaders are very simple, rendering a differential point is not much more expensive than rendering a rectangle by using the *fixed-function pipeline*, and we can render about $1M$ differential points per second on our GeForce 3 graphics board.

In the following, we present some visual results. Starting from the $67,038$ points of the Cyberware Rabbit (Figure 42(a)), we created a differential point for each point in the point set. The result of the differential point rendering can be seen in Figure 42(b), and compared to a hardware splat rendering shown in Figure 42(c), the surface appears much smoother. In order to illustrate the extent of the rectangles representing the differential points, we scaled the rectangles in a close-up of the Cyberware Rabbit's ear as shown in Figure 42(d).



(a) $67,038$ quasi-uniformly distributed points.

(b) Differential point rendering (16 fps).

(c) Hardware splat rendering (64 fps).

(d) Close-up of the wireframe rendering with scaled rectangles.

Figure 42: Differential point rendering of the Cyberware Rabbit.

6.6 Conclusions

In this chapter, we presented another point-based rendering technique for an implicit surface reconstructed from a point set $\mathcal{P}$ that is running on programmable graphics hardware. We extract the local differential geometry for every point $\boldsymbol{p}_i$ in the point set $\mathcal{P}$ using a reconstructed implicit surface, store this information in so-called differential points, and then render the differential points as fragment-shaded rectangles. Note that by using *displacement mapping* proposed on some recent graphics boards [105], the rendering quality can be increased even further.

By packing more information in each point of the initial point set $\mathcal{P}$ resulting in a differential point, a lower number of differential points is required to represent the initial point set $\mathcal{P}$, and once the differential points are extracted, the reconstructed implicit surface is no longer required. This reduces the required memory to store the surface representation and ensures a lower bus traffic between the CPU and the graphics board during rendering. Furthermore, like for all point-based rendering techniques, no information about the connectivity of the differential points is required, and the differential points can be structured in a hierarchical data structure such as the bounding sphere hierarchy that we presented in Chapter 5.

At the moment, the point-based rendering technique based on differential point rendering is not well adapted for non-uniformly distributed point sets. This is due to the fact, that we create one differential point for every point in the point set. In regions where few points in the point set define surfaces with high curvatures, there might either not be enough information to cover the surface with differential points, or the extent of the rectangles will be too high, and the second-order surface approximation as well as artefacts along the silhouette become visible.

Consequently, we believe that there are better solutions than creating a differential point for every initial point $\boldsymbol{p}_i$ in the point set $\mathcal{P}$, such as using a curvature-driven sampling technique for implicit surfaces and creating a differential point for every sample. We plan to develop a curvature-driven sampling technique by adapting the particle-based sampling technique for implicit surfaces proposed by Witkin and Heckbert [166] and its optimized implementation by Hart et al. [76].

Part III

Applications

The fundamentals of reconstructing and rendering of implicit surfaces from large unorganized point sets presented in Part I and Part II give rise to a variety of applications in various domains. In this part, we present an incomplete list of potential applications that use the fundamentals of the first two parts of this thesis.

In Chapter 7 we show how the partition of unity variational method and the hierarchical partition of unity variational method can be used in other reconstruction applications besides implicit surfaces. In particular, in Section 7.1, we present a new class of procedural solid texture that can be reconstructed from unorganized points. Furthermore, we present two applications in 2D, namely the reconstruction of terrains from 2D contour lines and reconstruction in 2D image processing, in Sections 7.2 and 7.3, respectively, before we conclude in Section 7.4.

In Chapter 8, we present a new interactive environment for constructive texturing of surfaces of arbitrarily defined 3D objects (including, of course, implicit surfaces). A user can texture the surface by defining space partitions that are combined using constructive texturing, and specifying attributes that are applied in the space partition. The partition of unity variational method can be used to define space partitions amongst other primitives. When specifying the attributes, the new class of procedural solid textures can be used as well. In order to give an interactive feedback, a point-based multiresolution representation of the surface is used that is not only exploited for rendering, but also for the evaluation of the texture.

Chapter 7

Further Reconstruction Applications

7.1 Procedural Solid Textures from Points

In this section, we present a new method to define procedural solid textures from an unorganized point set $\mathcal{P}$ that uses either the partition of unity variational method or the hierarchical partition of unity variational method introduced in Part I. Unorganized point sets coming from range scanners, for example, are often delivered with a *material attribute* a_i for every point $\boldsymbol{p}_i$. In the following, we will refer to material attributes as a genus for ambient, diffuse and specular color channels, reflectance and transmittance coefficients, and others. In order to define a procedural solid texture for the material attributes, we regard every material attribute a_i for a point $\boldsymbol{p}_i \in \mathcal{P}$ separately and try to find a *texture function* f_a that is satisfying the N constraints

$$f_a(\boldsymbol{p}_i) = a_i \qquad i = 1, ..., N. \tag{79}$$

In order to reconstruct a texture function f_a from a high number of constraints N, we use the partition of unity method and solve the interpolation problems locally before sticking them together as described in Part I. The global domain is subdivided into M local subdomains with N_m points each for $m = 1 \dots M$, either by using the octree domain decomposition (Chapter 2) or the binary tree domain decomposition (Chapter 3). We determine the M local interpolation functions f_{a_m} for $m = 1 \dots M$ by using variational techniques with radial basis functions according to Turk and O'Brien [156] as well, but since we assume, that there are at least two material attributes a_i and a_j with $a_i \neq a_j$ for $i \neq j$ in every local interpolation problem, we do not need any additional off-surface constraints (such as normal constraints for example) in order to avoid the trivial solution. Consequently, when reconstructing the texture function, there are only a third of the numbers of constraints per local subdomain compared to reconstructing the defining function of the implicit surface. Recall, that the local functions interpolating the attributes of the points $\boldsymbol{p}_i$ for $i = 1 \dots N_m$ are defined by functions of the class

$$f_{a_m}(\boldsymbol{x}) = \sum_{i=1}^{N_m} \omega_i \phi(\boldsymbol{x} - \boldsymbol{p}_i) + \sum_{\alpha=1}^{Q} \pi_\alpha p_\alpha(\boldsymbol{x}). \tag{80}$$

After choosing a suitable basic function $\phi : [0, \infty) \to \mathbb{R}$ and the order of the polynomial, the coefficients ω_i and π_α can be determined by solving the following linear system:

$$\begin{aligned}
\boldsymbol{A}\boldsymbol{x} &= \boldsymbol{b} \\
\boldsymbol{A} &= \begin{bmatrix} \boldsymbol{\Phi} & \boldsymbol{P}^T \\ \boldsymbol{P} & 0 \end{bmatrix} \\
\boldsymbol{\Phi} &= \left[\phi(||\boldsymbol{p}_i, \boldsymbol{p}_j||)\right] \qquad \text{where } i = 1 \ldots N_m, \qquad j = 1 \ldots N_m \\
\boldsymbol{P} &= [p_\alpha(\boldsymbol{p_i})] \qquad \text{where } i = 1 \ldots N_m, \qquad \alpha = 1 \ldots Q \\
\boldsymbol{x} &= [\omega_1, \omega_2, \ldots, \omega_{N^m}, \pi_1, \pi_2, \ldots, \pi_Q]^T \\
\boldsymbol{b} &= \left[a_1, a_2, \ldots, a_{N^m}, \underbrace{0, 0, \ldots, 0}_{Q \text{ times}}\right]^T
\end{aligned} \tag{81}$$

In order to give some results of the new procedurally defined solid textures, let us now suppose, without the loss of generality, that we have material attributes $\boldsymbol{m}_i$ containing diffuse RGB color values $\boldsymbol{m}_i = [r_i, g_i, b_i]^T$ for every point $\boldsymbol{p}_i \in \mathcal{P}$. We use the partition of unity variational method to interpolate the three procedurally defined texture functions f_r, f_g, and f_b, one for each color channel of the RGB color values:

$$\left.\begin{aligned} f_r(\boldsymbol{p}_i) &= r_i \\ f_g(\boldsymbol{p}_i) &= g_i \\ f_b(\boldsymbol{p}_i) &= b_i \end{aligned}\right\} \quad \text{for } i = 1 \ldots N \tag{82}$$

See Figure 43 for the visual results of four different examples of the reconstruction of the implicit surface and the procedurally defined solid texture of a chameleon using the partition of unity variational method. All images were ray-traced by our plugin for POVray [126] in order to get the best quality. In Figures 43(a) and 43(b), the implicit surface is reconstructed from 101,685 points, whereas in Figures 43(c) and 43(d), the implicit surface is reconstructed from only 10,000 points. The reconstruction of the texture functions is done from $101,685$ points in Figures 43(a) and 43(c), and from $10,000$ points in Figures 43(b) and 43(d).

See Table 15 for the reconstruction times for the implicit surface, the three procedurally defined solid textures, and the total reconstruction time including the domain decomposition. We used $T_{min} = 50$, $T_{max} = 100$, $\beta = 1$, a spherical distance function, and $\upsilon_3(x) = -6x^5 + 15x^4 - 10x^3 + 1$. For $101,685$ points, the domain decomposition into $M = 15,772$ domains was done in 91 seconds, and for $10,000$ points, the domain decomposition into $M = 1317$ subdomains was done in 2.76 seconds. Recall, that there are no off-surface constraints when reconstructing the procedurally defined solid textures, hence reconstructing a texture function for one color channel is faster since it involves only a third of the number of constraints per local subdomain compared to reconstructing the defining function of the implicit surface, while the number of local subdomains M remains equal.

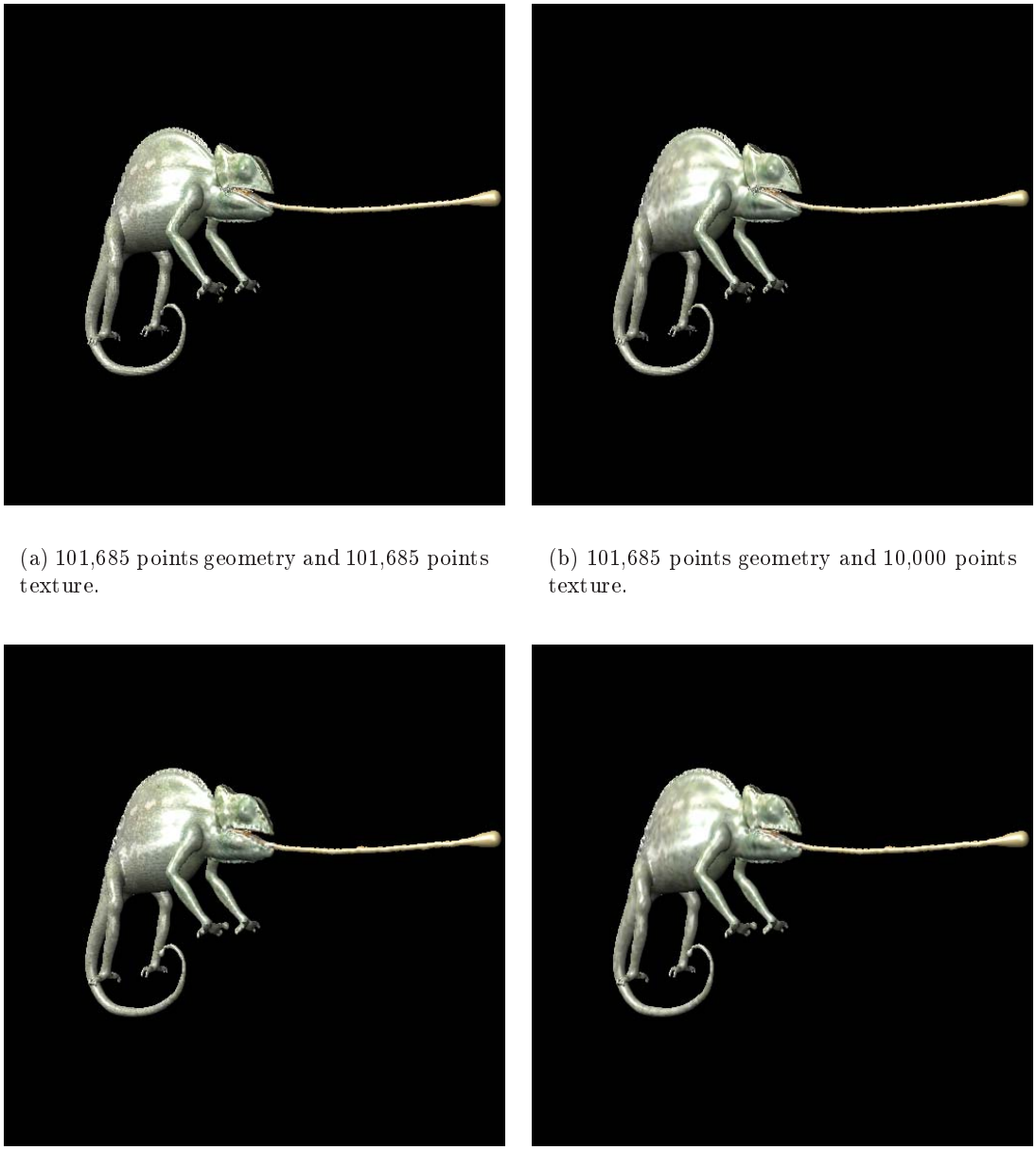
The new procedurally defined solid texture can be applied to surfaces of arbitrary 3D objects, regardless whether the surface is defined as a polygonal mesh, as a parametric or

Model	**Defining function**		**Texture function**		**Total**
	$N^{geometry}$	$t_{recon}^{geometry}$	$N^{textures}$	$t_{recon}^{textures}$	t_{total}
Chameleon	101,685	1599.47	101,685	729.57	2511.04
	101,685	1599.47	10,000	41.59	1734.82
	10,000	150.48	101,685	729.57	973.81
	10,000	150.48	10,000	41.59	197.59

Table 15: Reconstruction times in seconds for the defining function and the texture functions from different number of points.

implicit surface, or whatever.

Of course, all characteristics of solid textures still apply to this new procedurally defined solid texture. Since both the surface as a 2D manifold in 3D and the solid texture are defined in $\mathbb{R}^3$, deformations applied to the surface and to the texture conserve the geometry-texture coherence and can hence be done without distortions. See for example Figure 44 for the chameleon reconstructed using the partition of unity variational method that was twisted along the z-axis.



(a) 101,685 points geometry and 101,685 points texture.

(b) 101,685 points geometry and 10,000 points texture.

(c) 10,000 points geometry and 101,685 points texture.

(d) 10,000 points geometry and 10,000 points texture.

Figure 43: Procedural solid textures from points.

