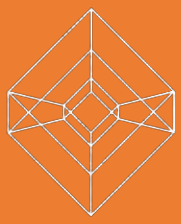


HARVEST4D

HARVESTING DYNAMIC 3D WORLDS FROM COMMODITY SENSOR CLOUDS

Publications for Task 3.2

Deliverable 3.21

Date:	10.07.2015
Grant Agreement number:	EU 323567
Project acronym:	HARVEST4D
Project title:	Harvesting Dynamic 3D Worlds from Commodity Sensor Clouds

Document Information

Deliverable number	D3.21
Deliverable name	Publications for Task 3.2
Version	0.4
Date	2015-07-10
WP Number	3
Lead Beneficiary	PARISTEC
Nature	R
Dissemination level	PU
Status	Final
Author(s)	PARISTEC

Revision History

Rev.	Date	Author	Org.	Description
0.1	2015-06-30	Tamy Boubekeur	PARISTEC	Initial Draft
0.2	2015-07-01	Tamy Boubekeur	PARISTEC	Beta Version
0.3	2015-07-08	Tamy Boubekeur	PARISTEC	Final Version
0.4	2015-07-10	Michael Wimmer	VUT	Typos

Statement of originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

TABLE OF CONTENTS

1 EXECUTIVE SUMMARY

1.1 INTRODUCTION

This deliverable describes the publications that resulted from Task 3.2, and how they fit into the work plan of the project.

The objective of Task 3.2 is to develop high level shape processing and analysis algorithms, some of them being based on the operators of Task 3.1 (see Delivrable D3.11). These algorithms are designed to process the data acquired within the Harvest4D project. During the second period of the Harvest4D project, we have developed a number of shape algorithms to alter the geometry and topology of 3D point clouds and meshes. We have particularly focused on analysis algorithms for point clouds, as the constraints of Harvest4D – processing large streams of 3D data frequently updated – do not always allow for a full surface mesh reconstruction, while analysis may still be mandatory for simplification, repairing, editing or recognition. In particular, this task has led to a journal publication in the ACM Transactions on Graphics (Proceedings of the SIGGRAGH 2014 conference) regarding the expression of a powerful shape analysis framework, *Mathematical Morphology*, in the context of point data. A second component of this deliverable addresses the problem of data structures for efficient collision detection, which exploits a screen-space formulation to contain both memory and computation cost. A couple of other publications are primarily attached to other tasks or work packages but have nonetheless also contributed advances in geometric algorithms. In the following, we give an overview of the main contributions.

So far, there are 2 publications that are mainly associated to Task 3.2, and these can be found in the appendix of this deliverable:

- Stéphane Calderon and Tamy Boubekeur
  Point Morphology
  ACM Transaction on Graphics (Proc. SIGGRAPH), 2014
- Mohamed Radwan, Stefan Ohrhallinger, and Michael Wimmer
  Efficient collision detection while rendering dynamic point clouds
  Proceedings of Graphics Interface 2014 (GI '14), 2014

Additionally, a number of other papers are primarily attached to other tasks but exhibit contributions on geometric algorithms:

- Jean-Marc Thiery, Emilie Guy and Tamy Boubekeur
  Sphere-Meshes: Shape Approximation Using Spherical Quadric Error Metrics
  ACM Transactions on Graphics (Proc. SIGGRAPH Asia), 2013

- Noura Faraj, Jean-Marc Thiery, Tamy Boubekeur
  Progressive Medial Axis Filtration
  ACM SIGGRAPH Asia 2013, Technical Brief Program
- Thiery Guillemot, Andrès Almansa, Tamy Boubekeur
  Covariance Trees for 2D and 3D Processing
  IEEE Conference on Computer Vision and Pattern Recognition (CVPR, Oral), 2014
- Emilie Guy, Jean-Marc Thiery, Tamy Boubekeur
  SimSelect: similarity-based selection for 3D surfaces
  Computer Graphics Forum (Proc. EUROGRAPHICS 2014), 33(2):165-173, 2014
- Reinhold Preiner, Oliver Mattausch, Murat Arikan, Renato Pajarola, Michael Wimmer
  Continuous Projection for Fast L1 Reconstruction
  ACM Transactions on Graphics (Proc. SIGGRAPH), 2014
- Simon Fuhrmann and Michael Goesele
  Floating Scale Surface Reconstruction
  ACM Transactions on Graphics (Proc. SIGGRAPH 2014), 2014
- Tim Tutenel, Christian Kehl and Elmar Eisemann
  Interactive visual analysis of flood scenarios using large-scale LiDAR point clouds
  Geospatial World Forum 2013
- Tamy Boubekeur
  ShellCam: Interactive Geometry-Aware Virtual Camera Control
  IEEE International Conference on Image Processing, to appear
- Emilie Guy, Jean-Marc Thiery, Tamy Boubekeur
  SimSelect: similarity-based selection for 3D surfaces
  Computer Graphics Forum (Proc. of EUROGRAPHICS 2014), 33(2), p.165-173, 2014
- Tamy Boubekeur
  Mesh Reconstruction from a Point Cloud
  Chapter in Digital Representations of the Real World: How to Capture, Model and Render Visual Reality, 2015

2 DESCRIPTION OF PUBLICATIONS

2.1 OVERVIEW

The two main contributions to this deliverables are the point morphology framework (Section 2.2) and a new screen-space approach to collision detection (Section 2.3). In the following, we give more details about these two methods. However, one shall note that “geometric algorithms” cover a wide range of applications and methods within the Harvest4D project, and appear in many of the Harvest4D publications, work-packages and tasks. We listed earlier the main ones and give a brief overview of their link to Task 3.2 (Section 2.4).

2.2 POINT MORPHOLOGY

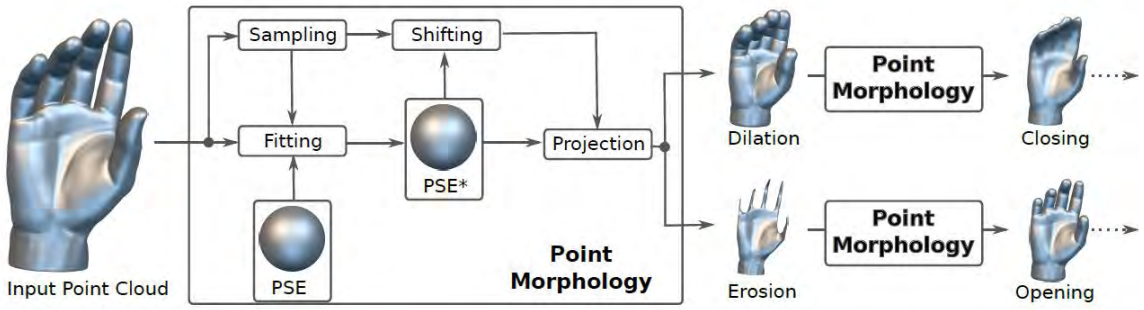


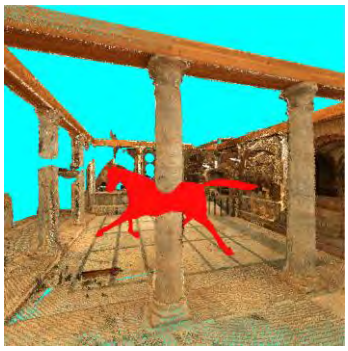
Figure 1 Overview of our Point Morphology framework. Simple chains of morphological operators yield powerful shape analysis, such as the tight bounding "closing" obtained on the top right, or the tight inset "opening" on the bottom right. In both cases, a large part of the geometric features are preserved.

Shape analysis covers a diverse set of methods aiming at extracting parameters from raw 3D shapes, to fuel automatic signal-inspired processing primitives, interactive shape editing tools or advanced shape recognition engines. Large part of these methods are specialized and tuned for one particular application, being defined using a mesh structure most of the time. In the context of Harvest4D, we cannot always afford reconstructing a mesh before analyzing it. Clearly, pulling the analysis stage earlier in the pipeline – right after geometry emerges from capture in the form of an unstructured point cloud for instance – opens new way of interpreting the acquired content. In particular, we aim at developing an analysis framework than can be used in multiple contexts, while still relying on a small set of operators that act directly on point sets. Under these constraints, *Mathematical Morphology*, which is a powerful image analysis framework, appears as a good candidate. But expressing it for point clouds requires a complete reformulation of its basis.

To address this challenge, we introduce a complete morphological analysis framework for 3D point clouds called *Point Morphology* [Calderon and Boubekeur 2014]. Starting from an unorganized point set sampling a surface, we propose morphological operators in the form of projections, allowing sampling erosions, dilations, closings and openings of an object without any explicit mesh structure. Our framework supports structuring elements with arbitrary shape, accounts robustly for geometric and morphological sharp features, remains efficient at large scales and comes together with a specific adaptive sampler. Based on this meshless framework, we propose applications which benefit from the non-linear nature of morphological analysis and can be expressed as simple sequences of our operators, including medial axis sampling, hysteresis shape filtering and geometry-preserving topological simplification.

2.3 EFFICIENT COLLISION DETECTION WHILE RENDERING DYNAMIC POINT CLOUDS

Data flowing in the Harvest4D pipelines often come as unstructured and temporally varying point clouds, such as the one generated by affordable depth cameras or used in augmented-reality simulations. Performing collision detection on such models has numerous applications in interactive graphics applications. State-of-the-art methods for collision detection usually create a spatial hierarchy in order to capture these dynamic point cloud surfaces, but they require



O(N logN) construction time for N points. We propose a novel screen-space representation for point clouds that exploits the property of the underlying surface being 2D [Radwan et al. 2014]. To perform dimensionality reduction, a 3D point cloud is converted into a series of thickened layered depth images. This data structure can be constructed in O(N) time and allows for fast surface queries due to its increased compactness and memory coherency. Moreover, parts of its construction come for free, being already handled by the rendering pipeline. As an application, we demonstrate online collision detection between dynamic point clouds. It shows superior accuracy when compared to other methods and robustness to sensor noise since uncertainty is hidden by the thickened boundary.

2.4 OTHER RESULTS

Beyond the two previous papers, a number of Harvest4D publications have contributed geometric algorithms. In particular, as part of this deliverable, we have derived a full automatic shape-rigging algorithm based on the sphere-mesh approximation [D3.11][Thiery et al. 2013] that allows loading raw meshes and instantaneously equip them with a multi-resolution control structure allowing to deform them easily. In the long run, Harvest4D partners also try to explore alternative representations for geometric data and aim at fueling them with high-level processing primitives. Among them, the medial axis, discussed in Point Morphology (Section 2.2), is a good candidate to link surface-based and volume-based representations. In this context, our *progressive medial axis filtration* (PMAT [Faraj et al. 2013]) method introduces a fast scheme to remove small components from medial axis representations while preserving their main structures, instantaneously. Statistical representations are another direction explored by Harvest4D, e.g., Gaussian mixtures exploited in Point Morphology and CLOP (D3.11). In this respect, the Covariance Tree [Guillemot et al. 2014] proposes a new hierarchical representation to collaboratively model and process data streams such as high-resolution point clouds. Beyond geometric algorithms that mimic signal processing primitives, shape analysis is instrumental in work package 8. In particular, geometric analysis for interactive navigation (D8.41) and similarity modeling (D8.51/D8.52) make intensive use of both meshless geometric operators (e.g., moving least squares [Boubekeur 2014]) and statistical geometric tools (e.g., our new approximate shape context descriptor [Guy et al. 2014]). We refer the reader to the different papers listed earlier for more technical details.

3 APPENDIX

The following pages contain all the publications that are directly associated with this deliverable. Other publications referenced in this deliverable can be found in the public Harvest4D webpage (for already published papers), or in the restricted section of the webpage (for papers under submission, conditionally accepted papers, etc.).

Point Morphology

Stéphane Calderon Tamy Boubekeur
Telecom ParisTech - CNRS LTCI - Institut Mines-Telecom

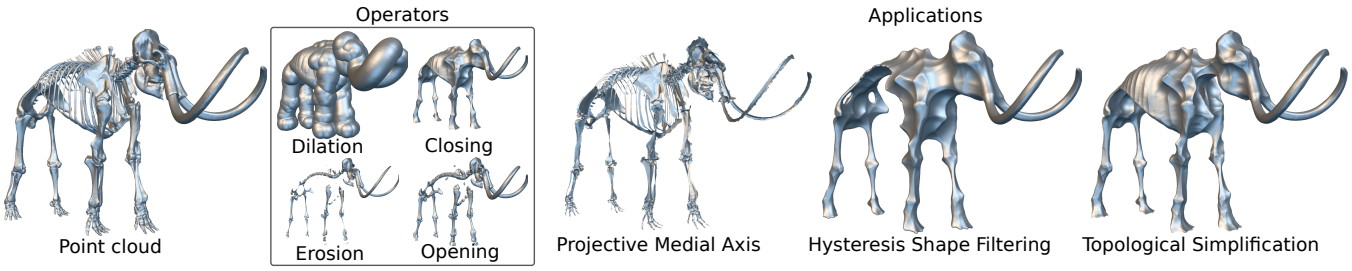


Figure 1: *Starting from an unstructured 3D point cloud, we define morphological operators based on a single projection procedure and propose advanced shape analysis applications in the form of simple sequences of these operators.*

Abstract

We introduce a complete morphological analysis framework for 3D point clouds. Starting from an unorganized point set sampling a surface, we propose morphological operators in the form of projections, allowing to sample erosions, dilations, closings and openings of an object without any explicit mesh structure. Our framework supports structuring elements with arbitrary shape, accounts robustly for geometric and morphological sharp features, remains efficient at large scales and comes together with a specific adaptive sampler. Based on this meshless framework, we propose applications which benefit from the non-linear nature of morphological analysis and can be expressed as simple sequences of our operators, including medial axis sampling, hysteresis shape filtering and geometry-preserving topological simplification.

CR Categories: Computer Methodologies [Computer Graphics]: Shape Modeling—Point-based Models, Shape Analysis;

Keywords: point-based modeling, shape analysis, morphology

Links: DL PDF

1 Introduction

Point-based modeling aims at processing, analyzing and interacting with digital shapes which are represented by unorganized point samplings without any explicit connectivity. The related set of meshless operators are particularly attractive in the context of 3D and 4D capture but can also benefit any computer graphics application as long as they can provide a point sampling of their surface models. For instance multiple registered range maps coming from laser scanners, dense point sets generated using multiview stereovision or large polygon soups designed in CAD software can all be expressed as a list of point samples with attributes and consequently be processed within the same point-based pipeline.

Standard point-based methods take place at the earliest stages of the processing pipeline, prior to mesh reconstruction and are often based on operators which alter the point sampling and embedding. The majority of these operators mimic the classical signal processing primitives, namely filtering, sampling and reconstruction. They commonly allow to remove noise, increase/decrease the point density or improve its distribution. Although these local geometric computations significantly enhance data quality for the upcoming processing steps, the global analysis of the shape is usually delayed until post-meshing stages, where the mesh connectivity makes it possible explore its components, medial structures and topology.

This typical pipeline has two major drawbacks: first, the shape analysis is performed too late to avoid dealing with topology changes and large geometric alterations on a mesh structure, which is often unstable and prone to artifacts, in particular when the manifoldness of the mesh is not guaranteed; second, the meshing process itself would benefit from this analysis if performed at the earliest stages. Actually, it would be preferable to define shape analysis methods which can act directly on the point set and influence the global geometry and (implicit) topology of the shape prior to the reconstruction of a mesh, if ever required.

Among the various shape analysis frameworks that exist for structured 2D and 3D data, *mathematical morphology* appears as one of the most powerful and versatile, with the advantage of providing a large number of high level shape transformations employing a restricted set of operators. Image filtering, segmentation, skeletonization, recognition and many other processes have successfully benefited from discrete mathematical morphology, in the context of medical imaging, metrology and robotics.

In this paper, we propose *point morphology* (see Fig. 1), a complete morphological framework for analyzing and processing 3D point sets (Sec 3). Using a new model for the underlying structuring element (Sec. 3.2), we reformulate the basic morphological operators, namely erosion and dilation, as *projections* (Sec 3.3). Used with a new feature-aware point sampler (Sec 3.5), we define closing and opening accounting robustly for the sharp morphological structures which appear even on simple shapes, keeping computations tractable.

Our operators are simple to implement, scalable and robust: we evaluate them on a wide range of inputs (see Sec. 4). Based on this framework, we perform non-trivial shape transformations directly on point-based models using simple sequences of point morphological operators. We illustrate this property by proposing several applications (Sec. 5), including projective medial axis modeling, hysteresis shape filtering and geometry-preserving topological simplification.

2 Background

Our approach is based on two distinct fields: point set surfaces and mathematical morphology. In the following, we recall their basic principles before discussing recent related work. For both, we consider a shape as a compact subset B of $\mathbb{R}^3$ and its variational representation as a scalar field $\mathscr{B} : \mathbb{R}^3 \rightarrow \mathbb{R}$:

$$\mathscr{B}(\mathbf{x}) = \begin{cases} < 0, & \text{if } \mathbf{x} \in \mathring{B} \\ 0, & \text{if } \mathbf{x} \in \partial B \\ > 0, & \text{if } \mathbf{x} \notin B \end{cases} \quad (1)$$

2.1 Point Set Surfaces

A *Point Set Surface* [Alexa et al. 2001; Amenta and Kil 2004] (PSS) models a smooth manifold from an unorganized 3D point cloud based on a projection operator. This representation has been successfully used for the complete point-based modeling chain, including resampling, reconstruction, analysis, editing, compression and visualization.

PSS Definition Let us consider $\Pi = \{\boldsymbol{\pi}_i = (\mathbf{p}_i, \mathbf{n}_i)\}$ a set of surface samples, with $\mathbf{p}_i \in \mathbb{R}^3$ (resp. $\mathbf{n}_i \in \mathbb{R}^3$) the sample's position (resp. normal), and $\mathbf{x} \in \mathbb{R}^3$. The Moving Least Square (MLS) projection [Levin 1998; Levin 2003; Alexa et al. 2004] is defined as:

$$MLS_\Pi : \mathbb{R}^3 \rightarrow \mathbb{R}^3, \mathbf{x} \mapsto \mathcal{P}(\mathbf{x}) \quad (2)$$

The operator $\mathcal{P}(\mathbf{x})$ embeds two fundamentals procedures:

1. **fitting**: optimizes a weighted least squares primitive B that approximates Π around $\mathbf{x}$.,
2. **projection**: projects $\mathbf{x}$ onto B.

A PSS is defined in its *projective form* as the stationary set of $\mathbb{R}^3$ under this MLS projection (see Fig. 2):

$$PSS = \{\mathbf{x} \in \mathbb{R}^3 | \mathcal{P}(\mathbf{x}) = \mathbf{x}\} \quad (3)$$

To reach the PSS from any $\mathbf{x} \in \mathbb{R}^3$ we simply iterate MLS_Π until convergence (for any $\mathbf{x} \in \mathbb{R}^3$, $\mathcal{P}^\infty(\mathbf{x}) = \mathcal{P} \circ .. \circ \mathcal{P}(\mathbf{x}) \in PSS$).

Shape Fitting The fitted shape B is parameterized by a vector field $\mathbf{q}^* : \mathbb{R}^3 \rightarrow \mathbb{R}^d, \mathbf{x} \rightarrow \mathbf{q}^*(\mathbf{x})$ so that $B := B_{\mathbf{q}^*}$. This vector field defines a set of parameters modeling the fitted primitive (e.g., position and radius if B is a sphere, position and orientation if B is a plane, etc.) at each point in space, approximating Π around $\mathbf{x}$:

$$\mathbf{q}^*(\mathbf{x}) = \underset{\mathbf{q}}{\operatorname{argmin}} \sum_i \omega_\sigma \left(\|\mathbf{x} - \mathbf{p}_i\| \right) d\left(\mathbf{q}, \boldsymbol{\pi}_i\right)^2 \quad (4)$$

where ω_σ is a smoothly decaying weighting kernel ensuring partition of unity in the sum. The scale at which B is fitted to Π is typically controlled by a parameter σ which relates to the support size (or influence radius) of ω_σ. We consider $d\left(\mathbf{q}, \boldsymbol{\pi}_i\right)^2$ the distance between the primitive defined by $\mathbf{q}$ and an input sample.

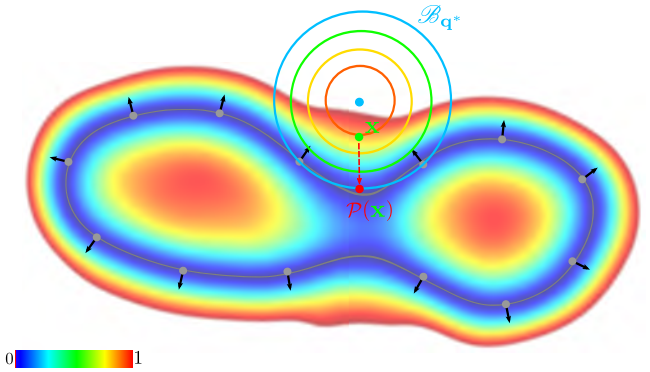


Figure 2: ***PSS principle in 2D.*** *The input point set is represented with grey dots and black normals. In green a point* $\mathbf{x}$ *candidate for a projection; in gradient color circles* $|\mathscr{B}_{\mathbf{q}^*}|$ *(here the signed distance field's absolute value) of the fitted primitive (a circle) at* $\mathbf{x}$ *and its parameters* $\mathbf{q}^*$*; in red the projection* $\mathcal{P}(\mathbf{x})$ *onto* $\mathscr{B}_{\mathbf{q}^*}$*. The point set "curve" (resp. the absolute value of its* implicit form*) are represented in grey (resp. gradient color) in the background.*

Shape Projection Given the locally fitted primitive $B_{\mathbf{q}^*}$ we project onto it through:

$$\mathcal{P}(\mathbf{x}) = \mathbf{x} - \mathscr{B}_{\mathbf{q}^*}(\mathbf{x}) \frac{\nabla \mathscr{B}_{\mathbf{q}^*}(\mathbf{x})}{\left\| \nabla \mathscr{B}_{\mathbf{q}^*}(\mathbf{x}) \right\|} \quad (5)$$

where $\mathscr{B}_{\mathbf{q}^*}(\mathbf{x})$ is the variational shape representation of $B_{\mathbf{q}^*}$. Beyond this projective form, the PSS has also an *implicit form* defined as the zero set of a scalar field $\mathscr{I}_\Pi(\mathbf{x}) = \mathscr{B}_{\mathbf{q}^*(\mathbf{x})}(\mathbf{x})$, both being related by:

$$\mathcal{P}(\mathbf{x}) = \mathbf{x} \Leftrightarrow \mathscr{I}_\Pi(\mathbf{x}) = 0.$$

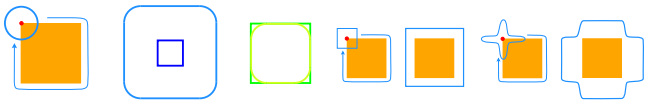
PSS Models Popular instances of this general PSS definition include the Simple PSS (SPSS) model [Adamson and Alexa 2004] which uses an implicit plane representation for B — $\mathbf{q} = (\mathbf{c}, \mathbf{n})$, $\mathscr{B}_{\mathbf{q}}(\mathbf{x}) = (\mathbf{x}-\mathbf{c})\cdot\mathbf{n}$, and $d\left(\mathbf{q}, \boldsymbol{\pi}_i\right)^2 = \|\mathbf{c} - \mathbf{p}_i\|^2 + \|\mathbf{n} - \mathbf{n}_i\|^2$ with $\mathbf{c}$ (resp. $\mathbf{n}$) the center (resp. normal) of the plane — and the Algebraic PSS (APSS) model [Guennebaud and Gross 2007] which uses an algebraic sphere representation for B — $\mathscr{B}_{\mathbf{q}}(\mathbf{x}) = [1, \mathbf{x}, \mathbf{x}^T\mathbf{x}] \cdot \mathbf{q}$ and $d\left(\mathbf{q}, \boldsymbol{\pi}_i\right)^2 = \beta \left\| \nabla \mathscr{B}_{\mathbf{q}}(\mathbf{p}_i) - \mathbf{n}_i \right\|^2 + \left\| \mathscr{B}_{\mathbf{q}}(\mathbf{p}_i) \right\|^2$ with β weighting the derivative constraints.

A number of PSS variations have been proposed, including hermitian interpolation [Alexa and Adamson 2009], scale-space representation [Pauly et al. 2006; Mellado et al. 2012], point cell complex definition [Adamson and Alexa 2006] and feature preservation [Fleishman et al. 2005; Reuter et al. 2005; Öztireli et al. 2009]. Essentially, PSS models allow to process and analyze point clouds by varying the support of the kernel (e.g., large supports act as low-pass filters), mimicking the Gaussian analysis of signals.

2.2 Mathematical Morphology

Every signal analysis theory uses a set of transformation operators revealing its structure. For instance, linear analysis is based on convolutions since constraining the operators to be linear and translation invariant naturally gives rise to a convolution [Babaud et al. 1986]. Giving up the linear constraint widens the signal exploration.

Mathematical Morphology [Serra 1983] (or morphology) is a shape analysis theory exploiting non-linear operators which intuitively alter the object at every point with a particular shape B called in this



(a) (b)

Figure 3: ***Continuous Morphology.*** ***(a)*** *A circular SE (light blue) sweeping a binary input (orange) with resulting Dilation (light blue), Erosion (blue), Closing (green) and an Opening (light green).* ***(b)*** *The shape of the SE influences strongly morphological transformations (blue) for the same input (orange).*

context the *structuring element* (or SE). Given the shape I of an object, the two basic operators are the *dilation* $D_{I,B} = I \oplus B$ and the *erosion* $E_{I,B} = I \ominus B$ where $\oplus$ and $\ominus$ are Minkowski sum and subtraction i.e., $I \oplus B = \bigcup_{\mathbf{y} \in I} B_{\mathbf{y}}$ and $I \ominus B = \overline{\overline{I} \oplus B^{\dagger}}$ where $B_{\mathbf{y}} = \{\mathbf{b} + \mathbf{y} | \mathbf{b} \in B\}$ is the translated SE, $\bar{}$ is the complementary operator and $B^{\dagger}$ is the symmetric of B w.r.t to $\mathbf{0}$. These operators are combined to define a *Closing* $C_{I,B} = E_{[D_{I,B}],B}$ and an *Opening* $O_{I,B} = D_{[E_{I,B}],B}$ (see Fig. 3). As we shall see later, the choice of B strongly influences the resulting transformation as well as the performed analysis of I. Unlike raster images, the case of 3D surfaces is usually addressed using *continuous* morphology through a binary function classifying the ambient space as either inside or outside the object.

Mathematical morphology has a large spectrum of applications, including scale-space analysis, skeletonization, segmentation, compression and micro-structure modeling. We refer to the book of Najam and Talbot [2010] for a recent survey.

2.3 Related Work

Mathematical morphology has been so far mostly used in its discrete form, for 2D and 3D images. However, Minkowski sums have been studied for polyhedral meshes and point sets in several works.

Meshes Barki et al. [2011] introduced a method to compute Minkowski sums of fold-free polyhedron with a convex polyhedral SE. They propose the notion of *contributing vertices* to build a tight superset of geometric primitives. Then the exact Minkowski sum is extracted from this superset. Another approach by Campen et al. [2010] permits exact Minkowski sum computation with an arbitrary SE and an efficient computational framework. However, contrary to Barki et al., only the outer boundary of the sum is extracted, leaving the inner boundary to a grid structure and a prior knowledge of its location. In both cases, a clean mesh input is required, which is typically not available at the early stage of the modeling pipeline.

Points Sets Observing that the signed distance function (SDF) of a surface encompasses dilations by a spherical SE, Molchanov et al. [2010] use directly the SDF of an algebraic point set surface [Guennebaud and Gross 2007] to define a Minkowski sum. This formulation provides a smooth output but is restricted to spherical SEs and presents defects in the vicinity of singular points of the SDF (medial axis). In practice, hard thresholding on the neighborhood selection is used to decide which part of the point set surface is taken into account in the SDF evaluation. Indeed a proper medial axis model (i.e., smoothness) is not guaranteed, which becomes problematic around the many sharp edges appearing when dilating.

Lien et al. [2007] define purely point-based Minkowski sums and do not aim at representing the morphological transformation as a continuous surface, modeling the SE itself as a point set. First, at each point of the input point set, all SE points are added to the output. Second, this superset is decimated to remove all the points that do not belong to the dilation. The result is a point sampling of the dilation. Although very simple, this approach has several drawbacks. First, smooth reconstructions of the resulting point set often gives rise to strong artifacts, in particular for non spherical SEs. Second, the sharp features emerging from the sums and subtractions, which are critical in morphological analysis, are not captured. Third, the computational cost, with an intermediate sampling having the complexity of the model times the SE, is prohibitive for dense input (millions of points) and/or complex SEs (thousands of points).

Peternell et al. [2007] and Nelaturi et al. [2009] use a similar approach but then proceed with either a grid based decimation [Peternell and Steiner 2007] or a flood filling [Nelaturi and Shapiro 2009] of the resulting sum to extract the outer boundary of the dense result, producing similar caveats. Chen et al. [2005] compute *offsets* (dilation with a spherical SE) in a similar fashion as Lien et al. [2007] and Peternell et al. [2007]. However, the SE's sampling for the sum is sensitive to the input surface's curvature, reducing the magnitude of the decimation stage.

Most of these methods rely on the construction of a superset of points and extract the Minkowski sum by decimating it. Such solutions are perfectly valid for the computation of a single sum. Unfortunately, morphological algorithms require **sequences** of sums and subtraction, which has at least three consequences: (i) the intermediate shape produced at a given step of the sequence should be properly resampled for the next step; (ii) the sharp features appearing during the sequence should be preserved, independently of the input density, as they typically capture the structure revealed by morphology; (iii) an end-to-end local computation avoiding the generation of supersets is required in practice to process real data in a reasonable amount of time. Our framework addresses these three issues.

3 Method

3.1 Overview

Our goal is to compute erosions, dilations, openings and closings of a surface point cloud Π. To do so, we adopt a *projective* approach where these morphologies are seen through the projection of the surrounding space. This allows us to compute them without explicit connectivity in the input, using any structuring element, scaling to large data by bypassing intermediate supersets and preserving the rising sharp structures robustly.

In practice, our framework (summarized in Fig. 4) is composed of three main components: (i) a *point structuring element* model which can have any shape and size, (ii) a *projection procedure* substituting the explicit Minkowski sum and (iii) a *feature-aware sampler* distributing points on the transformed shape.

In the following, we start by explaining how to project a single point $\mathbf{x} \in \mathbb{R}^3$ onto the dilation (resp. erosion) of the point cloud. This operation requires the optimization of the SE for $\mathbf{x}$ w.r.t. the point cloud before projecting $\mathbf{x}$ onto it (see Sec. 3.3) to reach the dilated (resp. eroded) shape (see Sec. 3.4). Then, we explain how to sample these morphologies properly to supply forthcoming alterations (see Sec. 3.5). Last, we describe the computation of closings and openings (see Sec. 3.6).