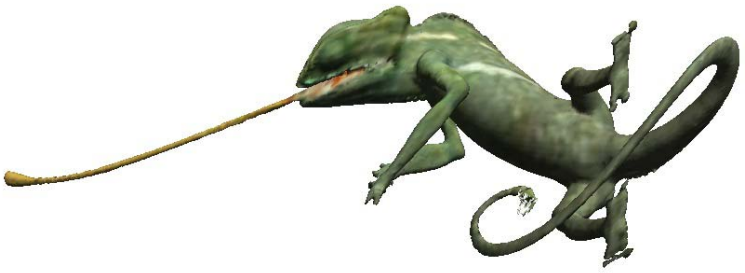


Figure 44: Twisted chameleon without texture distortion.

7.2 Heightfields

In this section, we show how the partition of unity variational method presented in Chapter 2 of Part I can be succesfully applied to reconstruct continuous heightfields from terrain data. Commonly, terrain data issued by modern acquisition devices measured from airplanes or satellites is given as height values on regular rectangular grids. Nevertheless, terrain data is often more accessible given as contour lines in 2D with an associated height value that is invariant in each contour line. In the following, we present a method to reconstruct continuous heightfields that is suited both for regular as well as scattered terrain data issued from contour lines.

Formally, the terrain data is given as an organized or unorganized point set $\mathcal{P} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_N\}$ with $\boldsymbol{p}_i \in \mathbb{R}^2$ for $i = 1 \ldots N$, with an assoicated height h_i for every point $\boldsymbol{p}_i$. Consequently, reconstructing a continuous heightfield can be considered as an interpolation problem, and a function $f : \mathbb{R}^2 \to \mathbb{R}$ satisfying the following constraints has to be found:

$$f(\boldsymbol{p}_i) = h_i \qquad i = 1, ..., N \tag{83}$$

In order to allow the reconstruction of continuous heightfields from a high number of constraints N, we use the 2D counterpart of the partition of unity variational method presented in Chapter 2 of Part I. In 2D, the entire domain is decomposed by using a *quadtree* domain decomposition method, and the local reconstruction functions are functions of the class

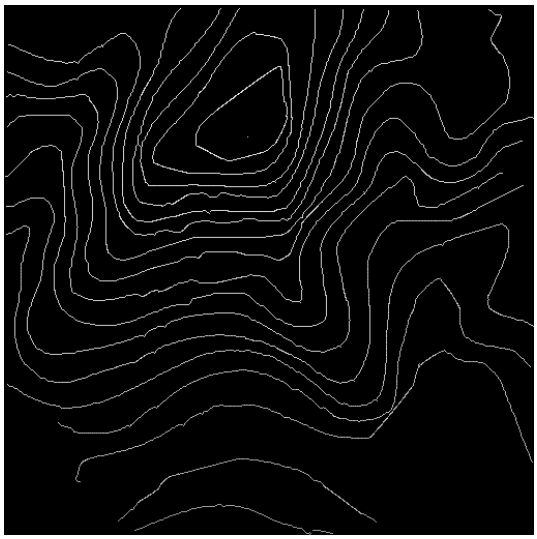
$$f(\boldsymbol{x}) = \sum_{i=1}^{N_m} \omega_i \phi(\boldsymbol{x} - \boldsymbol{p}_i) + \sum_{\alpha=1}^{Q} \pi_\alpha p_\alpha(\boldsymbol{x}), \tag{84}$$

and by using $\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|$ as basic functions together with a linear polynomial, i.e. $Q = 3$, the coefficients ω_i and π_α can again be determined by solving a linear system according to equation (82) of Section 7.1.

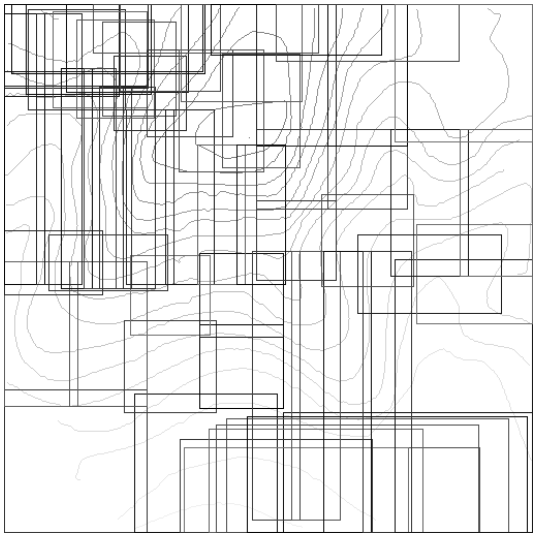
We illustrate the reconstruction of continuous heightfields using the partition of unity variationals method on a simple example. Consider the image with the contour lines grayscale-coded in the pixels in Figure 45(b) that were extracted from the map shown in Figure 45(a) with a resolution of 512^2 pixels. The lighter the gray value of a pixel, the higher is the height of the corresponding contour line. Black pixels do not reflect any information, and there are $10,028$ pixels reflecting the heights in the map, i.e. only about 4% of the pixels reflect information, and about 96% have to be reconstructed.

By using the quadtree domain decomposition method with $T_{min} = 300$, $T_{max} = 600$, $\beta = 1.2$, the cubical distance function, and $\upsilon_3(x) = -6x^5 + 15x^4 - 10x^3 + 1$, the resulting domain decomposition with $M = 64$ local subdomains can be seen in Figure 45(c). After the computation of the local reconstructions, the resulting global reconstruction using the partition of unity method can be seen in the grayscale-coded image of Figure 45(d). The total reconstruction time was 103 seconds. An example of the rendering of the resulting terrain as a polygonal mesh can be seen in Figure 46(a), and the textured terrain can be seen in Figure 46(b).

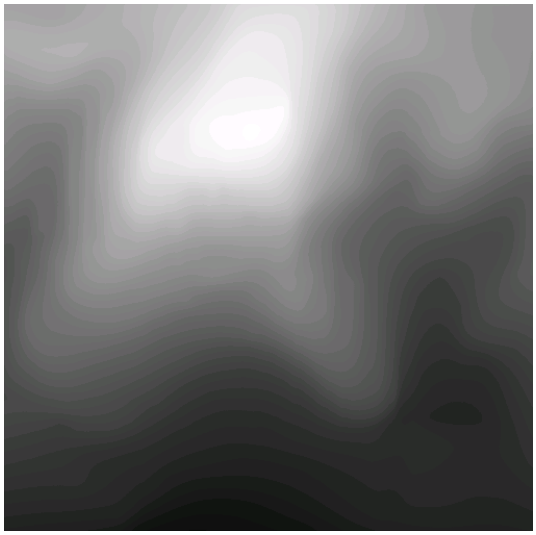
(a) Map of the contour lines.



(b) Contour lines as grayscale-coded pixels.

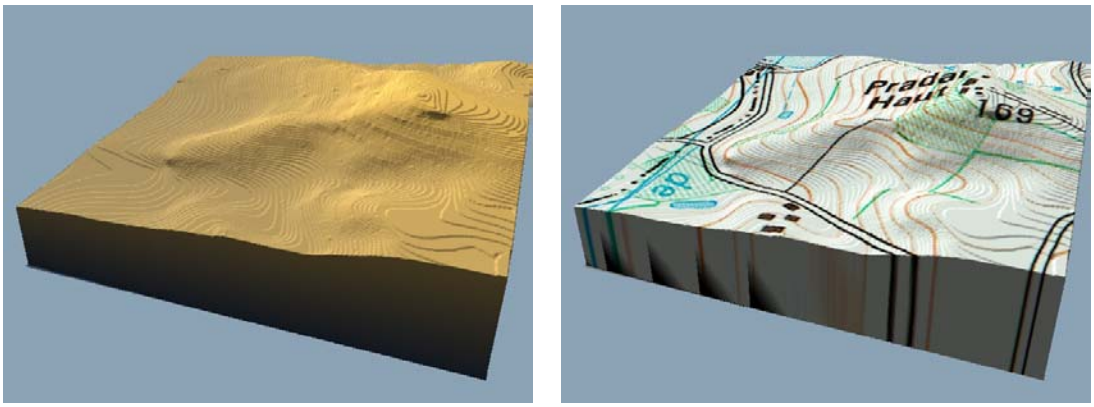


(c) Quadtree domain decomposition.



(d) Reconstructed height values.

Figure 45: Reconstruction of a continuous heightfield.



(a) Without texture.

(b) With texture.

Figure 46: Rendering of the textured terrain.

(a) Original image.

(b) Damaged image.

(c) Segmentation of damaged zones.

(d) Reconstructed image.

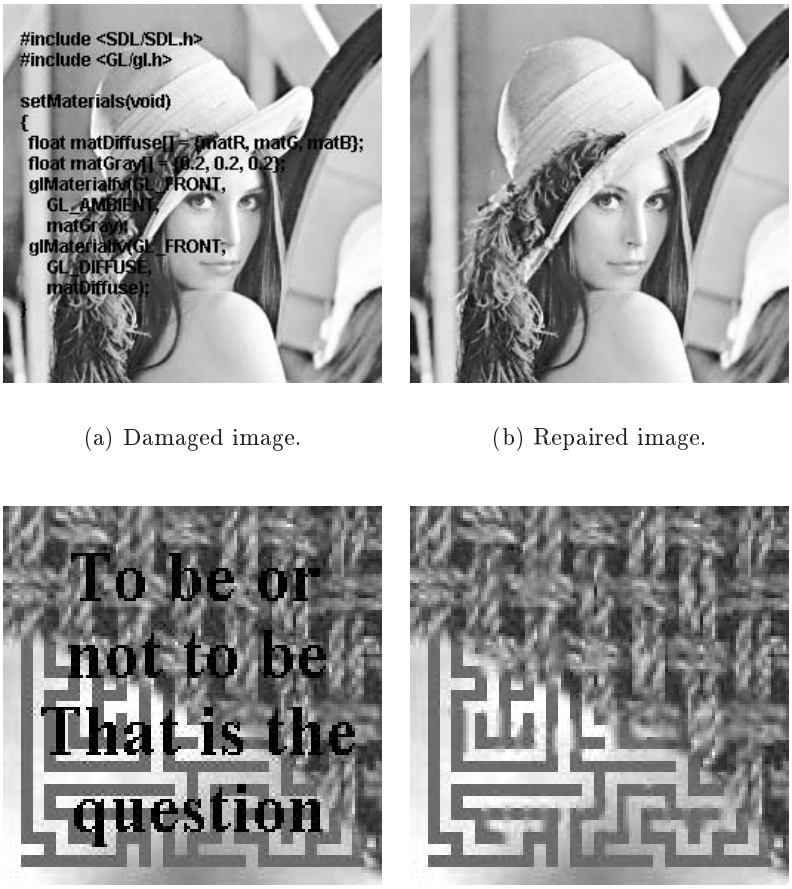
Figure 47: Example of 2D image processing using the partition of unity variational method: repairing 2D images.

7.3 2D Imaging

In this section, we show how the partition of unity variational method can be applied to 2D imaging, based on similar ideas presented in the preceding sections. Pixels of a discrete 2D image can be considered as discrete samples, and by handling every channel of a 2D image separately, a continuous function for each channel can be reconstructed by using either the partition of unity variational method or the hierarchical partition of unity variational method. This continuous function can be used for 2D image processing, such as repairing images, changing resolutions, rotations, and others. We believe that the partition of unity variational and hierarchical partition of unity variational methods are particularly useful when they are used in addition to other traditional image processing techniques.

Figure 47 shows a simple example of repairing a 2D image from incomplete data. Let us suppose that an original scanned photograph (Figure 47(a)) has been damaged by some accident (Figure 47(b)). In our implementation, we simply require the damaged pixels to be segmented (Figure 47(c)) in order to reconstruct the missing part. Note that the segmentation step can be rather simple as shown in the example, where a simple thresholding on the brightness has been used. A user-assisted process could also be defined that can be easily included in conventional image editing software. We used the partition of unity variational method to reconstruct the damaged parts of the image starting from the scattered data represented by all pixels except the segmented ones, and we handled each of the RGB color channels separately. The resulting repaired image can be seen in Figure 47(d). At most of the damaged parts of the image, the missing data was reconstructed quite well, but of course a postprocessing is required at the damaged part of the mouth, where not enough data was available.

Figure 48 shows two other examples. Again, we used the partition of unity variational method to repair the damaged images of Figures 48(a) and 48(c), and the resulting repaired images can be seen in Figures 48(b) and 48(d).



(a) Damaged image. (b) Repaired image.

(c) Damaged image. (d) Repaired image.

Figure 48: Two other examples for repairing 2D images by using the partition of unity variational method.

7.4 Conclusions

In this chapter, we have shown how the partition of unity variational method and the hierarchical partition of unity variational method can be used in other reconstruction applications besides implicit surfaces. We defined a new class of procedurally defined solid texture that can be generated from the attributes of an unorganized point set. Furthermore, we presented two applications in 2D that emphasize the generality of the fundamentals of the partition of unity variational and partition of unity variational methods. Of course, there is an unexhausted potential to use the fundamentals for other applications. For example, we believe that they are even more appropriate for real-time deformations used in character animation than compactly supported radial basis functions [92, 91]. Moreover, constrained texture mapping using variational techniques [150] could be adapted to allow a higher number of constraints.

Chapter 8

Interactive Constructive Texturing

8.1 Overview

The main advantage of solid texturing is, that it can be applied to surfaces of arbitrary 3D objects without requiring a parameterization. On the other hand, applying different solid textures at specific locations of the surface is difficult for a non-expert user. In this chapter, we present a new interactive environment for solid texturing of surfaces whatever the underlying surface representation (i.e. polygonal meshes, voxel arrays, unorganized point sets, parametric or implicit surfaces, and others).

The involved process combines two approaches previously developed in computer graphics: *constructive texturing* [141], that we shortly present in Section 8.3, and point-based representations, that we already introduced in Part II. The first approach offers a general framework for texturing objects of arbitrary type, while the second one offers a flexible way for rendering the surfaces of such objects. The combination of both approaches allows the development of an intuitive tool for applying textures to geometric shapes with interactive feedback. This interactivity is guaranteed, whatever the complexity of the geometry and the texture, by using a multiresolution representation of discrete surface points extracted from the object, both for rendering and for texture evaluation.

The major advantage compared to the Pointshop3D system [174], the most commonly used system for point-based texturing, is, that we always keep a feedback to the initial geometric representation of the object. As a consequence, the final textured object can be easily exported to standard graphics software that cannot directly handle discrete surface points (e.g. *Computer Aided Design (CAD)* systems and photorealistic rendering engines).

In the following section, we present some previous work about interactive surface painting and texturing, before we briefly review the constructive texturing approach in Section 8.3. Then, in Section 8.4, we present our new interactive solid texturing technique in detail, and discuss some results in Section 8.5.

8.2 Previous Work

In the literature, several environments can be found, which can be used to interactively paint or texture the surface of an object. *Surface painting* environments are usually based on a

metaphor of the real painting approach, where one uses a symbolic pencil or a brush to paint the object, the color being applied is then evaluated according to a brush function resulting in a realistic effect [74, 4]. *Surface texturing* environments require first to specify a location and a direction on the surface, and second, to apply an existing 2D image. The latter step requires a local parameterization of the surface, which is often a non-trivial task. Furthermore, one has to think how to stitch the 2D images together without distortions. Several solutions exist, Praun et al. [127] show how to repeatedly map a small texture on a polygonal mesh using local overlapping parameterizations, and Turk [155], Wei and Levoy [162], as well as Nealen and Alexa [115] show how to synthesize textures on polygonal meshes.

In almost every existing surface painting and texturing environment, a polygonal mesh or a parametric patch is required. When the object is defined by polygons, no additional step is needed. But if it is defined by another underlying surface representation, a special preprocessing is required. For instance, Pedersen [123] proposes a general solution when the object is defined as an implicit surface, where a local parameterization is defined by estimating geodesics on the surface. Witkin and Heckbert's particle system [166] can also be used to establish a parameterization for texture mapping on implicit surfaces [173].

One solution to avoid parameterization when texturing surfaces of arbitrary objects is the use of octree textures developed recently [19, 70], where only the subsets of volume textures actually intersecting the surface are stored efficiently in an octree. However, the resulting texture is discrete and has a fixed resolution limit determined by the depth of the octree. Moreover, an octree has to be stored for every attribute, resulting in a significant storage overhead. Nevertheless, octree textures can be easily integrated in our interactive solid texturing environment thanks to the constructive texturing approach.

Another solution that does not require explicit parameterization is used in the Pointshop3D environment [174]. This innovating approach proposes to paint the surface of an object that is defined as a cloud of points. By applying a texture (2D texture, uniform color, or other), the corresponding surface points are colored. The interactive process of painting is intuitive, and good results are achieved. However, as it is totally based on discrete surface points, this approach suffers from a severe drawback: once the surface is textured, the resulting object can hardly be exported to standard graphics software that cannot handle discrete surface points (e.g. CAD systems and photorealistic rendering engines).

8.3 Constructive Texturing

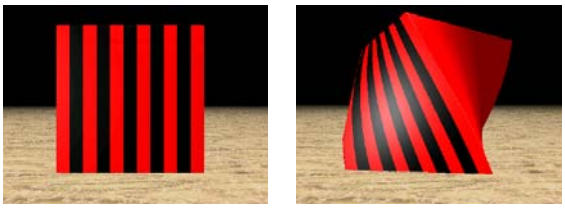
Constructive texturing is a general approach for solid texturing of objects of arbitrary type [141]. This technique consists in defining for a given object a *space partitioning* $\Xi = \bigcup_i \Xi_i$, where in each *space partition* $\Xi_i \subseteq \mathbb{R}^3$, a different set of attributes $\mathcal{A}_i$ is defined[1]. For each point of the object, one has to be able to answer the following question: "Is this point inside or outside a given space partition Ξ_i?" In the affirmative case, the attributes $\mathcal{A}_i$ corresponding to Ξ_i are applied.

One powerful solution for point membership classification is to use the FRep model [121]. In this model, each space partition Ξ_i is defined by a function f_i, and a set of attributes $\mathcal{A}_i$

[1] In this chapter, according to the involved references and in order to avoid confusions, we use the specific term space partition for a domain in $\mathbb{R}^3$.

is associated. All space partitions are organized as leaf nodes in a tree, which are combined together with operations in the interior nodes.

The main advantage of constructive texturing is its generality: it can be applied to polygonal meshes, boundary representations, parametric and implicit surfaces, voxel arrays, unorganized point sets, and others. Moreover, the texture behaves accordingly to geometry changes in animated sequences without creating distortions, like for all solid textures, as illustrated in Figures 44 and 49. On the other hand, the major drawback of the constructive texturing approach is that the creation of the space partitions is usually complex and rather painful to do at the desired locations of the surface. This is due both to the constructive approach inherent to the method, and to the lack of interactive tools. Consequently, it is difficult for a non-expert user to generate the nodes of the partition at the desired locations of the object's surface.



(a) Textured cube. (b) Twisted geometry and texture.

Figure 49: The texture follows the geometry after applying a twist operator.

8.4 A new Interactive Solid Texturing Approach

8.4.1 Overview

In the following, we present a new interactive solid texturing approach that overcomes these drawbacks. The basic idea is to let the user define the space partition by interactively selecting points on the surface.

Interactivity is guaranteed thanks to the dual nature of the point-based multiresolution representation being applied. First, as usual, the multiresolution representation is used for rendering to maintain a constant frame rate. Thanks to the adaptive and flexible multiresolution representation, areas of interest can be rendered in a higher resolution than the other areas of an object's surface without causing connectivity problems. Second, in addition to rendering, we also use the multiresolution approach in the texturing step: at low resolution, the user can paint large parts of the object, and when the resolution is increased, finer details can be painted. Moreover, the attributes can also be determined in a multiresolution manner, thus, when texturing an object's surface, visual results are obtained rapidly in a coarser resolution before refining them in a background process.

The procedure we have defined can be decomposed into 10 steps, as illustrated in Figure 50. In a preprocessing step, discrete surface points are extracted from a given object of arbitrary

type (Step 1), and a multiresolution representation of the cloud of discrete surface points is set up (Step 2). After this, the object is visualized in the adapted level-of-detail (Step 3). Then, the user can texture the object's surface (Steps 4-7), the attributes of the surface points are updated (Step 8), and visualized (back to Step 3). Finally, when the user has finished texturing, the texture can be exported (Step 9) and used during postprocessing (Step 10). Details of these steps are given in the following.

8.4.2 Preprocessing

Before texturing an object using point-based representations, one needs to define a cloud of discrete points on the object's surface (Step 1). The complexity of this preprocessing stage heavily depends on the object's type. In the case of polygonal meshes, parametric surfaces or unorganized point sets, the extraction is done directly as these representations explicitly define the boundary of the object. In the case of other representations, such as FRep, voxel arrays, or scanned data, one needs to extract the isosurfaces. A large number of techniques exist for this task, such as polygonization, particle systems [166], or ray-tracing applied to an object. Moreover, some efficient resampling techniques have been proposed in order to augment or lessen the number of extracted surface points, see for example Alexa et al. [7].

Once the cloud of discrete surface points is extracted, the multiresolution representation is set up (Step 2). This is done not only for level-of-detail rendering, but also for progressive evaluation of the constructive texturing tree. We use a hierarchy of bounding spheres stored in a binary tree in spirit of QSplat [136] which is built up in a top-down recursive manner during preprocessing as explained in Chapter 5 of Part II. In each node, we store the radius of the bounding sphere and the surface point lying closest to the barycenter of all surface points in the bounding sphere. Note that the radius of the bounding spheres at the leaf nodes is determined directly from the sampling grid used for extracting the surface points.

Defining solid texture coordinates by discrete surface point locations is prone to aliasing artefacts when high-frequency textures are used, even at the highest resolution of the hierarchy. This is also true for the surface textures natively defined in the Pointshop3D environment [174], but in contrast to Pointshop3D, discrete surface points are only used in an intermediate step to previsualize the textured surface. These aliasing artefacts will not occur during postprocessing as illustrated in Figure 51, where a high-frequency Perlin noise is applied to a tiger object defined in the FRep model.

8.4.3 Real-Time Process

In order to ensure real-time processing, the evaluation of the attributes of the surface points as well as the rendering itself has to be done in a given time. To meet this requirement, the adapted level in the multiresolution hierarchy, where the attributes of all nodes can be evaluated in the given time, is determined. This might not be the highest level of the hierarchy, as the cost of determining the attributes for a surface point increases with the complexity of the solid texture represented by the constructive texturing tree. Then, the attributes of all further nodes and surface points of the multiresolution hierarchy are determined (Step 8) in a background process as indicated by (a) in the algorithm.

In addition to choosing the adapted multiresolution level with respect to the evaluation

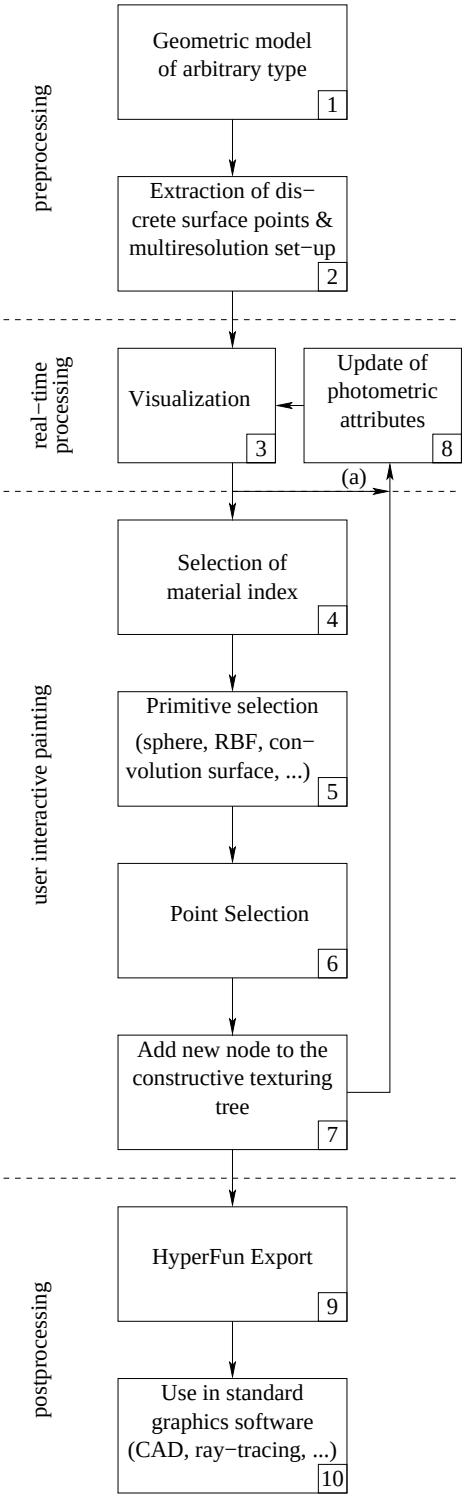
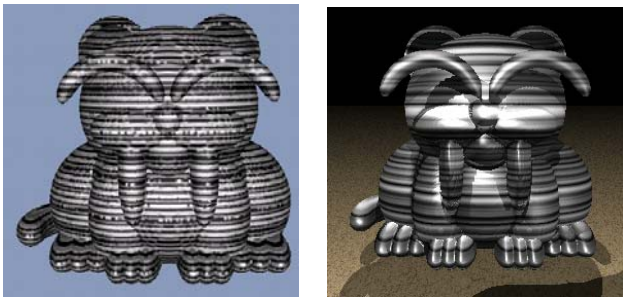


Figure 50: The different steps involved in our interactive texturing process.

cost of the attribute tree, the number of surface points that can be rendered by the graphics



(a) Previsualization.

(b) Final ray-traced image.

Figure 51: Aliasing artefacts do not occur in postprocessing.

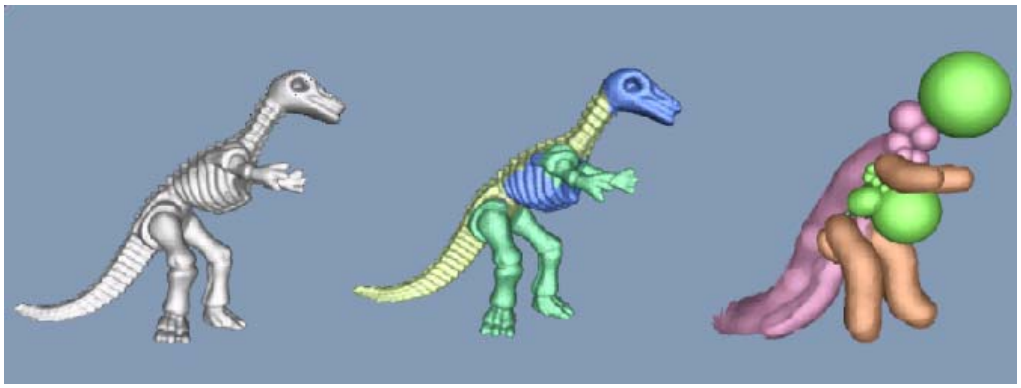


Figure 52: A set of established space partitions shown on the dinosaur statue.

hardware has also to be taken into account for efficient visualization (Step 3). This cost might be rather small when the projection of the bounding sphere of the surface point falls on a single screen pixel. Even when the projection of the bounding sphere falls on several screen pixels, the cost might stay small using some hardware-accelerated splatting technique [136], but increases when a high-quality splat is drawn [175]. However, even high-quality splats can now be drawn with hardware acceleration [130] using modern graphics hardware [101].

8.4.4 User Interactive Texturing

At this stage of the algorithm, the user textures the object's surface. This is done by choosing a material index for the texture to be used (Step 4), defining a space partition (Steps 5 and 6), and adding the space partition as a new node in the constructive texturing tree (Step 7). We will detail these steps in the following.

Selection of the material index (Step 4) First, the user selects a material index for a solid texture that will be applied in the space partition. Any kind of solid texture can be used, varying from procedurally defined solid textures, volumetric textures, simple

materials, and others. Of course, the new procedurally defined solid texture of Section 7.1 can be used as well, and we experienced that this solid texture is particularly useful at locations, where the user wants to leave the original texture of the surface unaffected.

Primitive selection (Step 5) Depending on the shape of the space partition the user wishes to obtain, different tools are available. By tool, we mean any primitive that can be defined in the FRep model. The simplest tools to define a space partition are the sphere and the block. By using *convolution surfaces* [26] for the space partition, the user can texture the object with a brush tool of a any size. With more complex primitives, the user has even more control over the space partition to define, and we integrated a tool that defines space partitions using the partition of unity variational method. This tool reveals to be extremely useful, since a significant number of points in Step 6 can be selected, and a space partition including all these points is created. For an example of various different space partitions issued by various tools, see the textured object and its corresponding space partitions of a dinosaur sculpture in Figure 52.