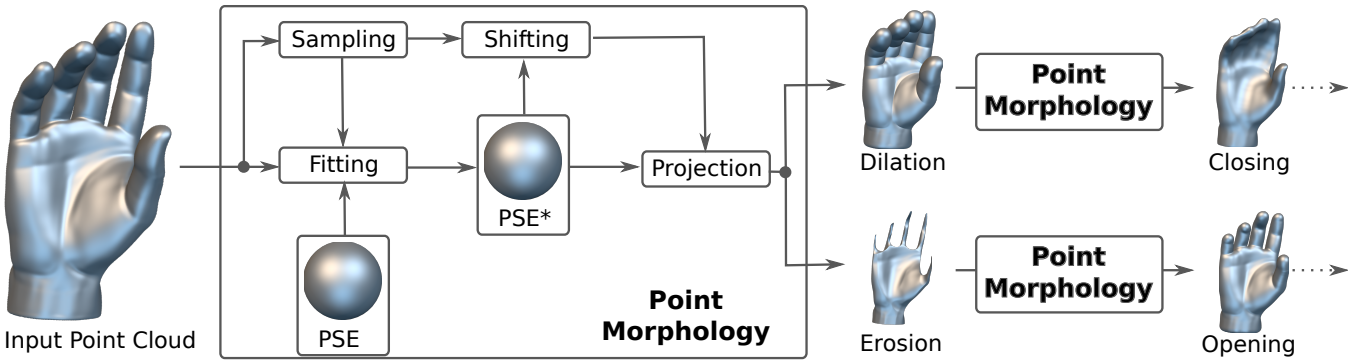


Figure 4: ***Overview:*** *our framework samples dilation, erosion, closing and opening of a point cloud.*

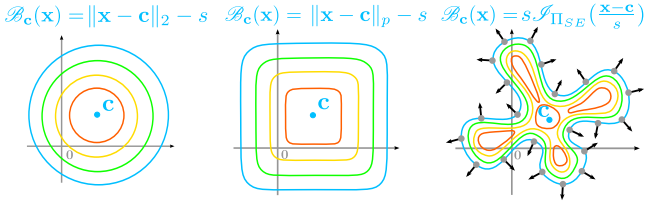


Figure 5: ***Point structuring element****: three PSEs with their distance field iso-contours in gradient color.*

3.2 Point Structuring Element

Analyzing a point cloud is challenging as no explicit topological space is available. However, we observe that, starting from $\mathbf{x}$, fitting a single SE to Π is sufficient to reach the dilation or erosion of Π as long as we can express a signed distance from $\mathbf{x}$ to the SE boundary. Therefore, we propose to model the SE itself as a *signed scalar field* and use an MLS-inspired optimization procedure to locate it w.r.t. $\mathbf{x}$. More formally, given a shape B, its signed distance field $\mathscr{B}$, a scale s and a center $\mathbf{c}$, we define a *Point Structuring Element* (or PSE, see Fig. 5) $\mathscr{B}_{\mathbf{c}}$ as:

$$\mathscr{B}_{\mathbf{c}}(\mathbf{x}) = s\,\mathscr{B}(\frac{\mathbf{x}-\mathbf{c}}{s}) \tag{6}$$

Simple PSEs, from spherical to cubic-like shape, are modeled analytically using the L_p norm:

$$\mathscr{B}_{\mathbf{c}}(\mathbf{x}) = \|\mathbf{x}-\mathbf{c}\|_p - s \tag{7}$$

For more complex PSEs, such simple analytical forms are usually not available and we rely on the IMLS field [Kolluri 2008] of a point sampling Π_B of B as it is close enough to a distance one:

$$\mathscr{B}_{\mathbf{c}}(\mathbf{x}) = s\,\mathscr{I}_{\Pi_B}(\frac{\mathbf{x}-\mathbf{c}}{s}) \tag{8}$$

3.3 Morphological Projection

We recall from classical set morphology that $D_{I,B} = \bigcup_{\mathbf{c}\in I} B_{\mathbf{c}}$. With our PSE model in hand, we can translate the *set* operator $\bigcup$ into a *variational* form. Using $\mathscr{I}$ the variational representation of the shape I sampled by Π, we define a variational dilation $D_{\mathscr{I},\mathscr{B}}$ as:

$$\begin{array}{rccl} D_{\mathscr{I},\mathscr{B}}: & \mathbb{R}^3 & \rightarrow & \mathbb{R} \\ & \mathbf{x} & \mapsto & \min(\mathscr{B}_{\mathbf{c}*}(\mathbf{x}), \mathscr{I}(\mathbf{x})) \end{array}$$

with $\mathbf{c}^*$ the optimized center of the PSE:

$$\mathbf{c}^* = \underset{\substack{\mathbf{c}\in\mathbb{R}^3 \\ \mathscr{I}(\mathbf{c})=0}}{\operatorname{argmin}} \mathscr{B}_{\mathbf{c}}(\mathbf{x}) \tag{9}$$

We use the optimized structuring element $\mathscr{B}_{\mathbf{c}*}$ within a an MLS-inspired projection procedure (see Appendix for a derivation from *set* morphology) which is composed of two steps:

1. **fitting**: optimizes a primitive $\mathscr{B}_{\mathbf{c}*}$ that approximates the morphological alteration of Π around $\mathbf{x}$,
2. **projection**: project $\mathbf{x}$ onto $\mathscr{B}_{\mathbf{c}*}$.

PSE Fitting Intuitively, fitting the PSE corresponds to moving its center $\mathbf{c}$ on the surface of the shape sampled by Π so that the distance between $\mathbf{x}$ and the PSE is minimized. This boils down to the optimization of $\mathbf{c}$ through Eq. 9 in which we choose $\mathscr{I}$ as the implicit form of a PSS of Π (see Sec. 2.1). We approximate a solution to this problem by running a mean shift procedure [Fukunaga and Hostetler 1975] on a point sampling $\overline{\Pi}$ of $\mathscr{I}$ (i.e., $\forall \boldsymbol{\pi}_i \in \overline{\Pi}, \mathscr{I}(\mathbf{p}_i) = 0$). Note that if Π is dense and we chose an interpolating PSS model for $\mathscr{I}$, we can safely set $\overline{\Pi} := \Pi$. We initialize the mean shift with several meaningful (usually 2) points of $\overline{\Pi}$ to find different local candidate minimizers of Eq. 9 i.e., $\{\mathbf{c}_j^0\} := \{\text{closest points in } \overline{\Pi} \text{ under PSE distance}\}$:

$$\mathbf{c}_j^k(\mathbf{x}) = \sum_i \omega_\sigma(\|\mathbf{c}_j^{k-1}(\mathbf{x}) - \mathbf{p}_i\|)\,\omega_\sigma(\mathscr{B}_{\mathbf{p}_i}(\mathbf{x}) + s)\mathbf{p}_i \tag{10}$$

After convergence ($\mathbf{c}_j^k(\mathbf{x}) \rightarrow \mathbf{c}_j^*$ for $k \rightarrow \infty$) we choose a global minimizer as:

$$\mathbf{c}^*(\mathbf{x}) = \underset{\mathbf{c}_j^*}{\operatorname{argmin}}\ \mathscr{B}_{\mathbf{c}_j^*}(\mathbf{x}) \tag{11}$$

Projection Once the PSE is fitted, we can compute the morphological projection of $\mathbf{x}$ (see Fig. 6, left):

$$\mathcal{P}_{\mathscr{B}}(\mathbf{x}) = \mathbf{x} - \mathscr{B}_{\mathbf{c}*}(\mathbf{x})\frac{\nabla\mathscr{B}_{\mathbf{c}*}(\mathbf{x})}{\|\nabla\mathscr{B}_{\mathbf{c}*}(\mathbf{x})\|} \tag{12}$$

At this stage, the stationary set of $\mathbb{R}^3$ under $\mathcal{P}_{\mathscr{B}}$ is made of two *crusts* and we cannot distinguish dilation from erosion yet. However, we can already observe that Eq. 10 and 11 give the basis of a continuous robust classification of our piecewise smooth morphologies: as for each point $\mathbf{x}$ we consider a surface with (potentially) several components (modes), the projection using Eq. 11 is equivalent to a projection on the union of the PSE optimizers (see Fig. 7).

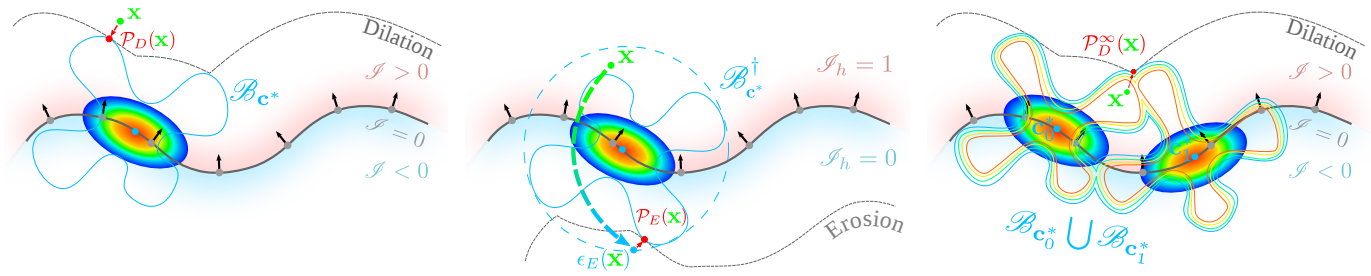


Figure 6: ***Morphological projection*** *with* $\mathbf{x}$ *the green point, its morphological projection the red point and* Π *the grey dots with black normals. Left: dilation projection with the mean shift objective function depicted in gradient color. Middle: erosion projection reached through the shifting procedure (*$\epsilon_E(\mathbf{x})$*). Right: two local minimizer configuration giving raises to sharp features.*

With this formulation, two pronounced modes (or local optimizers) can continuously merge into a single one, leading to a continuous transition from a sharp crease to a regular smooth surface (see Sec. 4 for examples).

3.4 Dilation and Erosion

Our morphological projection $\mathcal{P}_{\mathscr{B}}(\mathbf{x})$ models both the dilation and the erosion of Π. To enforce the projection to reach the dilation (resp. erosion) only, we introduce a *shifting* procedures ϵ_D (resp. ϵ_E) which sends $\mathbf{x}$ outside (resp. inside) the shape to find the dilation (resp. erosion):

$$\epsilon_D(\mathbf{x}) = \mathbf{x} + (1 - \mathscr{I}_h)\ \delta(\mathbf{x}), \epsilon_E(\mathbf{x}) = \mathbf{x} + \mathscr{I}_h\ \delta(\mathbf{x}),$$

with $\delta(\mathbf{x}) = e_m\ \frac{\mathbf{c}^*(\mathbf{x}) - \mathbf{x}}{\|\mathbf{c}^*(\mathbf{x}) - \mathbf{x}\|}$, e_m the maximal distance from $\mathbf{x}$ to the bounding sphere of B in the direction $\delta(\mathbf{x})$ and $\mathscr{I}_h$ an indicator function (i.e., 0 inside, 1 outside the shape of Π) evaluated using the sign of $\mathscr{I}$ (see Fig. 6).

Based on this shifting procedure, we define a dilation (resp. erosion) projection $\mathcal{P}_D$ (resp. $\mathcal{P}_E$):

$$\mathcal{P}_D(\mathbf{x}) = \mathcal{P}_{\mathscr{B}} \circ \epsilon_D(\mathbf{x}) \quad (13)$$
$$\mathcal{P}_E(\mathbf{x}) = \mathcal{P}_{\mathscr{B}^{\dagger}} \circ \epsilon_E(\mathbf{x}) \quad (14)$$

with $\mathscr{B}^{\dagger}(\mathbf{x}) = \mathscr{B}(-\mathbf{x})$. See Alg. 1 for a pseudo-code.

Consequently, the dilation and the erosion of Π are modeled as the stationary set of the following applications:

$$D_{\Pi} : \mathbb{R}^3 \to \mathbb{R}^3, \mathbf{x} \mapsto \mathcal{P}_D(\mathbf{x}) \quad (15)$$
$$E_{\Pi} : \mathbb{R}^3 \to \mathbb{R}^3, \mathbf{x} \mapsto \mathcal{P}_E(\mathbf{x}) \quad (16)$$

It follows that we can define two converged projection operators $\mathcal{P}^{\infty}_D = \mathcal{P}_D \circ \ldots \circ \mathcal{P}_D$ (resp. $\mathcal{P}^{\infty}_E$) that directly project onto the dilation (resp. erosion). Last, we can derive implicit forms from these projective ones, similarly to PSS (see Sec 2.1).

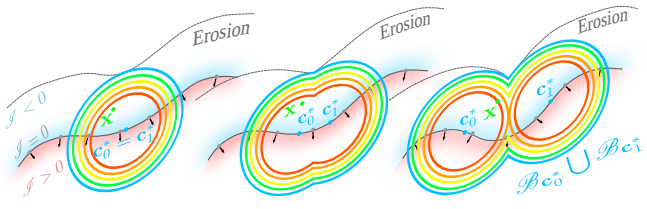


Figure 7: ***Feature modeling.*** *Eq. 11 is equivalent to a local union of primitives and captures sharp features better than running a single local optimizer (or mode).*

Algorithm 1 Dilation projection.

Input: $\mathbf{x} \in \mathbb{R}^3$, $\mathscr{B} : \mathbf{x} \mapsto \mathscr{B}(\mathbf{x})$, Π
Output: $\mathcal{P}_D(\mathbf{x}) \in \mathbb{R}^3$
  $\{\mathbf{c}^0_j\} :=$ *closest points* $\in \Pi$ *from* $\mathbf{x}$ *under* $\mathscr{B}_{\mathbf{c}}(\mathbf{x})$ *distance*
  for all j **do**
    $\mathbf{c}^*_j := MeanShift(\mathbf{x}, \mathbf{c}^0_j, \Pi)$
  end for
  $\mathbf{c}^* :=$ *closest point* $\in \{\mathbf{c}^*_j\}$ *from* $\mathbf{x}$ *under* $\mathscr{B}_{\mathbf{c}}(\mathbf{x})$ *distance*
  if $\mathscr{I}_h(\mathbf{x}) = 0$ **then**
    $\mathbf{x} := \mathbf{x} + \delta(\mathbf{x})$
  end if
  $\mathcal{P}_D(\mathbf{x}) := \mathbf{x}$ *projected onto* $\mathscr{B}_{\mathbf{c}*}$

3.5 Morphological Point Sampler

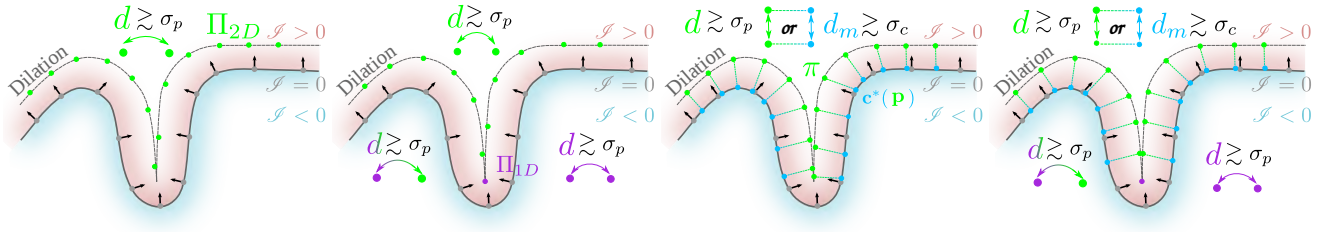
So far, we have explained how to project any point in space on the dilation or the erosion of a point cloud. Combined operators, such as openings and closings – indeed most morphological algorithms – are defined through sequences of these basic transformations. This translates to two specific constraints: the sampling of a dilation (or an erosion) should carefully capture the geometric features that emerged, as this is often the critical information raised by morphological analysis; second, this sampling should have a distribution which is suitable for the computation of a new dilation or erosion.

We tackle both issues by introducing a morphological sampler Σ. Basically, we observe that a sampling of the input surface with the blue noise property is the best condition to minimize the error between the solution of Eq. 9 and both Eq. 10 and Eq. 11. As this error typically increases around sharp features and thin parts, we adopt a two-stages sampling strategy. Starting from an initial dense sampling Π_{2D} (e.g., grid based or random based) of the dilation of Π and given σ_p the target point spacing, our sampler operates as follows:

1. we compute a feature sampling Π^*_{1D} by detecting and blue noise sampling the sharp edges of Π_{2D},
2. we compute a *morpho-adaptive* blue noise sampling Π^*_{2D} from Π_{2D}, preserving Π^*_{1D} fixed.

The final sampling is the union of the two sets:

$$\Sigma(\Pi) = \Pi^*_{2D} \bigcup \Pi^*_{1D}.$$



(a) Blue noise sampling (b) With feature preservation (c) Morpho-adaptivity (d) Both

Figure 8: ***Morphological sampling*** *influence of feature preservation and morpho-adaptivity on the blue noise distribution.*

It preserves sharp features (Π^*_{1D} is not altered by the construction of Π^*_{2D}) and is properly conditioned for any potential following operation (blue noise distribution, with increased density on thin components). See Fig. 8 for an illustration of our sampling strategy.

For a dilation/erosion by a PSE of minimum local feature size l (e.g., radius for a spherical PSE), we choose a conservative point spacing σ_p for both Π^*_{1D} and Π^*_{2D} as $\sigma_p = min(\sigma, l)/2$ where σ is set as the PSS kernel support size of the initial input surface. All subsequent transformations with Point Morphology inherit the same σ_p constraint to avoid any low pass filtering along the successive treatments. Using the same point spacing for the Π^*_{1D} and Π^*_{2D} improves stability by avoiding abrupt sampling variations.

Feature detection and distribution For each sample of Π_{2D}, we compute an optimal location $\mathbf{p}_{qem}$ as the minimizer of the quadric error metric [Garland and Heckbert 1997] in its vicinity and an optimal direction $\mathbf{d}_{qem}$ as the weakest singular vector of the QEM matrix A_{qem} [Kobbelt et al. 2001; Ohtake and Belyaev 2002]. We estimate the presence of a feature line with the ratio between the smallest singular value of A_{qem} and the two others: if it is large enough (greater than 10^3 in our implementation), we project the sample onto the line $[\mathbf{p}_{qem}, \mathbf{d}_{qem}]$ and add it to Π_{1D}. From this first set Π_{1D} we obtain a blue noise distributed set Π^*_{1D} using the method from Öztireli et al. [2010].

Morpho-adaptive distribution Dilations and erosions often create thin components (e.g., sheets, holes, branches) which require more samples to be properly modeled. To do so, we take inspiration from the sampler proposed by Öztireli et al. [2010]. This sampler adapts a blue noise distribution to the surface curvature by measuring distances between samples in 6D (positions and normals), locating more samples in highly curved regions. We adopt a similar strategy but use a 6D space which accounts for our morphological transformation: instead of using normals, we use the PSE centroids to distinguish samples that may be close in $\mathbb{R}^3$ but belong to different surface regions (e.g., two sides of a thin sheet). More precisely, we define the positional distance between two samples $\boldsymbol{\pi}_i$ and $\boldsymbol{\pi}_j$ as:

$$d(\boldsymbol{\pi}_i, \boldsymbol{\pi}_j)^2 = \frac{||\mathbf{p}_i - \mathbf{p}_j||_2^2}{\sigma_p^2} \tag{17}$$

and their the *morphological distance* as:

$$d_m(\boldsymbol{\pi}_i, \boldsymbol{\pi}_j)^2 = \frac{||\mathbf{c}^*(\mathbf{p}_i) - \mathbf{c}^*(\mathbf{p}_j)||_2^2}{\sigma_c^2} \tag{18}$$

with σ_c a scaling parameter (typically set to s).

As in [Öztireli et al. 2010], the optimal blue noise sampling is driven by a scalar value that measures the importance of a given sample $\boldsymbol{\pi}$:

$$m(\boldsymbol{\pi}) = 1 - \sum_{i,j} \tilde{w}_{ij}(\boldsymbol{\pi}) k(\boldsymbol{\pi}, \boldsymbol{\pi}_i) k^{-1}_{i,j} k(\boldsymbol{\pi}_j, \boldsymbol{\pi}) \tag{19}$$

with $k(\mathbf{u}, \mathbf{v}) = e^{-d(\mathbf{u},\mathbf{v})^2}$, $k^{-1}_{i,j}$ the elements of the inverse matrix formed by $k(\boldsymbol{\pi}_i, \boldsymbol{\pi}_j)$, $w_{ij}(\boldsymbol{\pi}) = e^{-(d_m(\boldsymbol{\pi},\boldsymbol{\pi}_i)^2 + d_m(\boldsymbol{\pi},\boldsymbol{\pi}_j)^2)}$ and $\tilde{w}_{ij}$ its normalized version.

The morphological blue noise sampling Π^*_{2D} is taken as the maximizer of $\sum_i m(\boldsymbol{\pi}_i)$ which is computed using a randomized linear scan subsampling followed by local gradient ascents [Öztireli et al. 2010] accounting for both Π_{2D} and Π^*_{1D}.

3.6 Closing and Opening

Finally, we have all the ingredients to compute closings and openings. As recalled in Sec. 2, to compute a closing C_Π (resp. an opening O_Π) of Π, we simply erode (resp. dilate) a morphological sampling of the dilation (resp. erosion) of Π. Thus we have : $C_\Pi = E_{\Sigma \circ D_\Pi}, O_\Pi = D_{\Sigma \circ E_\Pi}$.

4 Results

Simple experiments Fig. 9 show five examples of PSEs dilating the same model. Fig. 10 shows the complete set of our operators applied on a model coming from a performance capture sequence. More examples are provided as additional material.

Implementation We implemented a CPU (C++) and a GPU (CUDA) version of our framework. In Tab. 1 we report timings, measured on an Intel Core2Quad (single thread) at 2.7GHz with 8 Gb of memory and an nVidia GTX680 GPU, including measures for our four operators, diverse PSEs and several input point sets. We used a two-scale grid as the basic acceleration structure. Following Bowers et al. [2010], we only store a poisson disk subsam-

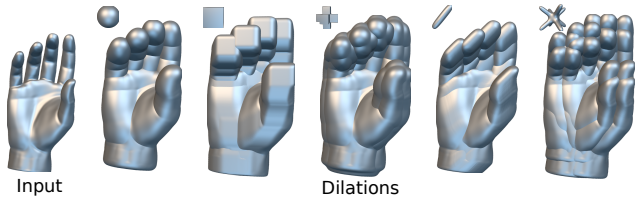


Figure 9: ***Varying the PSE shape:*** *a dilation performed with our framework for five different PSEs. The last three do not have a simple analytical form and are modeled using a PSS.*

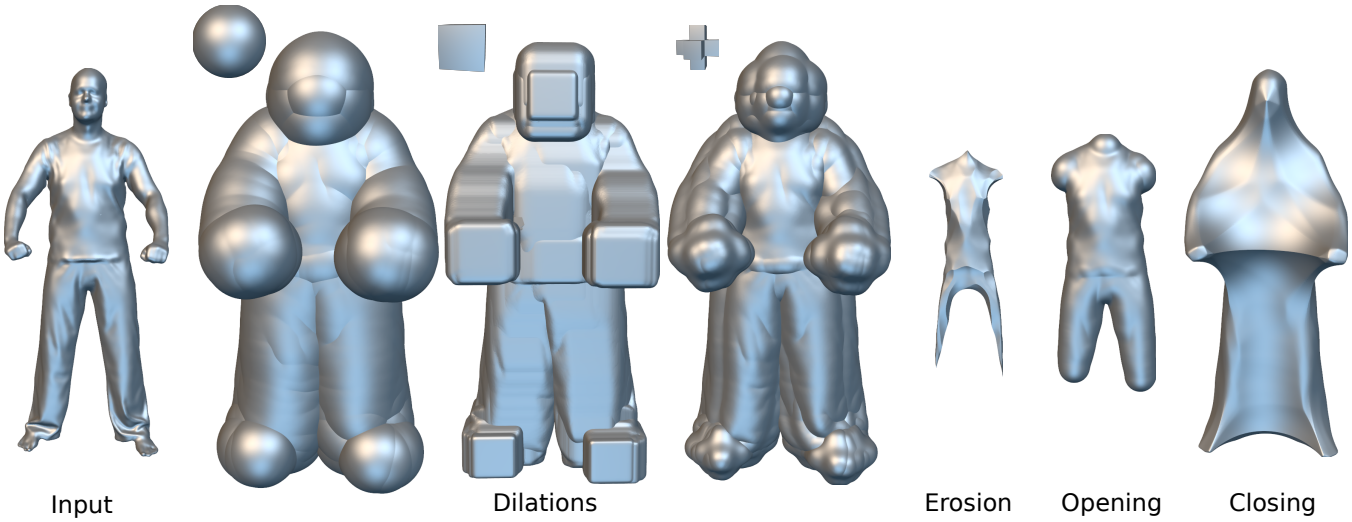


Figure 10: *Point morphology of a performance capture model (model courtesy Max Planck Institute).*

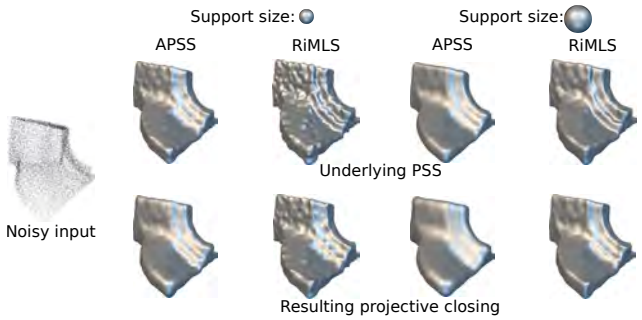


Figure 11: ***Sparse noisy input****. Starting from a sparse noisy point cloud, we compute closings by a spherical PSE, using different underlying PSS with two different support size.*

			$\mathcal{P}_{SE}$ **projections/sec**	
Model	**Nb. Pts**	**PSE**	CPU	GPU
Hand	75k	Sphere	10^4	$1.5x10^6$
		Cube	$1.5x10^3$	$0.8x10^6$
		Cross	10^3	$0.5x10^6$
Man	79k	Sphere	$8.5x10^3$	$1.5x10^6$
	79k	Cube	$0.8x10^3$	$0.8x10^6$
	79k	Cross	$0.6x10^3$	$0.5x10^6$
Buddha	255k	Sphere	$8x10^3$	$1.2x10^6$
Filgree	400k		$6x10^3$	$1.0x10^6$
Raptor	880k		$6x10^3$	$1.0x10^6$
Mammoth	1.1M		$5x10^3$	$0.9x10^6$
Neptune	1.2M		$5x10^3$	$0.8x10^6$

Table 1: ***Performance measures*** *for our morphological projection on different models illustrating this paper.*

pling of the initial point set at the first scale, the radius of this subsampling being set to $1/5$ of the SE scale. The cell size of this coarse grid structure is the scale of the SE. Once the first centroids on this coarse structure are found, we run the mean shift procedure at the second level of the grid which stores the full point set. As reported in Tab. 1, about 800k morphological projections can be computed every second for an input point cloud composed itself of a million samples. It takes typically ten iterations of this projection to reach the dilation or the erosion which means that we can sample about 100k points on them every second. This makes the design of morphological algorithms chaining multiple instances of these operators on real world data tractable (see Sec. 5). Overall, the computation of dense erosions, dilations, openings or closings never took more than a few minutes for all models presented in this paper (sampling included).

Noisy data We evaluated the behavior of our framework with noisy input. We start with a sparse noisy point set of the fandisk (see Fig. 11) which shows the influence of the underlying PSS model $\mathscr{I}$ on the resulting morphological analysis. For denser noisy input, such as the blade model (see Fig. 12), the influence of the PSS model is less critical. In both cases, we compute closings using either APSS [Guennebaud and Gross 2007] or RiMLS [Öztireli et al. 2009] as the underlying PSS model. Note that Fig. 12 exhibits a lot of fine scale Gaussian noise but also larger scale topological noise. While most of the fine scale high frequency noise is removed by both PSS models within our framework, the topological noise remains, even at large scale. We address this issue in our topological simplification application (Sec. 5.3).

Sampling Large sharp features play a major role in a shape analysis (see the sparse set of strong singular features in the closing of Fig. 10). In our morphological context, they may appear either after a single projection or progressively emerge from sequences of transformations. We address this geometric preservation problem at two stages. First, our morphological projections operators robustly model continuous transitions from smooth areas to sharp edges (see the erosion on Fig. 10). Second, our morphological sampler is instrumental here to preserve a good enough sampling all along the sequence and captures these structures without oscillations (see Fig. 13). Explicit discrete sums fail at distinguishing them from artifacts after a single iteration (see Fig. 15).

Comparisons First, we compare our approach to discrete (set) morphology on a manufactured solution: although the two kinds of input are drastically different (e.g., an unorganized point set and a voxel grid), this comparison is instructive. Given an implicit surface of a hand model, we compute a high resolution binary voxel grid

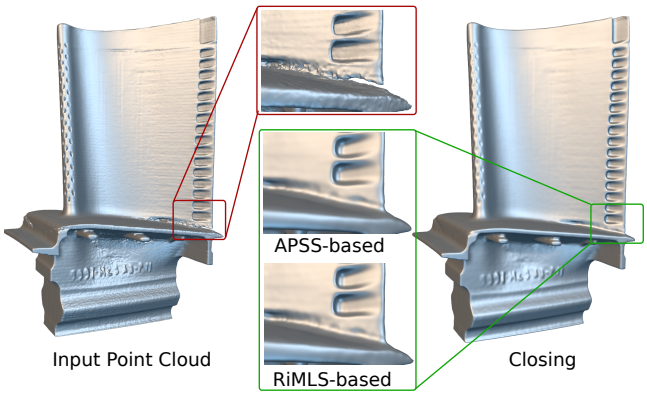


Figure 12: ***Influence of the underlying PSS model****. Although a geometric variation can be percieved, the impact of the underlying PSS model progressively vanishes with growing PSE and/or denser input.*

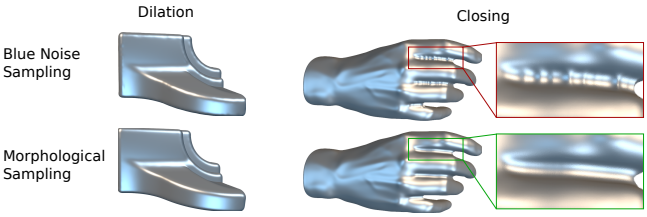


Figure 13: ***Feature-aware morphological sampling****. A blue noise sampling of the dilation of the FanDisk model (top left) exhibits oscilliations which are avoided using our 2-stage strategy (bottom left). The improvement becomes even stronger for chained operators, such as closings (right), where a small oscilliation during the first step (dilation) is amplified by the second one (erosion).*

Figure 14: ***Comparison to discrete morphology.*** *Result of a dilation using discrete (set) morphology (left) and our new point (projective) morphology (right), using either a voxelization (left) or a point sampling (right) of the same shape.*

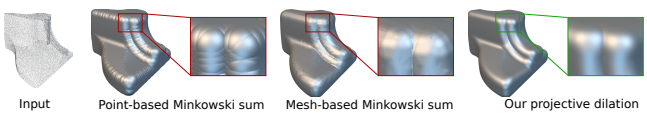


Figure 15: ***Comparison to point-based and mesh-based Minkowski sums****: our projective approach properly captures features and avoids surface oscilliations stemming from discrete schemes, even for simple shapes.*

Method	#pts	SE #pts	Output #pts/d-cover	time
Point-based	13k	13k	1.9M/0.002	300s
(Lien et al.)	13k	669	54k/0.02	8s
Mesh-based	13k	15k	121k/0.004	85s
(Campen et al.)	13k	1620	36k/0.01	26s
Ours	13k	-	2.5M/0.002	2.5s
	13k	-	515k/0.004	1s
	13k	-	87k/0.01	470ms
	13k	-	35k/0.02	350ms

Table 2: ***Performance comparison****. Timings for the FanDisk model of Fig. 15.*

(256^3) representing its interior and a point sampling of its boundary. Then, we perform two dilations: a discrete one on the grid and our projective one on the point set. Results are shown in Fig. 14. The Hausdorff distance between both dilations is smaller than the size of a voxel, which gives a practical validation of our projective scheme. See the Appendix for more formal elements. Note that voxelizing an implicit representation of the input induces a number of drawbacks: even using a high-resolution sparse data structure to store a rasterized implicit surface, the ability to represent accurately sharp features requires prohibitive computational costs and memory overhead. This is problematic not only for the initial inside/outside discretization, but also when chaining morphological transformations which typically give rise to key features that need to be preserved along the sequence.