Point selection (Step 6) In this step, the user selects the surface points. These surface points are the parameters of the FRep primitive defining the space partition. For example, by using the partition of unity variational method tool, a space partition including all the selected surface points is determined. When the chosen tool is a brush, the selected surface points define the skeleton of the convolution surface for the space partition. When a sphere is chosen as the tool, the selected surface point defines the center, and a radius can be specified interactively.

New node in the constructive texturing tree (Step 7) Once the new space partition has been created (i.e. a new primitive), it is added to the current constructive texturing tree. This is done automatically while using the set-theoretic union operation, but any other operation available in the FRep model can be used (including other set-theoretic operations such as intersection, or blending union). In the case of overlapping partitions, set-theoretic operations need to be defined by the user. Indeed, if one considers a union operation of, for instance, two overlapping blocks, the resulting geometry is well defined. But if one considers the union of attributes, the result needs to be specified. In the case of a red and a green block, the color of the intersection of the blocks can be either red, or green, or yellow, or any other color; it corresponds to different operations on attributes, namely priority given to an attribute, a min/max function, or a user defined operation. By default, we give priority to the last added primitive.

When the user has painted on the surface by choosing the desired texture and points to define the space partition, the attributes of the surface points are updated (Step 8) and visualized (back to Step 3). Then, the user can continue to add further space partitions and to associate attributes with them. Although the space partitions can be very complex, interactivity is achieved because the evaluation of the solid texture represented by the constructive texturing tree is processed only for the discrete surface points used for visualization, at the adapted level-of-detail.

8.4.5 Postprocessing

Once the solid texture defined by the constructive texture is created, it can be saved, and exported using the extended version of HyperFun [3] (Step 9), which is a special high-level language that supports all the main notions of FRep modeling and has been recently extended to support constructive texturing. A set of plug-ins has been developed such that several existing software tools support objects described by HyperFun, such as Maya [107], POVRay [126], and other (Step 10). Export to polygonal representations, such as VRML, is also supported.

8.5 Results

8.5.1 Overview

We implemented our tool as a plugin for Pointshop3D [174] in C++ using Qt for the graphical user interfaces. A screenshot of our plugin can be seen in Figure 53, where the user is selecting the tool to define a space partition. Besides the rendering modes provided by Pointshop3D, we implemented our own rendering modes to manage the multiresolution representation. All timings given in this section were measured on a 1.7 GHz Pentium PC with 512 MB RAM.

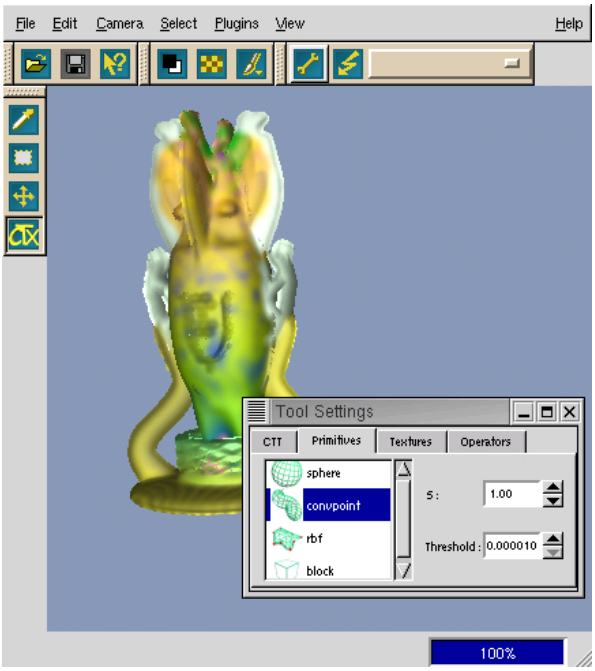
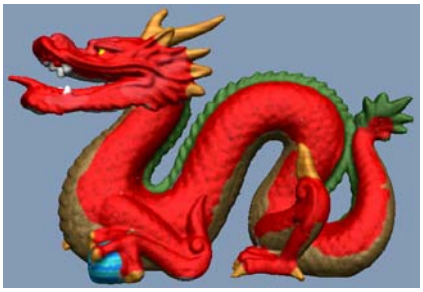


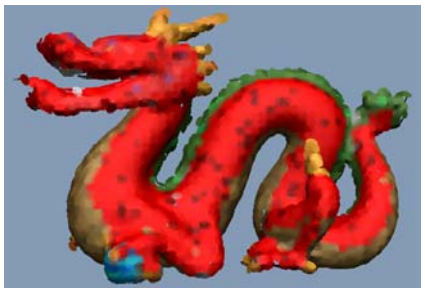
Figure 53: A screenshot of our plugin.

8.5.2 Preprocessing

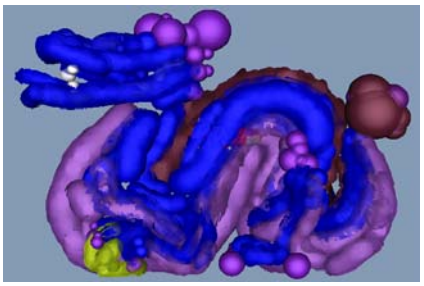
The preprocessing step is divided into two parts, the extraction of the surface points and the creation of the multiresolution representation. The time for the extraction of the surface



(a) High resolution.



(b) Very low resolution.



(c) Corresponding space partitions.

Figure 54: A more complex example using multiple space-partitions.

points is strongly related to the type and complexity of the object's geometry. Setting up the multiresolution representation is fast, it only depends on the number of extracted surface points and is in $O(n \log n)$.

Since our texturing technique can be applied to surfaces of objects from arbitrary type, we decided to take example objects with different underlying object representations. For the polygonal mesh of the Stanford Dragon, extracting the 437,645 surface points (Figure 54(b)) is done instantaneously like for the 38,619 surface points of the dinosaur statue (Figure 52) by directly using the vertices of the mesh. For the implicit surface of the 3D ant defined by an FRep model, it took 46 seconds to extract the 140,616 surface points (Figure 56(g)). Extracting the 78,499 surface points (Figure 55(a)) from the well known Siemens head sampled on a 150x200x192 voxel grid took 13 seconds.

Setting up the multiresolution representation took less than five seconds for every example shown in this chapter, for more details see the results in Chapter 5 of Part II.

8.5.3 Real-Time Process

There are two major bottlenecks which determine the interactivity of our texturing approach. First, the evaluation of the constructive texturing tree to determine the attributes of the surface points when nodes to the attribute tree were added, and second, the rendering of the surface points. Thanks to the multiresolution approach we use, maintaining interactivity for

both bottlenecks is achieved by choosing the right balance between the number of surface points which can be evaluated by traversing the constructive texturing tree and the number of surface points which can be rendered.

The time to determine the attributes of the surface points heavily depends on the complexity of the solid texture defined by the constructive texturing tree. The evaluation of the attributes of the surface points becomes critical only when very complex primitives are used. In all the examples shown in this chapter, the attributes of all surface points as well as of the interior nodes of the hierarchy used for multiresolution could be evaluated in less than a second.

If the attributes of a huge number of surface points are determined, simple hardware-accelerated splatting of the surface points suffices [136], resulting in high-quality images when the projection of the bounding sphere associated to each surface point falls only on a few screen pixels. Using our implementation, we can render up to 5M splats per second using a GeForce 3 graphics board as we already experienced in Chapter 5 in Part II.

If only a small number of surface points could be evaluated, a high-quality software splatting technique is used for point-based rendering. The adapted level-of-detail of the multiresolution representation is chosen to ensure interactive framerates. Using our implementation of the output-sensitive surface splatting technique, we can render up to 300k high-quality EWA splats per second in a 512x512 window (see also Chapter 5 of Part II). However, we envisage to use the hardware-accelerated approach of [130], which is less output-sensitive and where up to 3M surface high-quality splats per second can be rendered, or to use the even more performant hardware-accelerated approach of Botsch and Kobbelt [28]. Moreover, we strive to integrate the hardware-accelerated point-based multiresolution rendering approach of Dachsbacher et al. [46], which is perfectly suited for our approach. By shifting the computational cost to traverse the multiresolution hierarchy from the CPU to the graphics hardware, CPU load is liberated for evaluating the attributes while rendering over 50M surface points per second.

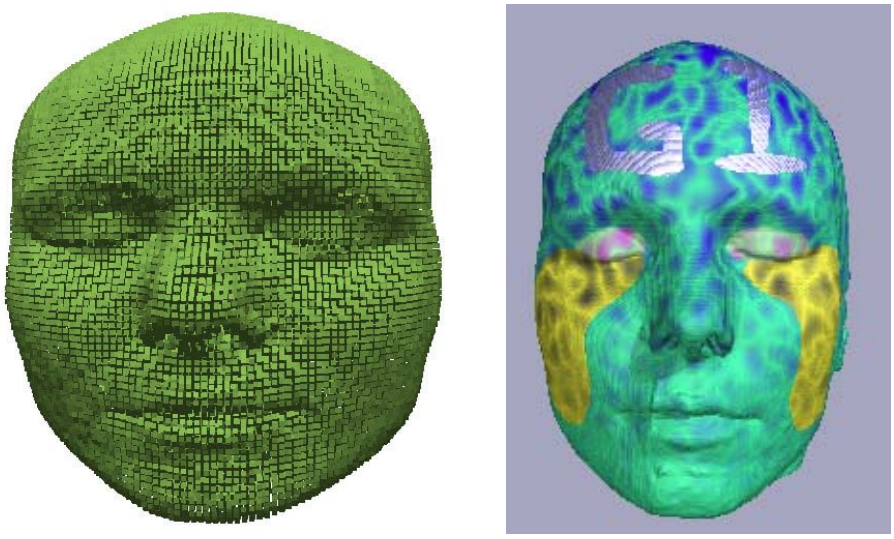
The desired quality/speed trade-off can be chosen, Figure 56 shows different levels-of-detail of our 3D ant rendered using EWA splats (Figures 56(a)-56(c)) and hardware splats (Figures 56(d)-56(f)), as well as the obtained framerates.

8.5.4 User Interactive Texturing

A more complex textured object, the Stanford Dragon described by 437,645 points, can be seen in Figure 54(a). We also show another resolution that was used during the texturing step (Figure 54(b)) as well as the corresponding space partitions that have been established (Figure 54(c)).

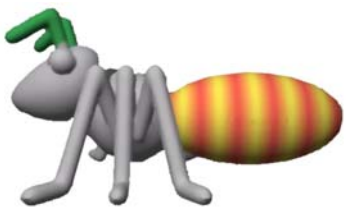
8.5.5 Postprocessing

In our environment, we used POVRay [126] to obtain photorealistic rendering. Some of our results are shown in Figure 55(b) and Figure 56(h).



(a) 78,499 extracted discrete surface points. (b) Ray-traced image.

Figure 55: Texturing a surface from the Siemens voxel array head sampled on a 150x200x192 grid.

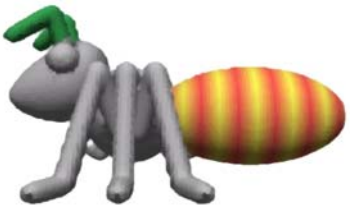


(a) 140,616 high-quality EWA splats rendered at > 1.5 fps.

(b) 34,980 high-quality EWA splats rendered at > 5 fps.

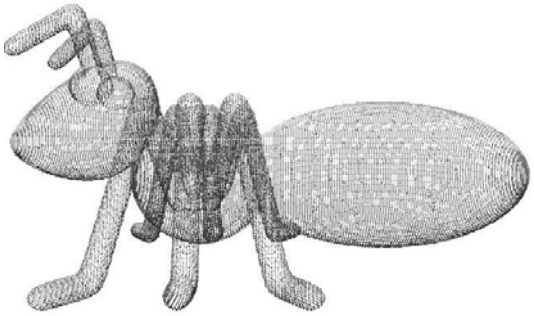
(c) 11,043 high-quality EWA splats rendered at > 15 fps.

(d) 140,616 hardware splats rendered at > 35 fps.

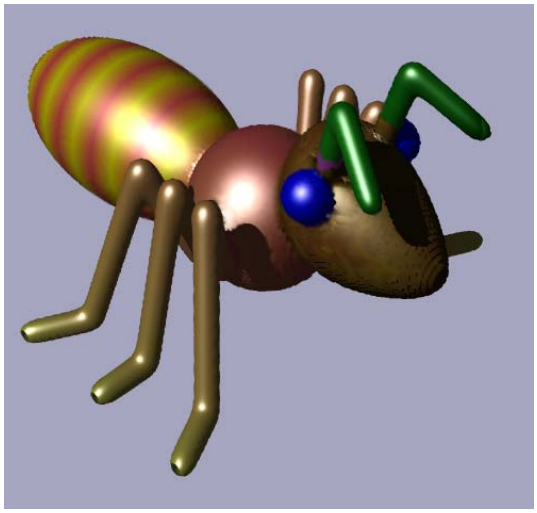


(e) 34,980 hardware splats rendered at > 140 fps.

(f) 11,043 hardware splats rendered at > 400 fps.



(g) 140,616 extracted discrete surface points from FRep model.



(h) Ray-traced image.

Figure 56: Multiresolution rendering using hardware splats and high-quality EWA splats and obtained framerates, starting from 140,616 discrete surface points extracted from an FRep model, and the final ray-traced image.

8.6 Conclusions

In this chapter, we presented a new idea that links the texture and the geometry of an object by combining two approaches previously developed in computer graphics: constructive texturing and point-based multiresolution representations. This combination allowed us to develop a software environment where 3D objects of arbitrary type can be textured by using an interactive and intuitive process. An interactive framerate is always guaranteed, whatever the complexity of the geometry and/or the texture, because the multiresolution representation of discrete surface points extracted from the object is tuned according to the performance of the graphics hardware and according to the texture complexity. This multiresolution representation offers also high-quality antialiased point-based rendering. One major advantage of our approach is that point-based rendering is only used during the interactive texturing step. We always keep a feedback to the initial geometric representation of the object (polygonal mesh, parametric or implicit surface, or whatever) which means that the final textured object can be easily exported to standard graphics software that cannot directly handle discrete surface points (e.g. CAD systems, photorealistic rendering engines).

Our implementation is still under development: currently four kinds of FRep primitives (i.e. spheres, blocks, convolution surfaces, and radial basis functions applied to a set of discrete surface points) can be used to create a partition of the constructive texturing tree. An immediate extension will be to widen the set of available FRep primitives. Two more straightforward extensions will be to integrate the hardware-accelerated high-quality splatting technique [130] that renders more than 3M points per second on current graphics hardware, and to integrate the sequential point trees technique [46], that shifts the CPU load for the multiresolution rendering to the graphics hardware.

Conclusions and Future Research Directions

Conclusions

Recent three-dimensional acquisition technologies provide a huge number of unorganized points in 3D. In this thesis, we have shown how to reconstruct implicit surfaces from such large unorganized point sets, how to render the resulting surfaces, and how these fundamentals can be used in a wide variety of applications.

More precisely, in Part I, we focused on the reconstruction of implicit surfaces from unorganized point sets and we started by presenting some previous work. We have seen that the existing methods suffer from various drawbacks. For example, some methods do not scale well with respect to the size of the point set, others are not robust against non-uniformly distributed points or noise in the point set, and others involve complex calculations or are just very hard to implement.

Consequently, we presented two new methods that overcome these problems. The new methods divide the global reconstruction domain into smaller overlapping local subdomains by using adaptive domain decomposition methods, solve the reconstruction problems in the local subdomains using radial basis functions with global support, and blend the solutions together using the partition of unity method. Whereas the first method called partition of unity variational method uses an adaptive domain decomposition method based on an octree and blends the solutions of all the leaf nodes together by forming a partition of unity, the second method called hierarchical partition of unity variational method uses an adaptive domain decomposition method based on a perfect binary tree and at all interior nodes, the solutions of the two child nodes are blended together by forming a partition of unity. The hierarchical partition of unity variational method is even more robust against non-uniformly distributed points than the partition of unity variational method, since the number of points that are in the intersection of the two overlapping subdomains can be specified explicitly. Both methods have a nice linear behavior of the required reconstruction time and memory usage with respect to the number of points in the point set since the $O(N \log N)$ creation involved in the domain decomposition method is negligible compared to the $O(N)$ reconstruction of the local subdomains.

In Part II, we focused on the rendering of implicit surfaces reconstructed from unorganized point sets. Again, we started by presenting some previous work, and we found that there is a wide variety of rendering techniques for unorganized point sets as well as for implicit surfaces. To our knowledge, the only rendering techniques using both the point set and the

reconstructed implicit surface were presented for point set surfaces [7, 6, 2] and for variational implicit surfaces reconstructed from small point sets [44]. This fact motivated us to define two new rendering techniques that use both the point set and a reconstructed implicit surface.

In the first new rendering technique, the implicit surface is rendered view-dependently in an output-sensitive multiresolution manner using points as rendering primitive. The initial unorganized point set is used to establish a multiresolution representation as a bounding sphere hierarchy, and a reconstructed implicit surface is used to generate additional points through a local ray-casting when the initial unorganized point set does not provide enough detail for rendering. By its definition, the rendering technique is faster than standard ray-tracing techniques. Nevertheless, the generation of additional points can be rather costly depending on the evaluation complexity of the defining function of the implicit surface. This is why the first new rendering technique is particularly adapted for implicit surfaces reconstructed from very large unorganized point sets with a defining function that is very fast to evaluate.

The second rendering technique based on differential point rendering [85, 86] uses a reconstructed implicit surface to pack information about local differential geometry in every point of the point set resulting in a differential point. In flat regions or regions of low curvature, a differential point approximates larger vicinities than in regions of high curvature, and thus the rectangle has a larger extent. The differential points are then rendered as non-connected fragment-shaded rectangles on programmable graphics hardware. A particularly nice feature is, that after the differential points have been created, the reconstructed implicit surface is no longer required, and a lower number of differential points suffices to represent the implicit surface. This reduces the required memory and ensures lower bus traffic between the CPU and the graphics board. However, the rendering technique is only appropriate for quasi-uniformly distributed point sets.

In Part III, we presented an incomplete list of potential applications that use the fundamentals of the first two parts of this thesis. In fact, the two new reconstruction methods presented in Part I are not limited to reconstruct the defining functions of implicit surfaces, but apply also to reconstruct continuous functions of any dimension. Consequently, we used the reconstruction methods to define a new class of procedural solid textures that can be reconstructed from the attributes of unorganized point sets. Furthermore, we applied the methods to reconstruct terrains from 2D contour lines and to repair images in 2D image processing.

Finally, we presented a new interactive environment for constructive texturing of surfaces of arbitrarily defined 3D objects (including, of course, implicit surfaces). A user can texture the surface by defining space partitions that are combined using constructive texturing, and by specifying attributes that are applied in the space partitions. The partition of unity variational method can be used to define space partitions amongst other primitives and revealed to be particularly useful. When specifying the attributes, the new class of procedural solid texture can be used as well. In order to give an interactive feedback, a point-based multiresolution representation of the surface is used that is not only exploited for rendering, but also for the evaluation of the texture.

Future Research Directions

The results of reconstruction and rendering of implicit surfaces from large unorganized point sets as well as the applications that we presented in this thesis give rise to various future research directions, and we present here some directions that seem to be particularly valuable.

The new implicit surface reconstruction methods from large unorganized point sets divide the global reconstruction domain into local subdomains that can be solved by various, non-communicating entities due to their independence. We believe that this fact enables a straightforward out-of-core implementation of the new implicit surface reconstruction methods, where only the data of the local subdomain to be reconstructed has to be kept in core thereby enabling to reconstruct implicit surfaces from hundreds of millions of unorganized points.

At the moment, the new implicit surface reconstruction methods generate one final implicit surface at highest precision. We are currently extending our reconstruction methods to generate implicit surfaces at various precisions by gradually refining lower precision implicit surfaces that are stored in the hierarchical data structure resulting from the domain decomposition method. A multi-level reconstruction method is not only interesting to rapidly gain a coarse approximation of the reconstructed implicit surface, but also valuable in bandwidth-limited network applications for a progressive transfer of the implicit surface.

In the first new point-based rendering technique, the most time-consuming process is the generation of additional points. Besides generating the additional points in parallel when several processors are available, we strive to move some operations that are involved in the local ray-casting of the implicit surface to the graphics processing unit (GPU) on programmable graphics hardware. We think that this is particularly beneficial for partition of unity blendings of lower order algebraic implicit surfaces, such as the quadratic surfaces involved in multi-level partition of unity implicits, since the ray-casting can be calculated analytically.

The differential point rendering technique for point sets using a reconstructed implicit surface is still under development. We believe that a curvature-driven sampling technique of the implicit surface in order to create the differential points further reduces the required number of differential points. Consequently, the initial point set is no longer required, and the rendering technique can be applied to implicit surfaces in general.

The new implicit surface reconstruction methods from unorganized points enable to design point-based modeling environments, where points are used as modeling primitives, as an alternative to modeling techniques based on polygonal meshes. By using our point-based rendering techniques instead of rendering the implicit surfaces by polygonal meshes, we plan to design a completely meshless interactive modeling environment by using points both as modeling and as rendering primitive.

We have shown that the new reconstruction methods can be applied to reconstruct continuous functions of different dimensions in various application domains. We intend to apply the procedural solid texture reconstructed from unorganized points of an object with a first geometry to an object with a different geometry by defining a shape transformation $\mathbb{R}^3 \rightarrow \mathbb{R}^3$ from the first to the second geometry, and by applying the shape transformation to the procedural solid texture. Moreover, we want to show the power of our new reconstruction methods in further applications, such as repairing 3D polygonal meshes and the reconstruction of missing information in medical data sets.

Appendix A

Differentials

3D RBF reconstruction

Biharmonic basic functions

The radial basis function reconstruction by using biharmonic basic funtions $\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|$ associated with a linear polynomial in vector notation is:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} w_i \|\boldsymbol{x} - \boldsymbol{p}_i)\| + \sum_{\alpha=1}^{4} \pi_\alpha p_\alpha(\boldsymbol{x}),$$

By setting $\boldsymbol{x} = [x, y, z]^T$ and $\boldsymbol{p}_i = [x_i, y_i, z_i]^T$, the radial basis function is:

$$f(x, y, z) = \sum_{i=1}^{N} \omega_i \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} + \pi_1 x + \pi_2 y + \pi_3 z + \pi_4$$

The partial first and second derivatives are the following:

$$\begin{aligned}
\frac{\partial f}{\partial x}(x, y, z) &= \sum_{i=1}^{N} \omega_i \frac{x - x_i}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}} + \pi_1 \\
\frac{\partial f}{\partial y}(x, y, z) &= \sum_{i=1}^{N} \omega_i \frac{y - y_i}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}} + \pi_2 \\
\frac{\partial f}{\partial z}(x, y, z) &= \sum_{i=1}^{N} \omega_i \frac{z - z_i}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}} + \pi_3
\end{aligned}$$

$$\begin{aligned}
\frac{\partial f}{\partial x^2}(x, y, z) &= \sum_{i=1}^{N} \frac{\omega_i}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}} \left(1 - \frac{(x-x_i)^2}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}^3}\right) \\
\frac{\partial f}{\partial y^2}(x, y, z) &= \sum_{i=1}^{N} \frac{\omega_i}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}} \left(1 - \frac{(y-y_i)^2}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}^3}\right) \\
\frac{\partial f}{\partial z^2}(x, y, z) &= \sum_{i=1}^{N} \frac{\omega_i}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}} \left(1 - \frac{(z-z_i)^2}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}^3}\right) \\
\frac{\partial f}{\partial x \partial y}(x, y, z) &= \frac{\partial f}{\partial y \partial x}(x, y, z) = -\sum_{i=1}^{N} \frac{\omega_i (x-x_i)(y-y_i)}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}^3} \\
\frac{\partial f}{\partial x \partial z}(x, y, z) &= \frac{\partial f}{\partial z \partial x}(x, y, z) = -\sum_{i=1}^{N} \frac{\omega_i (x-x_i)(z-z_i)}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}^3} \\
\frac{\partial f}{\partial y \partial z}(x, y, z) &= \frac{\partial f}{\partial z \partial y}(x, y, z) = -\sum_{i=1}^{N} \frac{\omega_i (y-y_i)(z-z_i)}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}^3}
\end{aligned}$$

Triharmonic basic functions

The radial basis function reconstruction by using triharmonic basic functions $\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|^3$ with associated with a quadratic polynomial in vector notation is:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} w_i \|(\boldsymbol{x} - \boldsymbol{p}_i)\|^3 + \sum_{\alpha=1}^{10} \pi_\alpha p_\alpha(\boldsymbol{x}), \tag{85}$$

By setting $\boldsymbol{x} = [x, y, z]^T$ and $\boldsymbol{p}_i = [x_i, y_i, z_i]^T$, the radial basis function is:

$$\begin{aligned} f(x,y,z) &= \textstyle\sum_{i=1}^{N} \omega_i \sqrt{(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2}^3 \\ &\quad +\pi_1 x^2 + \pi_2 y^2 + \pi_3 z^2 + \pi_4 xy + \pi_5 yz + \pi_6 xz + \pi_7 x + \pi_8 y + \pi_9 z + \pi_{10} \end{aligned}$$

The partial first and second derivatives are the following:

$$\begin{aligned} \tfrac{\partial f}{\partial x}(x,y,z) &= 3\textstyle\sum_{i=1}^{N} \omega_i (x-x_i)\sqrt{(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2} \\ &\quad +2\pi_1 x + \pi_4 y + \pi_6 z + \pi_7 \\ \tfrac{\partial f}{\partial y}(x,y,z) &= 3\textstyle\sum_{i=1}^{N} \omega_i (y-y_i)\sqrt{(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2} \\ &\quad +2\pi_2 y + \pi_4 x + \pi_5 z + \pi_8 \\ \tfrac{\partial f}{\partial z}(x,y,z) &= 3\textstyle\sum_{i=1}^{N} \omega_i (z-z_i)\sqrt{(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2} \\ &\quad +2\pi_3 z + \pi_5 y + \pi_6 x + \pi_9 \end{aligned}$$

$$\begin{aligned} \tfrac{\partial f}{\partial x^2}(x,y,z) &= 3\textstyle\sum_{i=1}^{N} \left(\omega_i \left(\frac{(x-x_i)^2}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}} + \sqrt{(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2}\right)\right) \\ &\quad +2\pi_1 \\ \tfrac{\partial f}{\partial y^2}(x,y,z) &= 3\textstyle\sum_{i=1}^{N} \left(\omega_i \left(\frac{(y-y_i)^2}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}} + \sqrt{(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2}\right)\right) \\ &\quad +2\pi_2 \\ \tfrac{\partial f}{\partial z^2}(x,y,z) &= 3\textstyle\sum_{i=1}^{N} \left(\omega_i \left(\frac{(z-z_i)^2}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}} + \sqrt{(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2}\right)\right) \\ &\quad +2\pi_3 \\ \tfrac{\partial f}{\partial x \partial y}(x,y,z) &= \tfrac{\partial f}{\partial y \partial x}(x,y,z) = 3\textstyle\sum_{i=1}^{N} \omega_i \frac{(x-x_i)(y-y_i)}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}} + \pi_4 \\ \tfrac{\partial f}{\partial x \partial z}(x,y,z) &= \tfrac{\partial f}{\partial z \partial x}(x,y,z) = 3\textstyle\sum_{i=1}^{N} \omega_i \frac{(x-x_i)(z-z_i)}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}} + \pi_6 \\ \tfrac{\partial f}{\partial y \partial z}(x,y,z) &= \tfrac{\partial f}{\partial z \partial y}(x,y,z) = 3\textstyle\sum_{i=1}^{N} \omega_i \frac{(y-y_i)(z-z_i)}{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}} + \pi_5 \end{aligned}$$

3D Partition of Unity blending

The partition of unity blending of M local reconstruction functions f_i with the weighting functions $\hat{w}_i$ is given by:

$$f(\boldsymbol{x}) = \frac{\sum_{i=1}^{M} f_i(\boldsymbol{x})\hat{w}_i(\boldsymbol{x})}{\sum_{i=1}^{M} \hat{w}_i(\boldsymbol{x})}.$$