Then, we compare our projective dilation to an explicit (point-based) Minkowski sum [Lien 2007]. In Fig. 15, we can observe that, with the FanDisk model for instance, our approach properly captures sharp features without introducing oscillations on the smooth regions, which allows sequencing the operators for higher level analysis (see Sec. 5). We reports timings in Tab. 2.

An alternative to our point-based framework would be to compute a (e.g., Poisson [Kazhdan et al. 2006]) mesh reconstruction of the point cloud before using mesh-based Minkowski sums [Campen and Kobbelt 2010] to perform morphological analysis. Although mesh-based Minkowski sums clearly target different application scenarios, it is interesting to see that our approach is almost 2 orders of magnitude faster for better visual quality compared to the method from Campen et al. [2010] (see Fig. 15 and Tab. 2).

5 Applications

Our framework allows to design a variety of shape analysis methods directly on point sets. As shown below, in spite of their high-level impact on the shape geometry or topology, these applications are very simple to implement once our operators in hand.

5.1 Projective Medial Axis

The medial axis [Amenta et al. 2001], an important tool in geometry modeling, is used to characterized both the geometry and the topology of a shape. When this shape is modeled with a point set, the computation of the medial axis usually relies, in one way or another, on the meshing of the set followed by a medial-axis transform. Our framework allows to sample it directly from the input point cloud. Indeed, by definition, all the singularities emerging from an erosion with a spherical PSE at scale t are located on the medial axis. Our erosion operator reaches these singularities as the local intersection of several PSEs. Consequently, when growing the scale of the PSE, the locii of theses singularities sweep a piecewise smooth surface which approximates the medial axis accurately.

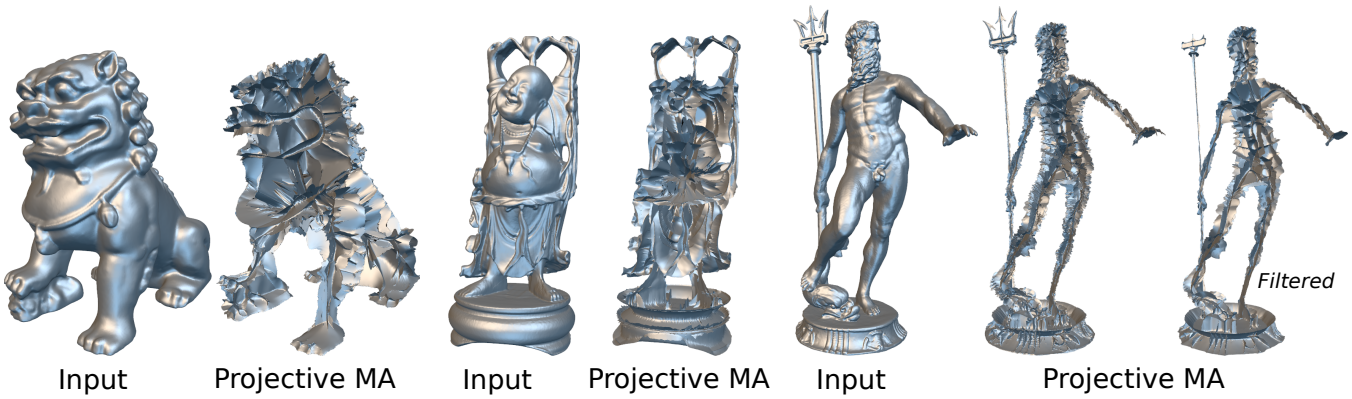


Figure 16: ***Projective medial axis sampling***. *For the Neptune model, we present both the full res. medial axis and a filtering based on a preliminary hysteresis shape filtering (see Sec. 5.2).*

	PowerCrust		Ours		
Models	*Time*	*Mem.*	*Time*	*Mem.*	RMS
Neptune	840s	21Gb	92s	1.5Gb	3.3x1e-4
Filigree	196	7.6Gb	69s	1.2Gb	2.5x1e-4
Oil Pump	162s	6.7Gb	66s	1.2Gb	1.2x1e-3

Table 3: *Medial Axis : comparison with powercrust. The RMS error is expressed w.r.t. to the input model bouding box diagonal.*

Algorithm We sample the medial axis of Π using a set of points M on $\mathscr{I}_\Pi$ and projecting each point $\boldsymbol{\pi}_j \in M$ in three steps. First, we compute t_j the *local feature size* as the radius of the minimal sphere tangent to $\boldsymbol{\pi}_j$ and touching a point of Π:

$$t_j = \min_{\boldsymbol{\pi}_i \in \Pi} \frac{1}{2} \frac{\|\mathbf{p}_j - \mathbf{p}_i\|^2}{(\mathbf{p}_j - \mathbf{p}_i) \cdot \mathbf{n}_j} \tag{20}$$

Secondly, we perform a rough medial axis projection by pushing $\boldsymbol{\pi}_j$ along its normal such that $\mathbf{m}_j = \mathbf{p}_j - t_j \cdot \mathbf{n}_j$. Third, we project $\mathbf{m}_j$ onto the *singularities* of $\mathcal{P}_E$ with a spherical PSE of scale t_j. To do so, we analyze the mean shift's modes distribution used to fit the PSE (see Sec 3 and Eq. 10). If we detect a single mode, the sample is discarded. Otherwise, we iteratively project it on the intersection of the local PSE modes and output the resulting location.

This projective approach to the computation of the medial axis does not require any intermediate mesh and allows to densely sample the medial axis directly from the input cloud (see Fig. 16). However, if a meshed medial axis is required in the application scenario, we generate M as the vertex set of a polygonization of Π and keep the so-defined connectivity during the projection. In practice, we use a marching cube meshing a PSS defined from Π.

Comparison We compare our projective medial axis sampling to the PowerCrust algorithm [Amenta et al. 2001], which is a popular method to compute a medial axis from a point set. It is based on the 3D Voronoï tessellation of the input set and outputs a mesh representing the medial axis.

In terms of quality, the PowerCrust generates a noisier medial axis compared to ours, even on nearly perfect input (see on Fig. 17, top) and is less robust to incomplete point clouds with large missing regions (see on Fig. 17, bottom). Concerning performance, we report time and memory measures in Tab. 3: our projective approach is about one order of magnitude faster, requiring up to one order of magnitude less memory. The distance between both medial axes is also negligible (excluding the case of incomplete point clouds). We observed a similarly good approximation (RMS error below 1e-4 of the bounding box diagonal) when comparing to a union-of-balls medial axis such as used as input by Miklos et al. [2010].

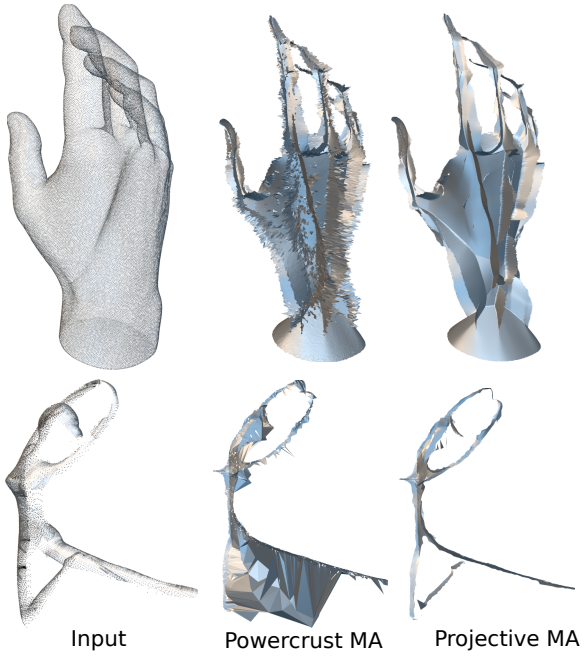


Figure 17: ***Medial axis comparison to Powercrust****, with a high quality input point cloud (top) and an incomplete one coming from a range scan (partial input, bottom).*

5.2 Hysteresis Shape Filtering

Linear filters are efficient at removing small scale geometric features from surfaces. However, for larger structures, they often fail at doing so without severely damaging the rest of the object. With point morphology, we can selectively remove structures of a given size while preserving a rich signal everywhere else by simply stringing together (i) a closing by a PSE of size s_C and (ii) an opening by a PSE of size s_O: $O_\Pi \circ C_\Pi$. Intuitively, this corresponds to an hysteresis process which "carves" convex parts smaller than s_O and "fills" concave ones smaller than s_C.

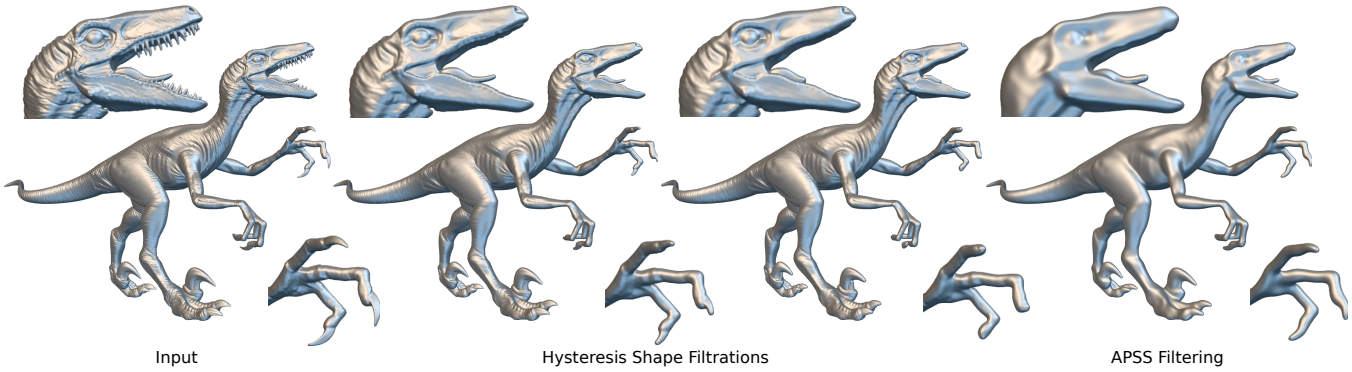


Figure 18: ***Hysteresis shape filtering***. *At small scale (middle left, $s_C = 0$ and $s_O = 0.004$) only the teeth are removed. Increasing s_O to 0.006 (middle right), claws are removed. The APSS filtering (right) is performed with the smallest support removing the raptor's teeth.*

We show in Fig. 18, that this filtering method removes the teeth of the Raptor when using a small value of s_O while preserving the rest of the shape. Increasing the hysteresis threshold ($s_O = 0.006$), the claws of its forelegs disappear. When applied prior to our projective medial axis sampling (see Sec. 5.1), such as with the Neptune model (see Fig. 16 right, computed with $s_C = s_O = 0.008$), this hysteresis process acts as a medial axis filtering [Miklos et al. 2010]. Another example is shown with the Mammoth model (see Fig. 1) where we used a closing of size $s_C = 0.1$ and an opening size $s_O = 0.01$. As a result, almost all its ribs are removed, preserving all the rest of its bone structure (4 legs, a head, a tail and horns).

5.3 Geometry Preserving Topological Simplification

Finally, beyond geometric structure removal, point clouds may implicitly contain a number of topological defects, with numerous unwanted tunnels and handles revealed in the forthcoming stages of the pipeline (e.g., meshing, rendering). Usually this issue is solved by either strongly low-pass filtering the point set before reconstruction or by manually editing it. Our framework provides a direct solution to this problem. Indeed, a closing by a small spherical PSE naturally fills these tunnels and handles while preserving the fine details in the other regions of the surface. We illustrate this effect on the Filigree model (see Fig. 19, top): this model has a high genus and applying a PSS interpolation with large support to reach a simpler topology loses most of the on-surface signal. On the contrary, closing it with our framework, even with a large PSE, retains a significant part of this signal, all the way down to genus 0. In Fig. 19 (bottom), we process a CT scan model of a *skull* with a high genus. Using a small PSE (0.01), we reduce the genus from 520 to 47: this removes small tunnels but preserves larger topological structures as well as the geometric texture of the input. In comparison, an APSS interpolation – even with a much bigger support – only simplifies to genus 68 and again over smooths the entire shape. Lastly, a feature-preserving RiMLS interpolation is stuck to genus 78 at similar scale, and still loses much more information than our morphological approach.

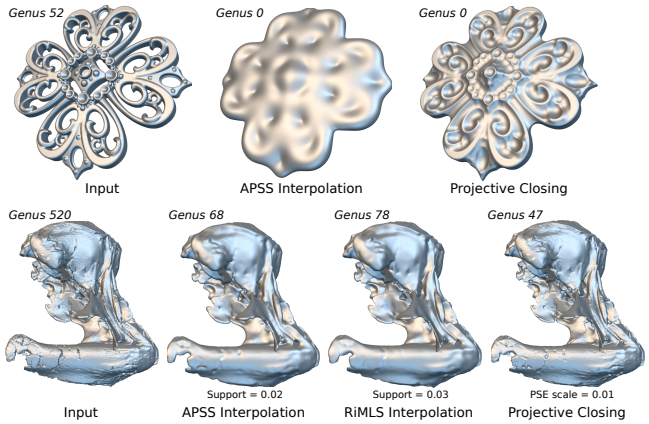


Figure 19: ***Geometry-preserving topological simplification***. *Top: an extreme simplification to sphere topology. For both APSS and closing we used the miminal radius to reach genus 0. Bottom: topology cleaning of a skull scan.*

6 Discussion

Limitations and Future Work. There are several limitations with our current pipeline which open potential research directions. First, we use a PSS as the underlying surface model of our framework. Although efficient to compute, PSS remain local solutions and are sensitive to structured outliers and poor input sampling conditions. An interesting direction for future work would be to use a better inside/outside classification technique [Jacobson et al. 2013] and account robustly for outliers [Lipman et al. 2007]. Second, if a connectivity is provided with the input point set, our projective approach is currently blind to it. Accounting for this information, even partially, would be useful for some applications scenarios. Third, although our framework supports a great variety of structuring elements, the applications we proposed are focused on the spherical case. Alternative PSEs, such as the cubic one for instance, could be instrumental for tight bounding volume computation, polycube generation or volume meshing. Interestingly, the form of our structuring element allows for spatial variability (see Fig. 20 for such an experiment), opening a potential direction toward spatially-varying morphological analysis. Last, analyzing the optimal sampling conditions at this stage is an interesting direction for future work and defining these transformations without any intermediate sampling step an even more exciting problem.