The first and second derivatives are the following:

$$f'(\boldsymbol{x}) = \frac{\sum_{i=1}^{M} \hat{w}_i(\boldsymbol{x}) \left(\sum_{i=1}^{M} {f'}_i(\boldsymbol{x})\hat{w}_i(\boldsymbol{x}) + \sum_{i=1}^{M} f_i(\boldsymbol{x})\hat{w}'_i(\boldsymbol{x})\right) - \sum_{i=1}^{M} \hat{w}'_i(\boldsymbol{x}) \left(\sum_{i=1}^{M} f_i(\boldsymbol{x})\hat{w}_i(\boldsymbol{x})\right)}{\left(\sum_{i=1}^{M} \hat{w}_i(\boldsymbol{x})\right)^2}$$

$$\begin{aligned} f''(\boldsymbol{x}) \quad = \quad & \frac{\sum_{i=1}^{M} [2{f'}_i(\boldsymbol{x})\hat{w}'_i(\boldsymbol{x}) + {f''}_i(\boldsymbol{x})\hat{w}_i(\boldsymbol{x}) + f_i(\boldsymbol{x})\hat{w}''_i(\boldsymbol{x})]}{\sum_{i=1}^{M} \hat{w}_i(\boldsymbol{x})} \\ & - \frac{2\sum_{i=1}^{M} \hat{w}'_i(\boldsymbol{x}) \sum_{i=1}^{M} [{f'}_i(\boldsymbol{x})\hat{w}_i(\boldsymbol{x}) + f_i(\boldsymbol{x})\hat{w}'_i(\boldsymbol{x})] + \sum_{i=1}^{M} \hat{w}''_i \sum_{i=1}^{M} f_i(\boldsymbol{x})\hat{w}_i(\boldsymbol{x}))}{(\sum_{i=1}^{M} \hat{w}_i(\boldsymbol{x}))^2} \\ & + \frac{2\sum_{i=1}^{M} \hat{w}'_i(\boldsymbol{x})^2 \sum_{i=1}^{M} f_i(\boldsymbol{x})\hat{w}_i(\boldsymbol{x})}{(\sum_{i=1}^{M} \hat{w}_i(\boldsymbol{x}))^3} \end{aligned}$$

with

$$\begin{aligned} \hat{w}_i(\boldsymbol{x}) &= v(d_i(\boldsymbol{x})) \\ \hat{w}'_i(\boldsymbol{x}) &= v'(d_i(\boldsymbol{x}))d'_i(\boldsymbol{x}) \\ \hat{w}''_i(\boldsymbol{x}) &= v''(d_i(\boldsymbol{x}))d'_i(\boldsymbol{x})^2 + v'(d_i(\boldsymbol{x}))d''_i(\boldsymbol{x}) \end{aligned}$$

The derivatives of several decay functions v are as follows:

Continuity	v	v'	v''
C^0	$v_0(x) = 1 - x$	$v'_0(x) = -1$	$v''_0(x) = 0$
C^1	$v_1(x) = 2x^3 - 3x^2 + 1$	$v'_1(x) = 6x^2 - 6x$	$v''_1(x) = 12x - 6$
C^1	$v_2(x) = (x^2 - 1)^2$	$v'_2(x) = 4x^3 - 4x$	$v''_2(x) = 12x^2 - 4$
C^2	$v_3(x) = -6x^5 + 15x^4 - 10x^3 + 1$	$v'_3(x) = -30x^4 + 60x^3 - 30x^2$	$v''_3(x) = -120x^3 + 180x^2 - 60x$

When using the cubical distance function for a cube spanned between $\boldsymbol{a} = [x_a, y_a, z_a]^T$ and $\boldsymbol{b} = [x_b, y_b, z_b]^T$ with side length u

$$d_i^{cube}(x, y, z) \quad = \quad 1 - \frac{64}{u^6} \prod_{v \in x,y,z} (v - v_a)(v_b - v) \frac{\|\boldsymbol{p}, \boldsymbol{c}\|}{r},$$

the partial first and second derivatives are as follows:

$$\begin{aligned}
\frac{\partial d_i^{cube}}{\partial x}(x,y,z) &= -\frac{64}{u^6}\Big([(x_b-x)-(x-x_a)]\,[(y-y_a)(y_b-y)(z-z_a)(z_b-z)]\Big)\\
\frac{\partial d_i^{cube}}{\partial x}(x,y,z) &= -\frac{64}{u^6}\Big([(y_b-y)-(y-y_a)]\,[(x-x_a)(x_b-x)(z-z_a)(z_b-z)]\Big)\\
\frac{\partial d_i^{cube}}{\partial x}(x,y,z) &= -\frac{64}{u^6}\Big([(z_b-z)-(z-z_a)]\,[(x-x_a)(x_b-x)(y-y_a)(y_b-y)]\Big)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial d_i^{cube}}{\partial x^2}(x,y,z) &= -\frac{128}{u^6}(y-y_a)(y_b-y)(z-z_a)(z_b-z)\\
\frac{\partial d_i^{cube}}{\partial y^2}(x,y,z) &= -\frac{128}{u^6}(x-x_a)(x_b-x)(z-z_a)(z_b-z)\\
\frac{\partial d_i^{cube}}{\partial z^2}(x,y,z) &= -\frac{128}{u^6}(x-x_a)(x_b-x)(y-y_a)(y_b-y)\\
\frac{\partial d_i^{cube}}{\partial x\partial y}(x,y,z) &= \frac{\partial d_i^{cube}}{\partial y\partial x}(x,y,z) = \frac{64}{u^6}(z-z_a)(z_b-z)([(y_b-y)-(y-y_a)][(x_b-x)-(x-x_a)])\\
\frac{\partial d_i^{cube}}{\partial x\partial z}(x,y,z) &= \frac{\partial d_i^{cube}}{\partial z\partial x}(x,y,z) = \frac{64}{u^6}(y-y_a)(y_b-y)([(z_b-z)-(z-z_a)][(x_b-x)-(x-x_a)])\\
\frac{\partial d_i^{cube}}{\partial y\partial z}(x,y,z) &= \frac{\partial d_i^{cube}}{\partial z\partial y}(x,y,z) = \frac{64}{u^6}(x-x_a)(x_b-x)([(z_b-z)-(z-z_a)][(y_b-y)-(y-y_a)])
\end{aligned}$$

When using the spherical distance function with center $\boldsymbol{c} = [x_c, y_c, z_c]^T$ and radius r

$$d_i^{sphere}(\boldsymbol{x}) = \frac{\|\boldsymbol{x},\boldsymbol{c}\|}{r},$$

the partial first and second derivatives are as follows:

$$\begin{aligned}
\frac{\partial d_i^{sphere}}{\partial x}(x,y,z) &= \frac{1}{r\sqrt{(x-x_c)^2+(y-y_c)^2+(z-z_c)^2}}(x-x_c)\\
\frac{\partial d_i^{sphere}}{\partial y}(x,y,z) &= \frac{1}{r\sqrt{(x-x_c)^2+(y-y_c)^2+(z-z_c)^2}}(y-y_c)\\
\frac{\partial d_i^{sphere}}{\partial z}(x,y,z) &= \frac{1}{r\sqrt{(x-x_c)^2+(y-y_c)^2+(z-z_c)^2}}(z-z_c)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial d_i^{sphere}}{\partial x^2}(x,y,z) &= \frac{1}{r\sqrt{(x-x_c)^2+(y-y_c)^2+(z-z_c)^2}} - \frac{(x-x_c)^2}{r(\sqrt{(x-x_c)^2+(y-y_c)^2+(z-z_c)^2})^3}\\
\frac{\partial d_i^{sphere}}{\partial y^2}(x,y,z) &= \frac{1}{r\sqrt{(x-x_c)^2+(y-y_c)^2+(z-z_c)^2}} - \frac{(y-y_c)^2}{r(\sqrt{(x-x_c)^2+(y-y_c)^2+(z-z_c)^2})^3}\\
\frac{\partial d_i^{sphere}}{\partial z^2}(x,y,z) &= \frac{1}{r\sqrt{(x-x_c)^2+(y-y_c)^2+(z-z_c)^2}} - \frac{(z-z_c)^2}{r(\sqrt{(x-x_c)^2+(y-y_c)^2+(z-z_c)^2})^3}\\
\frac{\partial d_i^{sphere}}{\partial x\partial y}(x,y,z) &= \frac{\partial d_i^{sphere}}{\partial y\partial x}(x,y,z) = \frac{(x-x_c)(y-y_c)}{r(\sqrt{(x-x_c)^2+(y-y_c)^2+(z-z_c)^2})^3}\\
\frac{\partial d_i^{sphere}}{\partial x\partial z}(x,y,z) &= \frac{\partial d_i^{sphere}}{\partial z\partial x}(x,y,z) = \frac{(x-x_c)(z-z_c)}{r(\sqrt{(x-x_c)^2+(y-y_c)^2+(z-z_c)^2})^3}\\
\frac{\partial d_i^{sphere}}{\partial y\partial z}(x,y,z) &= \frac{\partial d_i^{sphere}}{\partial z\partial y}(x,y,z) = \frac{(y-y_c)(z-z_c)}{r(\sqrt{(x-x_c)^2+(y-y_c)^2+(z-z_c)^2})^3}
\end{aligned}$$

Appendix B

Reference of Symbols

Symbols

$\bullet$	dot product
$\times$	cross product
$\circ$	chain rule
$\tilde{f}$	Fourier transform of f
(i, j)	edge between i and j
$[x, y, z]^T$	three-dimensional vector
$\|\boldsymbol{x}\|$	Euclidean norm of a vector $\sqrt{\sum_{i=1}^{d} x_i^2}$ with $\boldsymbol{x} = [x_1, x_2, \ldots, x_d]^T$
∇	gradient
$\rightarrow$	space mapping
$\Rightarrow$	follows

Math spaces

$\mathbb{N}$	natural numbers
$\P$	polynomial space
$\mathbb{R}$	real numbers

Domains (capital greek letters)

Ξ_i	$\subseteq \mathbb{R}^3$	space partition
Ω	$\subseteq \mathbb{R}^d$	global domain
Ω_i	$\subseteq \mathbb{R}^d$	local subdomain
$\Omega_i^{[l]}$	$\subseteq \mathbb{R}^d$	local subdomain at level l

Sets (rounded capital roman letters)

$\mathcal{A}$	set of attributes
$\mathcal{D}$	overlapping domains
$\mathcal{N}_i^{\{H\}}$	H-neighborhood of i
$\mathcal{P}$	point set
$\mathcal{P}^{[l]}$	point set at level l
$\mathcal{P}_i$	point Set of a local subdomain
$\mathcal{S}$	surface
$\mathcal{W}$	set of weighting functions

Vectors (bold small roman letters)

$\boldsymbol{a}$	lower extrema of box
$\boldsymbol{b}$	upper extrema of box
$\boldsymbol{c}$	center of sphere/cube
$\boldsymbol{c}$	center pixel
$\boldsymbol{d}$	ray direction
$\boldsymbol{n}$	normal
$\boldsymbol{n}_i$	normal in point $\boldsymbol{p}_i$
$\boldsymbol{o}_i$	centroid
$\boldsymbol{o}$	ray origin
$\boldsymbol{p}_i$	points for reconstruction
$\boldsymbol{p}_i^{[k]}$	points at level k
$\boldsymbol{p}^{near}$	point near an MLS surface
$\boldsymbol{p}_j^{add}$	additionally generated point
$\boldsymbol{p}_{is}$	ray-surface intersection point
$\boldsymbol{q}$	projected MLS point
$\boldsymbol{q}_i$	projected MLS point
$\boldsymbol{q}^{near}$	intersection point with $\mathsf{T}^{\mathsf{near}}$
$\boldsymbol{q}^{far}$	intersection point with $\mathsf{T}^{\mathsf{far}}$
$\boldsymbol{s}_i$	skeleton point for blobby objects
$\boldsymbol{u}_{\boldsymbol{p}}$	principal direction in $\boldsymbol{p}$
$\boldsymbol{v}_{\boldsymbol{p}}$	principal direction in $\boldsymbol{p}$
$\boldsymbol{w}$	small vector
$\boldsymbol{x}$	point in space

Matrices (capital bold roman letters)

$\boldsymbol{A}$	for Ax=b
$\boldsymbol{C}$	covariance matrix
$\boldsymbol{H}$	Hessian matrix
$\boldsymbol{I}$	identity matrix
$\boldsymbol{P}$	polynomial matrix
$\boldsymbol{R}$	rotation matrix

Univariate scalar Functions $\mathbb{R} \to \mathbb{R}$ (small greek letters)

θ	$\mathbb{R} \to \mathbb{R}$	weight function
υ	$\mathbb{R} \to \mathbb{R}$	decay function
ϕ	$\mathbb{R} \to \mathbb{R}$	basic function

Multivariate scalar Functions $\mathbb{R}^n \to \mathbb{R}$ (small roman letters)

d	$\mathbb{R}^3 \to \mathbb{R}$	distance function
f	$\mathbb{R}^3 \to \mathbb{R}$	defining function
f_i	$\mathbb{R}^3 \to \mathbb{R}$	local defining function
$f^{[l]}$	$\mathbb{R}^3 \to \mathbb{R}$	defining function at level l
f_c	$\mathbb{R}^3 \to \mathbb{R}$	carrier solid function
f_a	$\mathbb{R}^3 \to \mathbb{R}$	attribute reconstruction function
g	$\mathbb{R}^3 \to \mathbb{R}$	quadric function
g	$\mathbb{R}^2 \to \mathbb{R}$	polynomial MLS approximation
$o^{[l]}$	$\mathbb{R}^3 \to \mathbb{R}$	offset function at level l
p	$\mathbb{R}^3 \to \mathbb{R}$	polynomial basis function
q_i	$\mathbb{R}^3 \to \mathbb{R}$	potential function
u	$\mathbb{R}^3 \to \mathbb{R}$	volume spline function
w	$\mathbb{R}^3 \to \mathbb{R}$	normalized weighting function
$\hat{w}$	$\mathbb{R}^3 \to \mathbb{R}$	weighting function

Univariate Vectorial Functions $\mathbb{R} \to \mathbb{R}^m$ (capital roman letters)

R	$\mathbb{R} \to \mathbb{R}^3$	ray

Multivariate Vectorial Functions $\mathbb{R}^n \to \mathbb{R}^m$ (capital greek letters)

Π	$\mathbb{R}^2 \to \mathbb{R}^3$	parameterization function
Ψ	$\mathbb{R}^3 \to \mathbb{R}^3$	projection operator function

Functionals $f \to \mathbb{R}$ (capital bold roman letters)

E	energy functional
V	variational functional

Bounds (capital roman letters)

C	number of points in the cylinder
K	number of off-surface points
H	number of points in the H-neighborhood
L	number of levels
L	Lipschitz bound
M	number of domains
N	number of points
Q	dimension of the polynomial
T_{grid}	point bound
T_{min}, T_{max}	point bounds
T_{min}	point bound

Constants (small greek letters)

α	degree of derivative
β	degree of overlapping
γ	increasing overlapping
δ	for gaussian function
ϵ	precision
κ_i	for placing normal constraints
λ	regularization parameter
μ	for computational complexity
ν	for computational complexity
π	coefficient for polynomials
ρ	grid size step for polygonization
σ	support radius
τ	threshold
χ	maximum principal width
ω	coefficient for RBF

Variables (small roman letters)

a_i, b_i	parameters for blobby spheres
a_i	material attribute
c	for morphing
d	for dimensions
d_j	distance from the point
d	offset for hessian normal form
d_c	diagonal of a cubic octant cell
d_Ω	diagonal of the domain Ω
h	feature size
h_i	for interpolation constraints
$h_\mathcal{P}$	fill distance of the point set
i, j, k, l, m	running variables
i_1, i_2	for overlapping zone
m	order
$q_\mathcal{P}$	separation distance of the point set
r	radius
r_i	values of the carrier function
r_i, g_i, b_i	material attributes
t	parameter for ray R
t_{rec}, t_{total}, t_M, $t_{hierarchy}$	timings
u	side length of a cube
$u_{\boldsymbol{p}}$, $v_{\boldsymbol{p}}$	principal curvatures in $\boldsymbol{p}$
v	coordinate $v \in x, y, z$.
x, y, z	coordinates

Names (capital sans serif letters)

B	blobby primitive
C	cylinder
H	local MLS reference plane
T^{near}, T^{far}	planes for additional point generation

Bibliography

[1] Anders Adamson and Marc Alexa. Approximating and intersecting surfaces from points. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing 2003*, pages 230–239, 2003.