Conclusion. We have proposed a complete framework for the morphological analysis on point clouds. By introducing a new model for the structuring element and substituting the Minkowski sum with a new projection procedure, we can robustly explore the dilation and erosion of the input sampled shape in a completely meshless context. Using our morpho-adaptive sampler then allows to compute sequences of morphological alterations, in particular openings and closings, revealing the singular structures of the

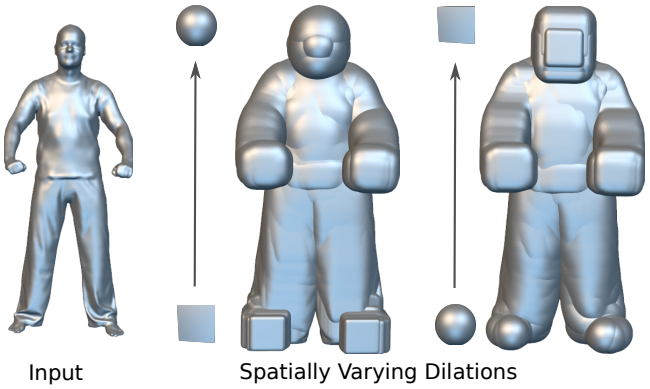


Figure 20: ***Spatially Varying PSE****. The variational nature of our structuring element allows to continuously morph its shape and open the way for spatially varying morphological analysis.*

point set. Based on this framework, we have proposed three new applications: a projective approach to the direct sampling of the medial axis, a controllable mechanism for selective shape filtering by hysteresis and a geometry-preserving topological simplification method. Although clearly non trivial, these applications boil down to simple sequences of our operators.

Acknowledgements We thank the anonymous reviewers for their suggestions and Isabelle Bloch for her advices. This work has been partially funded by the European Commission under contracts FP7-323567 HARVEST4D and FP7-287723 REVERIE, and by the ANR iSpace&Time project.

References

ADAMSON, A., AND ALEXA, M. 2004. Approximating bounded, non-orientable surfaces from points. In *Proceedings of the Shape Modeling International 2004*, IEEE Computer Society, Washington, DC, USA, SMI '04, 243–252.

ADAMSON, A., AND ALEXA, M. 2006. Point-sampled cell complexes. *ACM Trans. Graph. 25*, 3, 671–680.

ALEXA, M., AND ADAMSON, A. 2009. Interpolatory point set surfaces—convexity and hermite data. *ACM Trans. Graph. 28*, 2 (May), 20:1–20:10.

ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. T. 2001. Point set surfaces. In *Proceedings of the conference on Visualization '01*, IEEE Computer Society, Washington, DC, USA, VIS '01, 21–28.

ALEXA, M., RUSINKIEWICZ, S., ALEXA, M., AND ADAMSON, A. 2004. On normals and projection operators for surfaces defined by point sets. In *In Eurographics Symp. on Point-Based Graphics*, 149–155.

AMENTA, N., AND KIL, Y. J. 2004. Defining point-set surfaces. In *ACM SIGGRAPH 2004 Papers*, ACM, New York, NY, USA, SIGGRAPH '04, 264–270.

AMENTA, N., CHOI, S., AND KOLLURI, K. 2001. The power crust. In *6th ACM Symposium on Solid Modeling*, 249–260.

BABAUD, J., WITKIN, A. P., BAUDIN, M., AND DUDA, R. O. 1986. Uniqueness of the gaussian kernel for scale-space filtering. *IEEE Trans. Pattern Anal. Mach. Intell. 8*, 1 (Jan.), 26–33.

BARKI, H., DENIS, F., AND DUPONT, F. 2011. Contributing vertices-based minkowski sum of a nonconvex–convex pair of polyhedra. *ACM Trans. Graph. 30* (Feb.), 3:1–3:16.

BOWERS, J., WANG, R., WEI, L.-Y., AND MALETZ, D. 2010. Parallel poisson disk sampling with spectrum analysis on surfaces. In *ACM SIGGRAPH Asia 2010 papers*, ACM, New York, NY, USA, SIGGRAPH ASIA '10, 166:1–166:10.

CAMPEN, M., AND KOBBELT, L. 2010. Polygonal boundary evaluation of minkowski sums and swept volumes. *Computer Graphics Forum 29*, 5, 1613–1622.

CHEN, Y., WANG, H., W. ROSEN, D., AND ROSSIGNAC, J. 2005. A point-based offsetting method of polygonal meshes. Tech. rep.

CHENG, Y. 1995. Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intell. 17*, 8 (Aug.), 790–799.

FLEISHMAN, S., COHEN-OR, D., AND SILVA, C. T. 2005. Robust moving least-squares fitting with sharp features. *ACM Trans. Graph. 24*, 3, 544–552.

FUKUNAGA, K., AND HOSTETLER, L. D. 1975. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory 21*, 1, 32–40.

GARLAND, M., AND HECKBERT, P. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 209–216.

GUENNEBAUD, G., AND GROSS, M. 2007. Algebraic point set surfaces. In *ACM SIGGRAPH 2007 papers*, ACM, New York, NY, USA, SIGGRAPH '07.

JACOBSON, A., KAVAN, L., AND SORKINE-HORNUNG, O. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph. 32*, 4, 33:1–33:12.

KAZHDAN, M., BOLITHO, M., AND HOPPE, H. 2006. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SGP '06, 61–70.

KOBBELT, L. P., BOTSCH, M., SCHWANECKE, U., AND SEIDEL, H.-P. 2001. Feature sensitive surface extraction from volume data. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 57–66.

KOLLURI, R. 2008. Provably good moving least squares. *ACM Trans. Algorithms 4*, 2 (May), 18:1–18:25.

LEVIN, D. 1998. The approximation power of moving least-squares. *Mathematics of Computation 67*, 1517–1531.

LEVIN, D. 2003. Mesh-independent surface interpolation. *Geometric Modeling for Scientific Visualization 3*, 37–49.

LIEN, J.-M. 2007. Point-based minkowski sum boundary. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, USA, 261–270.

LIPMAN, Y., COHEN-OR, D., LEVIN, D., AND TAL-EZER, H. 2007. Parameterization-free projection for geometry reconstruction. *ACM Trans. Graph. 26*, 3.

MELLADO, N., BARLA, P., GUENNEBAUD, G., REUTER, P., AND SCHLICK, C. 2012. Growing least squares for the contin-

uous analysis of manifolds in scale-space. *Computer Graphics Forum* (July).

MIKLOS, B., GIESEN, J., AND PAULY, M. 2010. Discrete scale axis representations for 3d geometry. In *ACM SIGGRAPH 2010 Papers*, ACM, New York, NY, USA, SIGGRAPH '10, 101:1–101:10.

MOLCHANOV, V., ROSENTHAL, P., AND LINSEN, L. 2010. Non-iterative second-order approximation of signed distance functions for any isosurface representation. *Computer Graphics Forum 29*, 3, 1211–1220.

NAJMAN, L., AND TALBOT, H., Eds. 2010. *Mathematical Morphology: From Theory to Applications*. Wiley.

NELATURI, S., AND SHAPIRO, V. 2009. Configuration products in geometric modeling. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, ACM, New York, NY, USA, SPM '09, 247–258.

OHTAKE, Y., AND BELYAEV, A. G. 2002. Dual/primal mesh optimization for polygonized implicit surfaces. In *Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications*, ACM, New York, NY, USA, SMA '02, 171–178.

ÖZTIRELI, C., GUENNEBAUD, G., AND GROSS, M. 2009. Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum 28*, 2, 493–501.

ÖZTIRELI, A. C., ALEXA, M., AND GROSS, M. 2010. Spectral sampling of manifolds. In *ACM SIGGRAPH Asia 2010 papers*, ACM, New York, NY, USA, SIGGRAPH ASIA '10, 168:1–168:8.

PAULY, M., KOBBELT, L. P., AND GROSS, M. 2006. Point-based multiscale surface representation. *ACM Trans. Graph. 25* (April), 177–193.

PETERNELL, M., AND STEINER, T. 2007. Minkowski sum boundary surfaces of 3d-objects. *Graph. Models 69*, 3-4 (May), 180–190.

REUTER, P., JOYOT, P., TRUNZLER, J., BOUBEKEUR, T., AND SCHLICK, C. 2005. Surface reconstruction with enriched reproducing kernel particle approximation. In *IEEE/Eurographics Symposium on Point-Based Graphics*, Eurographics/IEEE Computer Society, 79–87.

SERRA, J. 1983. *Image Analysis and Mathematical Morphology*. Academic Press, Inc., Orlando, FL, USA.

A Appendix

We derive a variational formulation of Mathematical Morphology and show that our projective approach (Sec 3) is an approximation of this variational formulation. Note that all the proofs in Sec. A.2, A.4, A.6, A.7 are provided as additional materials. In the following an input shape I is defined as a 3-manifold compact subset of $\mathbb{R}^3$. First we recall the classical Mathematical Morphology which is based on set theory.

A.1 Set Morphology

Set Structuring Element. *A structuring element B is defined as follows:*

$$B \subset \mathbb{R}^3,\ \mathbf{0} \in B,\ B \text{ is compact and connected} \tag{21}$$

And $B^\dagger$ is defined as the symmetric of B w.r.t $\mathbf{0}$. The translated SE $B_\mathbf{c}$ with $\mathbf{c} \in \mathbb{R}^3$ is defined as:

$$B_\mathbf{c} = \{\mathbf{b} + \mathbf{c} | \mathbf{b} \in B\} \tag{22}$$

Set Morphology. *Given an input shape I and a SE B, the set Dilation is defined as:*

$$D_{I,B} = \bigcup_{\mathbf{c} \in I} B_\mathbf{c} \tag{23}$$

The boundary associated with this set Dilation is defined from a topological point of view as:

$$\partial D_{I,B} = \{\mathbf{x} \in \mathbb{R}^3,\ \forall r\ \exists\ (\hat{\mathbf{u}}, \check{\mathbf{u}}) \in \mathcal{N}^r_\mathbf{x} \mid \hat{\mathbf{u}} \in D_{I,B},\ \check{\mathbf{u}} \notin D_{I,B}\} \tag{24}$$

A.2 Set Boundary Equivalence

Equivalence Theorem. *Given an input shape I and a SE B we have:*

$$D_{I,B} = \bigcup_{\mathbf{c} \in I} B_\mathbf{c} = \bigcup_{\mathbf{c} \in \partial I} B_\mathbf{c} \cup I \tag{25}$$

A.3 Variational Morphology

We define a variational formulation of the set morphology.

Variational Subset of $\mathbb{R}^3$. *Given a compact subset B of $\mathbb{R}^3$, we define its variational representation as a (at least) C^0 scalar field $\mathscr{B} : \mathbb{R}^3 \to \mathbb{R}$ such as:*

$$\mathscr{B}(\mathbf{x}) = \begin{cases} < 0, & \text{if } \mathbf{x} \in \mathring{B} \\ 0, & \text{if } \mathbf{x} \in \partial B \\ > 0, & \text{if } \mathbf{x} \notin B \end{cases} \tag{26}$$

Variational Structuring Element. *Given a SE B we define its variational SE representation as the variational representation $\mathscr{B}$ of B. We define a translated variational SE $\mathscr{B}_\mathbf{c}$ as:*

$$\mathscr{B}_\mathbf{c} : \mathbb{R}^3 \to \mathbb{R}, \mathbf{x} \to \mathscr{B}(\mathbf{x} - \mathbf{c}) \tag{27}$$

Variational Morphology. *Given an input shape I and B a SE with its variational SE representation $\mathscr{B}$, we define a variational Dilation as:*

$$D_{I,\mathscr{B}}(\mathbf{x}) = \min_{\mathbf{c} \in I} \mathscr{B}_\mathbf{c}(\mathbf{x}) \tag{28}$$

The boundary associated with this variational Dilation is defined as:

$$\partial D_{I,\mathscr{B}} = \{\mathbf{x} \mid D_{I,\mathscr{B}}(\mathbf{x}) = 0\} \tag{29}$$

A.4 Set and Variational Formulation Equivalence

Now, we link set and variational morphologies in the form of an equality between the boundaries produced by both formulations.

Equivalence Theorem. *Given an input shape I and B, a SE with its variational SE representation $\mathscr{B}$, we have:*

$$\partial D_{I,B} = \partial D_{I,\mathscr{B}} \tag{30}$$

A.5 Variational Boundary Morphology

Variational Boundary Morphology. *Given an input shape I with its variational representation $\mathscr{I}$ and B a SE with its variational representation $\mathscr{B}$, we define (with $\wedge$ as the binary min) a variational boundary Dilation as:*

$$D_{\mathscr{I},\mathscr{B}}(\mathbf{x}) = \min_{\substack{\mathbf{c}\in\mathbb{R}^3 \\ \mathscr{I}(\mathbf{c})=0}} \mathscr{B}_{\mathbf{c}}(\mathbf{x}) \ \wedge \ \mathscr{I}(\mathbf{x}) \tag{31}$$

The boundary associated with this variational boundary Dilation is defined as:

$$\partial D_{\mathscr{I},\mathscr{B}} = \{\mathbf{x} \mid D_{\mathscr{I},\mathscr{B}}(\mathbf{x}) = 0\} \tag{32}$$

A.6 Set and Variational Boundary Formulation Equivalence

Now, we can show, similarly to variational morphology, but using the set boundary formulation as a basis, the same equivalence:

Equivalence Theorem. *Given an input shape I with its variational representation $\mathscr{I}$ and B a SE with its variational representation $\mathscr{B}$ we have:*

$$\partial D_{I,B} = \partial D_{\mathscr{I},\mathscr{B}} \tag{33}$$

A.7 Projective Morphology

Given an input shape I with its variational representation $\mathscr{I}$ and B a SE with its variational representation $\mathscr{B}$ we define a projection operator to reach $\partial D_{\mathscr{I},\mathscr{B}}$:

$$\mathcal{P}_{\mathscr{B}}(\mathbf{x}) = \mathbf{x} - \mathscr{B}_{\mathbf{c}*}(\mathbf{x}) \frac{\nabla \mathscr{B}_{\mathbf{c}*}(\mathbf{x})}{\|\nabla \mathscr{B}_{\mathbf{c}*}(\mathbf{x})\|} \tag{34}$$

$$\mathbf{c}^* = \operatorname*{argmin}_{\substack{\mathbf{c}\in\mathbb{R}^3 \\ \mathscr{I}(\mathbf{c})=0}} \mathscr{B}_{\mathbf{c}}(\mathbf{x}) \tag{35}$$

We can show that using the same definitions from Sec. 3.4 for $\mathcal{P}_D^{\infty}$, but using an optimized centroid $\mathbf{c}^*$ defined as the exact solution of Eq. 9 or Eq. 35, we can reach the actual Dilation $\partial D_{\mathscr{I},\mathscr{B}}$:

Projection Theorem. *For $\mathbf{x} \in \mathbb{R}^3$:*

$$\mathcal{P}_D^{\infty}(\mathbf{x}) \in \partial D_{\mathscr{I},\mathscr{B}} \tag{36}$$

The same holds for the erosion.

A.8 Point Morphology as a Sampled Projective Morphology

We can think of our morphological centroid as a sampled approximation of the projective morphology. We aim at reformulating Eq. 35 by a kernel density estimation of this global optimization problem with non linear constraints. We tackle this global optimization using the *mean shift* algorithm [Cheng 1995] on a sampling of its objective function. Thus, we replace Eq. 35 by:

$$\mathbf{c}^* = \operatorname*{argmin}_{\substack{\mathbf{c}\in\mathbb{R}^3 \\ \mathscr{I}(\mathbf{p}_i)=0}} \sum_i (\mathscr{B}_{\mathbf{p}_i}(\mathbf{x}) + \gamma)\omega_\sigma\left(\|\mathbf{c}-\mathbf{p}_i\|_2\right) \tag{37}$$

This equation is a simple reformulation of Eq. 35 where the objective function $\mathscr{B}_{\mathbf{c}}(\mathbf{x})$ and the constraint $\mathscr{I}(\mathbf{c}) = 0$ are replaced by a new objective function based on weighted kernel density estimation. The constraint is replaced by kernel density samples, and the objective function by weights on theses samples. The global offset $\gamma = \min_{\mathbf{x}\in\mathbb{R}^3} \mathscr{B}(\mathbf{x})$ ensures the positiveness of the weights, and as such makes the objective a proper density. We found that using Gaussian kernels also for the weights improves stability. Additionally this also transform the initial minimization equation into the following maximization problem:

$$\mathbf{c}^* = \operatorname*{argmax}_{\substack{\mathbf{c}\in\mathbb{R}^3 \\ \mathscr{I}(\mathbf{p}_i)=0}} \sum_i \omega_\sigma\left(\mathscr{B}_{\mathbf{p}_i}(\mathbf{x}) + \gamma\right)\omega_\sigma\left(\|\mathbf{c}-\mathbf{p}_i\|_2\right) \tag{38}$$

As a final step we instantiate the surface model $\mathscr{I}$ by the implicit form of a PSS model The new objective function of Eq. 38 can be maximized through the *mean shift* procedure [Fukunaga and Hostetler 1975; Cheng 1995].

$$\mathbf{c}^k(\mathbf{x}) = \sum_i \omega_\sigma(\|\mathbf{c}^{k-1}(\mathbf{x}) - \mathbf{p}_i\|)\, \omega_\sigma(\mathscr{B}_{\mathbf{p}_i}(\mathbf{x}))\mathbf{p}_i \tag{39}$$

$$\forall \boldsymbol{\pi}_i \in \Pi, \quad \mathscr{I}(\mathbf{p}_i) = 0 \tag{40}$$

A.9 Normals of Point Morphology

The normals of the morphological surfaces are computed by taking the gradient of their implicit forms:

$$\mathbf{n}(\mathbf{x}) = \nabla \mathscr{B}_{\mathbf{c}^*(\mathbf{x})}(\mathbf{x}) = \nabla \mathscr{B}_{\mathbf{c}^*}(\mathbf{x}) \nabla \mathbf{c}^*(\mathbf{x}) \tag{41}$$

We compute $\nabla \mathbf{c}^*(\mathbf{x})$ recursively through Eq. 39:

$$\nabla \mathbf{c}^k = \sum_i \omega_i^{k-1} \mathbf{p}_i \nabla \theta_i^{k-1} - \mathbf{c}^k \sum_i \omega_i^{k-1} \nabla \theta_i^{k-1} \tag{42}$$

$$\omega_i^{k-1} = \omega_\sigma(\|\mathbf{c}^{k-1} - \mathbf{p}_i\|)\, \omega_\sigma(\mathscr{B}_{\mathbf{p}_i}) \tag{43}$$

$$\nabla \theta_i^{k-1} = -\frac{2}{\sigma^2}((\mathbf{c}^{k-1} - \mathbf{p}_i)^T \nabla \mathbf{c}^{k-1} + \mathscr{B}_{\mathbf{p}_i} \nabla \mathscr{B}_{\mathbf{p}_i}) \tag{44}$$

Efficient Collision Detection While Rendering Dynamic Point Clouds

Mohamed Radwan* Stefan Ohrhallinger† Michael Wimmer‡

Vienna University of Technology, Austria

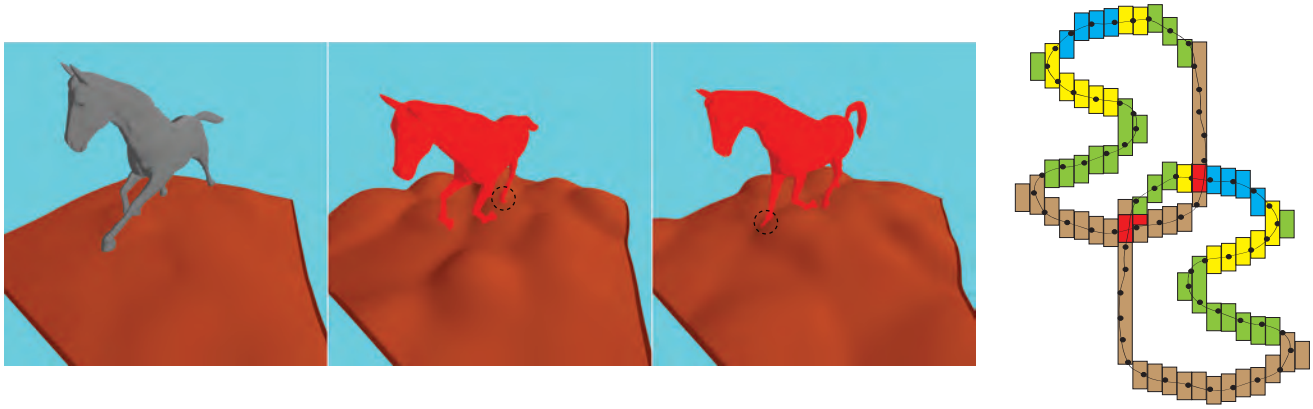


Figure 1: Left: Subsequent snapshots of an animated HORSE traversing continuously oscillating ground. Collisions between those two dynamic point clouds are marked by circles and HORSE is shaded red. Right: TLDIs of two objects. Colliding extents are shaded red.

ABSTRACT

A recent trend in interactive environments is the use of unstructured and temporally varying point clouds. This is driven by both affordable depth cameras and augmented reality simulations. One research question is how to perform collision detection on such point clouds. State-of-the-art methods for collision detection create a spatial hierarchy in order to capture dynamic point cloud surfaces, but they require $O(NlogN)$ time for N points. We propose a novel screen-space representation for point clouds which exploits the property of the underlying surface being 2D. In order for *dimensionality reduction*, a 3D point cloud is converted into a series of *thickened layered depth images*. This data structure can be constructed in $O(N)$ time and allows for *fast surface queries* due to its increased compactness and memory coherency. On top of that, parts of its construction come for free since they are already handled by the rendering pipeline. As an application we demonstrate online collision detection between dynamic point clouds. It shows *superior accuracy* when compared to other methods and *robustness to sensor noise* since uncertainty is hidden by the thickened boundary.

Index Terms: Computer Graphics [I.3.5]: Computational Geometry and Object Modeling—Hierarchy and Geometric Transformations Image Processing and Computer Vision [I.4.8]: Scene Analysis—Surface Fitting

1 INTRODUCTION

This paper proposes a novel accelerated approach for constructing and querying the underlying surface of dynamic point clouds. When those point clouds are rendered, calculations from the point-based rendering (PBR) pipeline are reused in the surface construction for the points inside the view frustum.

Collision detection requires determining the distance from the shape boundary of the object. For point clouds, especially noisy ones, reconstructing the surface as a triangulated mesh is a tedious process which currently is not feasible to do online. Applications where collision detection between dynamic point clouds is relevant include moving and posing of objects, as well as touch and grip. Such point clouds are dynamically changing environments, e.g., acquired by sensors attached to drones in a disaster scenario as simultaneous location and mapping (SLAM), human avatars captured by a Kinect, or deforming virtual objects for augmented-reality applications. Physically remote point clouds may be transposed into a common coordinate system to allow for interaction. Finally, user interaction can lead to non-rigid deformation or fragmentation.

We target medium-to-large and possibly noisy point clouds which are dynamic in the sense of having little or no temporal coherence. Constructing a spatial hierarchy for geometry, e.g., bounding volume hierarchies (BVH) [12], or tree structures [20], adds a logarithmic time factor to collision processing with respect to the number of handled points. This setup time is amortized only for static point clouds. Using a BVH allows for deformations and local rigid transformations, but not for entirely dynamic point sets. With interactive applications, the interest is often concentrated inside the view frustum, since it determines what the viewer can see and manipulate.

Our main goal is to enable online processing of medium-to-large dynamic point clouds without temporal coherence, such as Kinect input. We achieve this by avoiding construction of spatial hierarchies altogether and instead discretize the surface underlying the points into a screen-space grid. This two-dimensional structure reduces the dimensionality of the grid and thus results in more compact storage and faster intersection testing. The advantages of using a grid remain, namely that construction and evaluation can be parallelized well on the GPU.

*e-mail: radwan@cg.tuwien.ac.at
†e-mail:ohrhallinger@cg.tuwien.ac.at
‡e-mail:wimmer@cg.tuwien.ac.at

Our contributions are:

- *Efficient reconstruction of connectivity* for point clouds where an estimation of local sampling density is available, even in the presence of noise.
- *Compact boundary discretization* of point clouds by extending layered depth images with range, adapted to screen space.
- *Reuse of parts of the rendering pipeline* for constructing the boundary data structure for the point cloud.
- *Precise and online collision detection of dynamic point clouds* as an example application for surface distance queries.

2 Related Work

Bounding volume hierarchies (BVHs). These are spatial object representation structures that have been widely used in many applications. Different volume types are used to bound the geometric primitives, such as AABBs [2], OBBs [8], DOPs [13], and convex hulls [5]. BVHs are efficient in processing proximity queries, with $O(logN)$ time. Their construction of $O(NlogN)$ is also considered efficient, since for static objects the structure is constructed only once at set up. However, updating a BVH of an entirely dynamic data set is also of $O(NlogN)$ time. Therefore, for data sets with continuous temporal updates as we consider in this paper, BVHs suffer from inefficiency, whether they are updated or constructed from the start with every update.

Voxelization. Our work is related to scene voxelization approaches. Eisemann and Decoret [6] utilized the capabilities of the GPU to construct voxel-based representations which need not be aligned in one (i.e., the view) axis but are restricted to a fixed number of constant-size intervals, while Hinks et al. [11] use a similar representation to construct solid models for computational modeling. An older approach [4] also reconstructs a sampled surface implicitly at grid cells using a signed distance function. We compute discrete screen-space aligned layers instead, each layer represented by non-aligned depth ranges.

Image-based techniques for collision detection. Such techniques [14, 9, 10] do not require any pre-processing, and thus are appropriate for dynamically deforming objects. In [10], layered depth images (LDI) are computed for both objects, then volume representations are constructed and compared to find intersection regions. The algorithm, and almost all image-based collision detection (CD) algorithms as well, targets triangulated meshes. To our knowledge, only one approach [1] uses image space to detect collision between point clouds, but is restricted to movement in 2.5D space. They divide the space into slices and compute a height map for each. The approach assumes that obstacles are nearly parallel to YZ plane and perpendicular to XY plane, and uses these assumptions to infer obstacle information and save them with each pixel.

Static point-cloud collision detection. The most important approach [12] is both robust to noise and fast (interactive if need be, depending on the time budget). They construct a BVH and use a collision probability measure between pairs of nodes, in order to traverse the two objects' hierarchies ordered by priority. In the second stage, they sample the implicit surface at the leaf nodes to measure separation distance. However, as mentioned before, construction of a BVH is slow and can be memory-intensive, and the entire data needs to reside in memory as well. Thus it is impractical for large point sets and even detrimental to build such a structure for points sets which change dynamically and are not queried often enough to amortize its building cost. Our approach targets different application scenarios, but it surpasses the accuracy achieved by Klein's algorithm [12], as is shown in Section 7. Pan et al. [16] robustly detect collisions between noisy point clouds by defining the detection as a two-class classification problem and estimate collision probability with support vector machines, but runtime is comparatively slow. Our approach hides sensor noise by querying a thickened boundary.

Dynamic point-cloud collision detection. A very recent paper [17] uses BVHs and/or octrees to detect collisions and compute distances between sensor-captured point clouds. They propose two ideas, one is appropriate for static environments and the other for dynamic ones. For dynamic environments, they propose to mutually traverse an octree (environment point cloud) and an AABB (robot), and do an unspecified, probably simple collision test at leaf nodes. Although this approach for dynamic environments is simple, we avoid building a spatial hierarchy at all and can keep the GPU pipeline more occupied by streaming coherent data.

3 Overview

Our input data are *unstructured points*. We assume that the rendering pipeline has already culled points against the view frustum (or respective bounding box) and projected them into screen space. Further we assume that the *sampling density* for the individual input points is given, either globally uniform or, e.g., estimated from sensor device properties.

In Section 4 we define a *thickened boundary* which envelops the implicit surface of the point cloud, provided that the sampling is sufficiently dense. We then transform it into projected space and finally discretize it in screen space, adapted to the view point. Based on this representation, we show how the contained implicit surface can be queried quickly. Then we explain in Section 5 how we efficiently construct this boundary representation in parallel as a *thick layered depth image* extended with depth range (TLDI). For this we show reuse of several parts of a standard point-based rendering pipeline. We describe in Section 6 that detecting collisions by querying the surface in this TLDI data structure is straightforward to do. In Section 7 we compare our collision detection algorithm with sampled meshes as ground truth and show that it is significantly more precise than prior methods, robust in the presence of noise and fast enough to handle dynamic point clouds at *interactive frame rates*. We give concluding arguments in Section 8 along with an outlook to the extensions we are currently working on.

4 Surface Definition

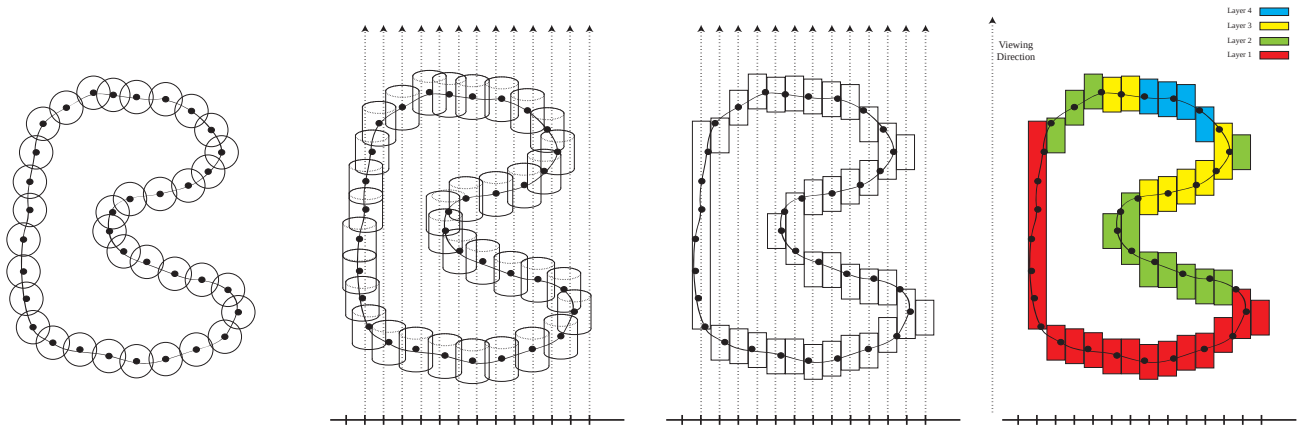
Our goal is to determine the distance of a point $p \in \mathbb{R}^3$ to the manifold, possibly bounded surface Σ that is implicitly defined by a set of points S, sampled on or close to it. All distances are in the Euclidean sense, unless otherwise noted. Since Σ is not known, we first define a *thickened boundary* Ω that contains such a surface near S, similar to an adaptive *spherical cover* as proposed in [15]. For precise evaluation of proximity queries to Σ we require that Ω bounds it as closely as possible, but also want to avoid holes in Ω that are not present in S. Evaluating the distance $\|p,\Omega\|$, which in turn allows us to approximate $\|p,\Sigma\|$, requires representation of Ω by a discrete spatial structure. Our design requirements are that it is compact and can be both constructed and evaluated quickly.