[2] Anders Adamson and Marc Alexa. Ray tracing point set surfaces. In *Proceedings of Shape Modeling International 2003*, pages 272–282, 2003.

[3] Valery Adzhiev, Richard Cartwright, Eric Fausett, Anatoli Ossipov, Alexander Pasko, and Vladimir V. Savchenko. Hyperfun project: A framework for collaborative multidimensional F-rep modeling. In *Proceedings of the Implicit Surfaces '99*, pages 59–69, 1999.

[4] Maneesh Agrawala, Andrew C. Beers, and Marc Levoy. 3D painting on scanned surfaces. In *Proceedings of the ACM Symposium on Interactive 3D Graphics 1995*, pages 145–150, 1995.

[5] Samir Akkouche and Eric Galin. Adaptive implicit surface polygonization using marching triangles. *Computer Graphics Forum*, 20(2):67–80, 2001.

[6] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, 2003.

[7] Marc Alexa, Johannes Behr, Daniel Cohen-Or, David Levin, Shachar Fleishman, and Claudio T. Silva. Point set surfaces. In *IEEE Visualization 2001*, pages 21–28, 2001.

[8] Eugene L. Allgower and Stefan Gnutzmann. Simplicial pivoting for mesh generation of implicitly defined surfaces. *Computer Aided Geometric Design*, 8(4):305 – 325, 1991.

[9] Applied Research Associates Ltd. 2003. `http://www.aranz.com`.

[10] Nicolas Aspert, Diego Santa-Cruz, and Touradj Ebrahimi. Mesh: Measuring errors between surfaces using the hausdorff distance. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, volume I, pages 705 – 708, 2002. `http://mesh.epfl.ch`.

[11] Ulf Assarsson and Tomas Möller. Optimized view frustum culling algorithms. Technical Report 99-3, Chalmers University of Technology, 1999.

[12] Ivo M. Babuska and Jens M. Melenk. The partition of unity finite method. *International Journal for Numerical Methods in Engineering*, 40:727–758, 1997.

[13] Richard K. Beatson, Jon B. Cherrie, and Cameron T. Mouat. Fast fitting of radial basis functions: Methods based on preconditioned GMRES iteration. *Advances in Computational Mathematics*, pages 253–270, 1999.

[14] Richard K. Beatson and Leslie Greengard. A short course on fast multipole methods. In M. Ainsworth, J. Levesley, W.A. Light, and M. Marletta, editors, *Wavelets, Multilevel Methods and Elliptic PDEs*, pages 1–37. Oxford University Press, 1997.

[15] Richard K. Beatson and Will A. Light. Fast evaluation of radial basis functions: methods for two-dimensional polyharmonic splines. *IMA Journal of Numerical Analysis*, 17(3):343–372, 1997.

[16] Richard K. Beatson, Will A. Light, and Stephen Billings. Fast solution of the radial basis function interpolation equations: Domain decomposition methods. *SIAM J. Sci. Comput.*, 5(22):1717–1740, 2000.

[17] Richard K. Beatson and Garry N. Newsam. Fast evaluation of radial basis functions. *Computational Mathematics and Applications*, 24(12):7–20, 1992.

[18] Dominique Bechmann. Space deformation models survey. *Computers & Graphics*, 18(4):571–586, 1994.

[19] David Benson and Joel Davis. Octree textures. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3):785–790, 2002.

[20] Eric Bittar, Nicolas Tsingos, and Marie-Paule Gascuel. Automatic reconstruction of unstructured 3D data: Combining medial axis and implicit surfaces. *Computer Graphics Forum (Eurographics '95)*, 14(3):457–468, 1995.

[21] James F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982.

[22] James F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. *Computer Graphics (Proceedings of ACM SIGGRAPH 82)*, 16(3):21–29, 1982.

[23] Jules Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5(4):341–355, 1988.

[24] Jules Bloomenthal. An implicit surface polygonizer. *Graphics Gems IV*, pages 324–349, 1994.

[25] Jules Bloomenthal. *Introduction to implicit surfaces*, chapter Surface Tiling. Morgan Kaufmann, 1997.

[26] Jules Bloomenthal and Ken Shoemake. Convolution surfaces. *Computer Graphics*, 25(4):251–256, 1991.

[27] Jean-Daniel Boissonnat, Olivier Devillers, and Monique Teillaud. A dynamic construction of higher-order Voronoi diagrams and its randomized analysis. Rapport de recherche 1207, INRIA, 1990.

[28] Mario Botsch and Leif Kobbelt. High-quality point-based rendering on modern GPUs. In *Proceedings of Pacific Graphics 2003*, pages 335–343, 2003.

[29] Mario Botsch, Andreas Wiratanaya, and Leif Kobbelt. Efficient high quality rendering of point sampled geometry. In *Rendering Techniques 2002 (Proceedings of the Eurographics Workshop on Rendering 2002)*, pages 53–64, 2002.

[30] Andrea Bottino, Wim Nuij, and Kees van Overveld. How to shrinkwrap through a critical point: an algorithm for the adaptive triangulation of iso-surfaces with arbitrary topology. In *Proceedings of Implicit Surfaces '96*, pages 53–73, 1996.

[31] Manfredo P. Do Carmo. *Differential Geometry of curves and surfaces.* Prentice-Hall, 1976.

[32] Loren Carpenter. The a-buffer, an antialiased hidden surface method. *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, 18(3):103–108, 1984.

[33] Jonathan C. Carr, Richard K. Beatson, Jon B. Cherrie, Tim J. Mitchell, W. Richard Fright, Bruce C. McCallum, and Tim R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In Eugene Fiume, editor, *Proceedings of ACM SIGGRAPH 2001*, pages 67–76, 2001.

[34] Jonathan C. Carr, Richard K. Beatson, Bruce C. McCallum, W. Richard Fright, Tim J. McLennan, and Tim J. Mitchell. Smooth surface reconstruction from noisy range data. In *Proceedings of ACM Graphite 2003*, pages 119–126, 2003.

[35] Edwin E. Catmull. *A subdivision algorithm for computer display of curved surfaces.* PhD thesis, University of Utah, 1974. Report UTECCSc -74-133.

[36] Chun-Fa Chang, Gary Bishop, and Anselmo Lastra. Ldi tree: A hierarchical representation for image-based rendering. In *Proceedings of ACM SIGGRAPH 99*, pages 291–298, 1999.

[37] Baoquan Chen and Minh Xuan Nguyen. Pop: a hybrid point and polygon rendering system for large data. In *IEEE Visualization 2001*, pages 45–52, 2001.

[38] Paolo Cignoni, C. Rocchini, and Roberto Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

[39] James H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19:547–554, 1976.

[40] Liviu Coconu and Hans-Christian Hege. Hardware-accelerated point-based rendering of complex scenes. In *Rendering Techniques 2002 (Proceedings of the Eurographics Workshop on Rendering 2002)*, pages 43–52, 2002.

[41] Jonathan D. Cohen, Daniel G. Aliaga, and Weiqiang Zhang. Hybrid simplification: Combining multi-resolution polygon and point rendering. In *IEEE Visualization 2001*, pages 37–44, 2001.

[42] Wagner T. Corrêa, Shachar Fleishman, and Cláudio T. Silva. Towards point-based acquisition and rendering of large real-world environments. In *Proceedings of SIBGRAPI 2002*, 2002.

[43] Benoit Crespin, Pascal Guitton, and Christophe Schlick. Efficient and accurate tessellation of implicit sweep objects. In *Proceedings of Constructive Solid Geometry '98*, 1998.

[44] Benoît Crespin. Dynamic triangulation of variational implicit surfaces using incremental delaunay tetrahedralization. In *Proceedings of the IEEE Symposium on Volume visualization and graphics 2002*, pages 73–80, 2002.

[45] Charles Csuri, Ron Hackathorn, Richard Parent, Wayne E. Carlson, and Marc Howard. Towards an interactive high visual complexity animation system. *Computer Graphics (Proceedings of ACM SIGGRAPH 79)*, 13(3):289–299, 1979.

[46] Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. Sequential point trees. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)*, 22(3):657 – 662, 2003.

[47] Luiz Henrique de Figueiredo, Jonas Gomes, Demetri Terzopoulos, and Luiz Velho. Physically-based methods for polygonization of implicit surfaces. In *Proceedings of Graphics Interface '92*, pages 250–257. CIPS, 1992.

[48] Boris N. Delaunay. Sur la sphère vide. *Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.

[49] Mathieu Desbrun, Nicolas Tsingos, and Marie-Paule Cani. Adaptive sampling of implicit surfaces for interactive modeling and animation. *Computer Graphics Forum*, 15(5), 1996. Published under the name Marie-Paule Gascuel.

[50] Oliver Deussen, Carsten Colditz, Marc Stamminger, and George Drettakis. Interactive visualization of complex plant ecosystems. In *Proceedings of the IEEE Visualization 2002*. IEEE, 2002.

[51] Tamal K. Dey and James Hudson. PMR: Point to mesh rendering, a feature-based approach. In *IEEE Visualization 2002*, 2002.

[52] Huong Quynh Dinh, Greg Slabaugh, and Greg Turk. Reconstructing surfaces using anisotropic basis functions. In *Proceedings of International Conference on Computer Vision (ICCV) 2001*, pages 606–613, 2001.

[53] J. Duchon. Spline minimizing rotation-invariant semi-norms in Sobolev spaces. In W. Schempp and K. Zeller, editors, *Constructive Theory of Functions of Several Variables*, volume 571, pages 85–100, 1977.

[54] Martin J. Dürst. Letters: Additional reference to "marching cubes". *Computer Graphics*, 22(2):72–73, 1988.

[55] Far Field Technology. *FastRBF Toolbox User Manual.* 2003. www.farfieldtechnology.com.

[56] Richard Franke. Scattered data interpolation: Tests of some methods. *Mathematics of Computation*, 38(157):181–200, 1982.

[57] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of ACM SIGGRAPH 2000*, pages 249–254, 2000.

[58] Allen Van Gelder and Jane Wilhelms. Topological considerations in isosurface generation. *ACM Transactions on Graphics*, 13(4):337–375, 1994.

[59] A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, Englewood Cliffs, NJ., 1981.

[60] F. Girosi, M. Jones, and T. Poggio. Priors, stabilizers and basis functions: From regularization to radial, tensor and additive splines. A.I. Memo No. 1430, MIT, 1993.

[61] GNU Scientific Library. `http://www.gnu.org/software/gsl/`.

[62] G. H. Golub and C. F. Van Loan. *Matrix Computations. 2nd ed.* Johns Hopkins Press, Baltimore, MD., 1989.

[63] Jonas Gomes and Luiz Velho. *Implicit Objects in Computer Graphics.* Série Monografias do IMPA, Rio de Janeiro, 1992.

[64] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In Holly Rushmeier, editor, *Proceedings of ACM SIGGRAPH 96*, pages 43–54. Addison Wesley, 1996.

[65] Henri Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computers*, C-20(6):623—629, 1971.

[66] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulation. *Journal of Computational Physics*, 73:325–348, 1987.

[67] Laurent Grisoni, Christophe Schlick, and Ireneusz Tobor. Rendering by surfels. *Proceedings of the 10th International Conference on Computer Graphics & Vision GraphiCon*, 2000.

[68] J. P. Grossman. Point sample rendering. Master's thesis, Massachusetts Institute of Technology, 1998.

[69] J. P. Grossman and William J. Dally. Point sample rendering. *Eurographics Rendering Workshop 1998*, pages 181–192, 1998.

[70] David (grue) DeBry, Jonathan Gibbs, Devorah DeLeon Petty, and Nate Robins. Painting and rendering textures on unparameterized models. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3):763–768, 2002.

[71] Gaël Guennebaud and Mathias Paulin. Efficient screen space approach for hardware accelerated surfel rendering. In *Proceedings of Vision, Modeling, and Visualization 2003*, 2003.

[72] Mark Hall and Joe Warren. Adaptive polygonalization of implicitly defined surfaces. *IEEE Computer Graphics & Applications*, 10(6):33–42, 1990.

[73] Pat Hanrahan. A survey of ray-surface intersection algorithms. In Andrew Glassner, editor, *Introduction to Ray Tracing*, pages 79–119. Academic Press, 1989.

[74] Pat Hanrahan and Paul E. Haeberli. Direct WYSIWYG painting and texturing on 3D shapes. *Computer Graphics (Proceedings of ACM SIGGRAPH 90)*, 24(4):215–223, 1990.

[75] John C. Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996.

[76] John C. Hart, Ed Bachta, Wojciech Jarosz, and Terry Fleury. Using particles to sample and control more complex implicit surfaces. In *Proceedings of Shape Modeling International 2002*, pages 129–136, 2002.

[77] Erich Hartmann. A marching method for the triangulation of surfaces. *The Visual Computer*, 14(3):95–108, 1998.

[78] Paul Heckbert. Fundamentals of texture mapping and image warping. Master thesis (technical report no. ucb/csd 89/516), University of California, Berkeley, 1989. `http://www.cs.cmu.edu/~ph`.

[79] Paul S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics & Applications*, 6(11):56–67, 1986.

[80] Adrian Hilton, Andrew Stoddart, John Illingworth, and Terry Windeatt. Marching triangles: Range image fusion for complex object modelling. In *Proceedings of the International Conference on Image Processing*, volume 2, pages 381–384, 1996.

[81] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, 26(2):71–78, 1992.

[82] Gregory M. Hunter. *Efficient computation and data structures for graphics.* PhD thesis, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, 1978.

[83] James T. Kajiya. Ray tracing parametric patches. *Computer Graphics (Proceedings of ACM SIGGRAPH 82)*, 16(3):245–254, 1982.

[84] James T. Kajiya. New techniques for ray tracing procedurally defined objects. *Computer Graphics (Proceedings of ACM SIGGRAPH 83)*, 17(3):91–102, 1983.

[85] Aravind Kalaiah and Amitabh Varshney. Differential point rendering. In K. Myskowski and S. Gortler, editors, *Rendering Techniques 2001 (Proceedings of the Eurographics Workshop on Rendering 2001)*, pages 139–150. Springer Verlag, 2001.

[86] Aravind Kalaiah and Amitabh Varshney. Modeling and rendering of points with local geometry. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):30–42, 2003.

[87] D. Kalra and A. H. Barr. Guaranteed ray intersections with implicit surfaces. *Computer Graphics*, 23(3):297–306, 1989.

[88] Tasso Karkanis and A. James Stewart. Curvature-dependent triangulation of implicit surfaces. *IEEE Computer Graphics and Applications*, 22(2):60–69, 2001.

[89] Leif P. Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. Feature-sensitive surface extraction from volume data. In *Proceedings of ACM SIGGRAPH 2001*, pages 57–66, 2001.

[90] Nikita Kojekine, Ichiro Hagiwara, and Vladimir Savchenko. Software tools using CSRBFs for processing scattered data. *Computers & Graphics*, 27(2):311–319, 2003.

[91] Nikita Kojekine, Vladimir Savchenko, Michail Senin, and Ichiro Hagiwara. A prototype system for character animation based on real-time deformations. *The Journal of Three Dimensional Images*, 16(4):91–95, 2002.

[92] Nikita Kojekine, Vladimir Savchenko, Michail Senin, and Ichiro Hagiwara. Real-time 3d deformations by means of compactly supported radial basis functions. In *Proceedings of Eurographics 2002 (Short Paper)*, pages 35–43, 2002.

[93] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. In *Proceedings of ACM SIGGRAPH 96*, pages 313–324, 1996.

[94] Jaroslav Krivanek, Jiri Zara, and Kadi Bouatouch. Fast depth of field rendering with surface splatting. In *Proceedings of Computer Graphics International 2003*, pages 196–201, 2003.

[95] Subodh Kumar, Dinesh Manocha, William Garrett, and Ming Lin. Hierarchical back-face computation. *Eurographics Rendering Workshop 1996*, pages 235–244, 1996.

[96] David Levin. The approximation power of moving least-squares. *Mathematics of Computation*, 67(224):1517–1531, 1998.

[97] David Levin. Mesh-independent surface interpolation. In *4th International Conference on Curves and Surfaces*, page 46, 1999.

[98] David Levin. Mesh-independent surface interpolation. In Guido Brunnett, Bernd Hamann, and H. Müller, editors, *Geometric Modeling for Scientific Visualization*. Springer, Heidelberg, Germany, 2003.

[99] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3D scanning of large statues. In *Proceedings of ACM SIGGRAPH 2000*, pages 131–144, 2000.

[100] Marc Levoy and Turner Whitted. The use of points as display primitive. Technical Report TR 85–022, University of North Carolina at Chapel Hill, 1985.

[101] Erik Lindholm, Mark J. Kilgard, and Henry Moreton. A user-programmable vertex engine. In *Proceedings of ACM SIGGRAPH 2001*, pages 149–158, 2001.

[102] Dani Lischinski and Ari Rappoport. Image-based rendering for non-diffuse synthetic scenes. In *Eurographics Rendering Workshop 1998*, pages 301–314, 1998.

[103] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (ACM SIGGRAPH 87 Proceedings)*, 21(4):163–169, 1987.

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

[105] Matrox. *Matrox Graphics - Parhelia-512.* `http://www.matrox.com`.

[106] Nelson Max and Keiichi Ohsaki. Rendering trees from precomputed z-buffer views. *Eurographics Rendering Workshop 1995*, pages 74–81, 1995.

[107] Maya. *Alias-WaveFont.* `http://www.aliaswavefront.com`.

[108] David K. McAllister, Lars Nyland, Voicu Popescu, Anselmo Lastra, and Chris McCue. Real-time rendering of real world environments. In *Rendering Techniques 1999 (Proceedings of the Eurographics Workshop on Rendering 1999)*, pages 145–160, 1999.

[109] Donald J. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 2(19):129–147, 1982.

[110] Gavin Miller and Andrew Pearce. Globular dynamics: A connected particle system for animating viscous fluids. *Computers & Graphics*, 13(3):305–309, 1989.

[111] Don P. Mitchell. Robust ray intersection with interval arithmetic. In *Proceedings of Graphics Interface '90*, pages 68–74, 1990.

[112] Benjamin Mora, J.P. Jessel, and René Caubet. Visualization of isosurfaces with parametric cubes. *Computer Graphics Forum (Eurographics 2001)*, 20(3):377–384, 2001.

[113] Bryan S. Morse, Terry S. Yoo, Penny Rheingans, David T. Chen, and K. R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Proceedings of Shape Modeling International 2001*, pages 89–98, 2001.

[114] Shigeru Muraki. Volumetric shape description of range data using "Blobby Model". *Computer Graphics (Proceedings of ACM SIGGRAPH 91)*, 25(4):227–235, 1991.

[115] Andrew Nealen and Marc Alexa. Hybrid texture synthesis. In *Proceedings of Eurographics Symposium on Rendering 2003*, pages 97–105, 2003.

[116] Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. Addison-Wesley, 1993.

[117] nVidia. *GeForce FX technical data*. `http://www.nvidia.com`.

[118] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)*, 22(3):463–470, 2003.

[119] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. Multi-scale approach to 3D scattered data interpolation with compactly supported basis functions. In *Proceedings of Shape Modeling International 2003*, pages 153–164, 2003.

[120] Yutaka Ohtake and Alexander G. Belyaev. Dual/primal mesh optimization for polygonized implicit surfaces. *Journal of Computing and Information Science in Engineering*, 2(4), 2002.

[121] Alexander Pasko, Valery Adzhiev, Alexei Sourin, and Vladimir V. Savchenko. Function representation in geometric modelling: concept, implementation and applications. *The Visual Computer*, 11(8):429–446, 1995.

[122] Mark Pauly, Richard Keiser, Leif Kobbelt, and Markus Gross. Shape modeling with point-sampled geometry. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)*, 22(3):641–650, 2003.

[123] Hans Køhling Pedersen. Decorating implicit surfaces. In Robert Cook, editor, *Proceedings of ACM SIGGRAPH 95*, pages 291–300. Addison Wesley, 1995.

[124] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. *Proceedings of ACM SIGGRAPH 2000*, pages 335–342, 2000.

[125] Bui-T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311—317, 1975.

[126] PovRay. *The Persistance of Vision*. `http://www.povray.org`.

[127] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In *Proceedings of ACM SIGGRAPH 2000*, pages 465–470, 2000.

[128] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.)*. Cambridge University Press, 1992.

[129] William T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, 2(2):91–108, 1983.

[130] Liu Ren, Hanspeter Pfister, and Matthias Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. *Computer Graphics Forum (Eurographics 2002)*, 21(3):461–470, 2002.

[131] Patrick Reuter. Efficient real-time level-of-detail rendering using surfels. Master's thesis, Technische Universität Darmstadt, 2001.

[132] Patrick Reuter, Johannes Behr, and Marc Alexa. An improved adjacency data structure for efficient triangle stripping. *accepted for publication in the Journal of Graphics Tools*, To appear.

[133] Patrick Reuter, Ireneusz Tobor, Christophe Schlick, and Sebastien Dedieu. Point-based modelling and rendering using radial basis functions. In *Proceedings of ACM Graphite 2003*, pages 111–118, 2003.

[134] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, 21(4):25–34, 1987.

[135] Steven M. Rubin and Turner Whitted. A 3-dimensional representation for fast rendering of complex scenes. *Computer Graphics*, 14(3):110–116, 1980.

[136] Szymon Rusinkiewicz and Marc Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000*, pages 343–352, 2000.

[137] Szymon Rusinkiewicz and Marc Levoy. Streaming QSplat: a viewer for networked visualization of large, dense models. In *Proceedings of the ACM Symposium on Interactive 3D Graphics 2001*, pages 63–68, 2001.

[138] Vladimir V. Savchenko, Alexander Pasko, Oleg G. Okunev, and Tosiyasu L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14(4):181–188, 1995.

[139] Robert Schaback. Remarks on meshless local construction of surfaces. In R. Cipolla and R. Martin, editors, *The mathematics of surfaces IX*, pages 34–58. Springer, 2000.

[140] Gernot Schaufler and Henrik Wann Jensen. Ray tracing point sampled geometry. In *Rendering Techniques 2001 (Proceedings of the Eurographics Workshop on Rendering 2001)*, pages 319–328, 2000.

[141] Benjamin Schmitt, Alexander Pasko, Valery Adzhiev, and Christophe Schlick. Constructive texturing based on hypervolume modeling. *The Journal of Visualization and Computer Animation*, 12(5):297–310, 2001.

[142] Robert Sedgewick. *Algorithms*. Addison-Wesley, second edition, 1988.

[143] Jonathan Shade, Steven J. Gortler, Li wei He, and Richard Szeliski. Layered depth images. In Michael Cohen, editor, *Proceedings of ACM SIGGRAPH 98*, pages 231–242, 1998.

[144] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):63–70, 1990.

[145] Leon A. Shirman and Salim S. Abi-Ezzi. The cone of normals technique for fast processing of curved patches. *Computer Graphics Forum (Eurographics '93)*, 12(3):261–272, 1993.

[146] Karl Sims. Particle animation and rendering using data parallel computation. *Computer Graphics (Proceedings of ACM SIGGRAPH 90)*, 24(4):405–413, 1990.

[147] Marc Stamminger and George Drettakis. Interactive sampling and rendering for complex and procedural geometry. In K. Myskowski and S. Gortler, editors, *Rendering Techniques 2001 (Proceedings of the Eurographics Workshop on Rendering 2001)*, pages 151–162. Springer Verlag, 2001.

[148] Barton T. Stander and John C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Proceedings of ACM SIGGRAPH 97*, pages 279–286, 1997.

[149] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, 26(2):185–194, 1992.

[150] Ying Tang, Jin Wang, Hujun Bao, and Qunsheng Peng. Rbf-based constrained texture mapping. *Computers & Graphics*, 27(3):415–422, 2003.

[151] Gabriel Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Fifth International Conference on Computer Vision (ICCV'95)*, pages 902–907, 1995.

[152] Ireneusz Tobor, Patrick Reuter, Laurent Grisoni, and Christophe Schlick. Visualisation par surfels. *13ièmes Journées de l'Association Française d'Informatique Graphique*, pages 193–204, 2000.

[153] Nicolas Tsingos, Eric Bittar, and Marie-Paule Gascuel. Implicit surfaces for semi-automatic medical organs reconstruction. In *Computer Graphics International '95*, 1995.

[154] Greg Turk. Generating textures for arbitrary surfaces using reaction-diffusion. *Computer Graphics (Proceedings of ACM SIGGRAPH 91)*, 25(4):289–298, 1991.

[155] Greg Turk. Texture synthesis on surfaces. In *Proceedings of ACM SIGGRAPH 2001*, pages 347–354, 2001.

[156] Greg Turk and James O'Brien. Variational implicit surfaces. Technical Report GIT-GVU-99-15, Georgia Institute of Technology, 1998.

[157] Greg Turk and James F. O'Brien. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics*, 21(4):855–873, 2002.

[158] Luiz Velho. Adaptive polygonization made simple. In *Proceedings of SIBGRAPI '95*, pages 111–118, 1995.

[159] Luiz Velho. Simple and efficient polygonization of implicit surfaces. *Journal of Graphics Tools*, 1(2):5–25, 1996.

[160] Michael Wand, Matthias Fischer, Ingmar Peter, Friedhelm Meyer auf der Heide, and Wolfgang Straßer. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *Proceedings of ACM SIGGRAPH 2001*, pages 361–370, 2001.

[161] Michael Wand and Wolfgang Strasser. Multi-resolution point-sample raytracing. In *Proceedings of Graphics Interface 2003*, 2003.

[162] Li-Yi Wei and Marc Levoy. Texture synthesis over arbitrary manifold surfaces. In *Proceedings of ACM SIGGRAPH 2001*, pages 355–360, 2001.

[163] Holger Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in Computational Mathematics*, 4:389–396, 1995.

[164] Holger Wendland. Fast evaluation of radial basis functions: Methods based on partition of unity. In C. K. Chui, L. L. Schumaker, and J. Stöckler, editors, *Approximation Theory X: Abstract and Classical Analysis*, pages 473–483. Vanderbilt University Press, Nashville, 2002.

[165] Holger Wendland. Surface reconstruction from unorganized points. Preprint Gottingen, 2002.

[166] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of ACM SIGGRAPH 94*, pages 269–278, 1994.

[167] Shin Ting Wu and Marcelo de Gomensoro Malheiros. On improving the search for critical points of implicit functions. In *Proceedings of Implicit Surfaces '99*, pages 137–144, 1999.

[168] Z. Wu. Compactly supported positive definite radial functions. *Advances in Computational Mathematics*, 4:283–292, 1995.

[169] Brian Wyvill, Craig McPheeters, and Geoff Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1986.

[170] Hansong Zhang and Kenneth E. Hoff. Fast backface culling using normal masks. In Michael Cohen and David Zeltzer, editors, *Proceedings of the ACM Symposium on Interactive 3D Graphics 1997*, pages 103–106. ACM SIGGRAPH, 1997.

[171] Hong-Kai Zhao, Stanley Osher, and R. Fedkiw. Fast surface reconstruction using the level set method. In *1st IEEE Workshop on Variational and Level Set Methods*, pages 194–202, 2001.

[172] Hong-Kai Zhao, Stanley Osher, Barry Merriman, and Myungjoo Kang. Implicit and non-parametric shape reconstruction from unorganized points using variational level set method. *Computer Vision and Image Understanding*, 3(80):295–319, 2000.

[173] Ruben Zonenschein, Jonas Gomes, Luiz Velho, and Luiz Henrique de Figueiredo. Controlling texture mapping onto implicit surfaces with particle systems. In *Proceedings of Implicit Surfaces '98*, pages 131–138, 1998.

[174] Matthias Zwicker, Mark Pauly, Oliver Knoll, and Markus Gross. Pointshop 3D: An interactive system for point-based surface editing. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3):322–329, 2002.

[175] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In *Proceedings of ACM SIGGRAPH 2001*, pages 371–378, 2001.

Index

W