4.1 Spherical Cover Ω Containing the Surface Σ

First, we want to define a volume Ω which covers the surface Σ underlying the samples so that we can perform distance queries to Σ. Let $B_i(s_i,r_i)$ be the balls centered at samples $s_i \in S$ in $\mathbb{R}^3$ with radii r_i chosen such that Σ is enclosed entirely in the union of balls Ω (see Figure 2a):

$$\Omega = \bigcup_{i=0}^{N} B_i(s_i,r_i)$$

If S is sampled non-uniformly densely, balls which are close but from geodesically remote parts of the surface may merge in Ω, and then Ω is not homeomorphic to Σ. This is not a problem for our



(a) Spherical cover (b) Cylinders (c) Blended and discretized (d) TLDI

Figure 2: Representations of the volume bounding the surface Σ: a) Union of balls Ω centered at samples. b) Projected onto the view plane as cylinders in object space Ω'. c) Blended depth intervals $\hat{\Omega}$. d) TLDI shaded per layer.

use case, since determining the distance from a point $p \in \mathbb{R}^3$ to a surface neither requires that surface to be manifold nor orientable.

If the radii r_i associated with the samples are just sufficiently large with respect to local sampling density, the B_i will overlap such that Σ is entirely contained in Ω. We assume r_i either to be a global constant, estimated from range image properties or determined in preprocessing, as for out-of-core huge point clouds [19]. Alternatively, r_i could be estimated locally by determining k-nearest neighbors in screen space, as shown in [18]. Note that real holes in the surface which are smaller than r_i could disappear in the representation.

Since Ω consists of balls, its thickness perpendicular to Σ will be large and oscillate considerably between samples. Determining the connectivity between samples would allow us to blend their balls and result in a more equally thickened boundary. As mentioned above, inserting the balls into a spatial hierarchy in $\mathbb{R}^3$ to recover the connectivity is slow because we have to sort in three dimensions. Instead, we show how to achieve this more efficiently in projected (2-dimensional) space, which has something in common with splat rendering, as described next.

4.2 Blending Cylinders in Projected Space

We define samples in S as connected if their balls overlap. Now we want to locate the connectivity between the samples so that we can blend their associated balls for neighbors to equalize boundary thickness. This is easier if we project them from $\mathbb{R}^3$ onto a plane. Then we just need to locate overlapping disks in that plane and check if they also overlap in depth with their radii, similar to rendering view-plane aligned splats. In object space this represents testing plane-parallel cylinders which contain the balls and are of minimum size (see Figure 2b). We name the union of cylinders Ω'.

Each point $\hat{\mathbf{x}}$ in the projection plane (i.e., the view plane) represents a view ray in object space and may intersect Ω' multiple times. Therefore each $\hat{\mathbf{x}}$ maps to a set of depth ranges (entry-exit point pairs of Ω') which we call its *layers*, represented by the function $F_i(\hat{\mathbf{x}})$ for layer i:

$F_i(\hat{\mathbf{x}}) = \{d_{i,near}, d_{i,far}\}$

We want to equalize the boundary thickness of Ω' since its associated values of $F(\hat{\mathbf{x}})$ change discretely at cylinder boundaries. So we blend its values (both *near* and *far*) for the N connected samples $\mathbf{s}_i$ whose cylinders overlap with the corresponding entry-exit pair along the view ray of $\hat{\mathbf{x}}$ as follows:

$$\hat{F}_i(\hat{\mathbf{x}}) = \sum_{i=1}^{N} d_i r(\|\mathbf{x} - \mathbf{s}_i\|)$$

where $r(x) = e^{x^2}$. We call $\hat{\Omega}$ the volume defined by the depth range layers of $\hat{F}$.

In regions where the surface is mostly parallel to the view plane, the set of cylinders intersected by a view ray in one layer is such that each cylinder overlaps with each other in that set. Where the surface is oblique, this may not hold because the depth range of a layer becomes large. We call such a set of cylinders containing non-overlapping subsets as *stacked*. For such stacks, we blend the frontmost cylinder only with its overlapping cylinders in the stack to get $d_{i,near}$, and similar for the backmost cylinder to get $d_{i,far}$. Figure 3 explains the two cases.

Σ is not known but implicitly assumed through its set of samples S. Nevertheless, we would like Σ to be bound by $\hat{\Omega}$, so we attempt to define it to lie centered in $\hat{\Omega}$. The way in which Σ approximates S can then be thought of as similar as a blended surface of splats. Our results in Section 7 confirm that $\hat{\Sigma}$ is reasonably close to S.

We would like to define $\hat{\Sigma}$ as the set of centers of maximum balls contained in $\hat{\Omega}$ which touch both sides of its boundary. However, boundary sides of $\hat{\Omega}$ are not clearly defined, but shooting view rays through it results in entry/exit pairs. Based on that information we can define $\hat{\Sigma}$ as the set of centers of maximum balls contained in $\hat{\Omega}$ which are centered along a view ray and growing monotonically either from its entry or exit point. A view ray then contains for each layer $\hat{F}_i$ either one or two balls. $\hat{\Sigma}$ is similar to a subset of the medial axis [3] of $\hat{\Omega}$ as the maximizing of balls along the view ray prunes spurious branches in that direction. However, it may contain spurious branches in the other axes.

4.3 Discretization of $\hat{\Omega}$ in Screen Space

For efficient spatial sorting, we discretize $\hat{\Omega}$ into a 2D grid with screen-space resolution. This data structure is well suited to parallel processing as the point primitives are streamed onto the GPU and connectivity has local extent in screen space so there is not much interdependency.

The result is a kind of non-aligned voxelization, since each pixel can reference multiple layers in $\hat{F}$, but their depth range does not

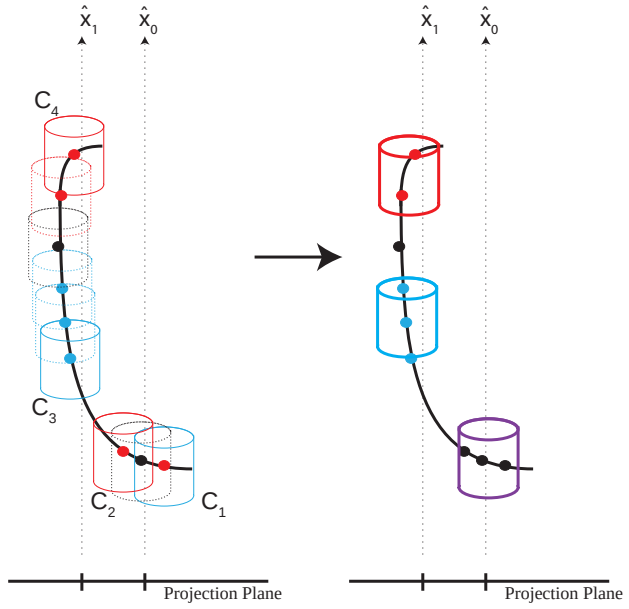


Figure 3: Left: This figure shows blending at stacked ($\hat{x}_1$) and non-stacked ($\hat{x}_0$) view rays. Frontmost (cyan) and backmost cylinders (red) are drawn with continuous lines, other cylinders as dashed. View ray $\hat{x}_0$ enters the layer at cylinder C_1 and leaves at C_2, intersecting three cylinders in total which all overlap and thus are blended together to a single d_i. View ray $\hat{x}_1$ on the other hand intersects a stack of cylinders, C_3 in the front and C_4 at the back. $d_{i,near}$ is then the result of blending C_3 with its overlapping (light cyan) cylinders and $d_{i,far}$ similar for C_4. Right: the result of blending are the cylinders drawn with thick stroke.

correspond between pixels (see Figure 2c). A closely related concept are *layered depth images* (LDI), which are typically used to peel off surface layers from a mesh as shown in [7]. In our case, layers represent depth ranges instead of scalar values, so we extend the depth of an LDI with a second value to represent the near (entry) and far (exit) intersection of the thickened boundary. We name this a *thickened LDI* (TLDI).

5 Construction of TLDI

Constructing the TLDI for a point cloud peels off layers similar as does depth peeling for a mesh (see Figure 2d). Since operations such as visibility culling, blending and normalization are involved, we can partially reuse work already done in the standard PBR pipeline which processes the points sequentially:

The Standard Three-Pass PBR Pipeline. The common pipeline of surface splatting employed by PBR algorithms is generally composed of three shader passes:

- *Visibility Pass:* All splats are simply rendered, depth culled, leaving only the front-most fragments in the output buffer.
- *Blending Pass:* Fragments of the splats that are within a certain threshold from the front depth values are rendered to accumulate the weighted colors and the weights themselves.
- *Normalization Pass:* The accumulated weighted colors value is divided by the accumulated weights value to get the blended depth.

For the blending and normalizing passes, we simply replace values of color with depth (front and back values respectively).

Modifications for TLDI layer computation:

For constructing a layer of the TLDI, we insert three passes between the visibility and blending pass:

- *Stacking Pass:* Since points are not processed in order, cylinders in a stack may occur after each other such that they do not overlap. We maintain a zero-initialized bit array for an assumed optimal stack size of size 128 bits, 32 bits for each one of the RGBA channels, quantized by the radius of the first encountered point. Subsequent cylinders encountered at that pixel and inside its range fill up the bits corresponding to their depth (see Figure 4).
- *Counter Pass:* The number of contiguous filled bits is determined, starting from the first filled bit.
- *Back Visibility Pass:* The previous count determines the backmost cylinder in that stack and also in the current layer.

The two pipelines are displayed in Figure 5. For each pixel in the TLDI, a pair of depth values (d_{near}, d_{far}) is output. In our implementation, we actually store their average d_{avg} along with half their distance, because for non-stacked pixels, d_{avg} already represents Σ.

In our experiments, we managed to capture all layers entirely within our assumed stack size of 128 bits. However it is important to note that layers exceeding this size would simply be split up into two, adding another layer to the data structure but not changing the underlying representation. We expect this to minimally decrease performance, but accuracy would not be affected.

We execute the above pipeline for each layer of the point cloud, however the collision detection application that we present next often terminates already after a single layer has been constructed.

6 Collision Detection as an Application

We now present collision detection as one application of querying the TLDI representation of the implicit surface of S. We show that it can be implemented efficiently by merging TLDI construction and collision testing into an existing PBR pipeline.

Simply put, collisions are detected by intersecting view rays from the camera for each pixel with the TLDI for each point cloud and testing if their depth ranges along that ray (since close to $\hat{\Sigma}$) intersect. For non-colliding point clouds, this also infers the separation distance in view direction, which especially makes sense for an object moving with the camera, such as an avatar. We describe next how the two point clouds' collision and distance queries are processed.

6.1 TLDIs Comparison

Comparisons between layers are performed pixel wise. A collision is detected if at a pixel the depth ranges for layers from two objects overlap. Since the boundary is thickened, we expect a number of false positives, i.e., $\hat{\Omega}_0, \hat{\Omega}_1$ of the point clouds intersect while the actual surfaces $\hat{\Sigma}_0, \hat{\Sigma}_1$ do not. In our experiments we discovered that we could compress the thickness of $\hat{\Omega}$ in view direction by a significant factor in order to eliminate most false positives while keeping the number of false negatives small. Since $\hat{\Omega}$ is projected in view direction, thinning it in this axis does not affect the general observations made in Section 4, in fact it approximates Σ more closely.

Since a collision may already be detected in the first layer (which terminates our method), we do not have to construct all layers of the object and compare them against each layer of the other object. Instead, we compute them in depth order on demand as long as no collision is detected, as outlined below. This limits the number of

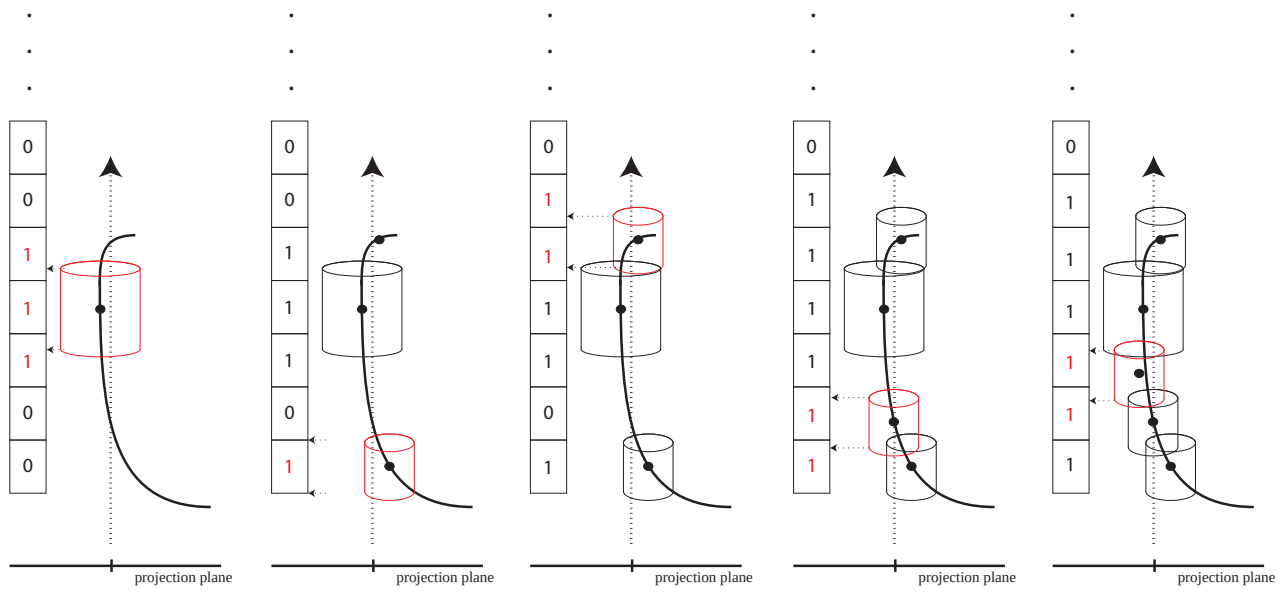


Figure 4: The figure demonstrates how the connectivity of a stack inside a layer is tracked by a bit array in the *stacking pass*. Depth is quantized into segments, where each segment is of height equal to the diameter of the first encountered cylinder in the initial *visibility pass*, and the first segment aligned to its lower disc. In the subsequent *stacking pass*, cylinders of encountered points are projected to gain occupancy information, as shown in order. Each cylinder fills the segments in the buffer bit that intersects with its cylinder depth interval. The order in which points are projected is assumed to be random, and cylinders in the figure are not all the same size.

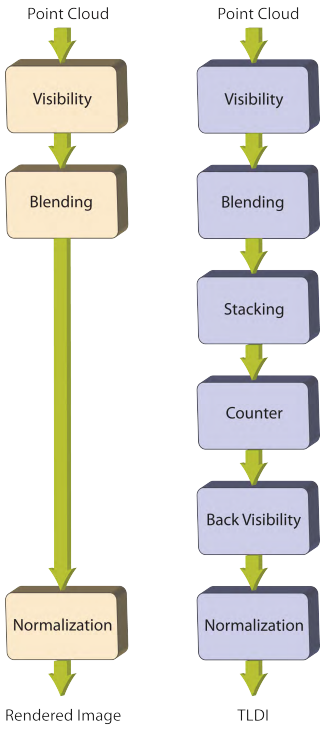


Figure 5: For TLDI construction we insert three additional passes into a standard PBR pipeline.

depth comparisons by the number of layers per object, corresponding to its view-dependent depth complexity.

Let there be two point clouds O_0, O_1. For O_0, the first and second layers – $l_{0,0}$ and $l_{0,1}$ – are computed. Then we consider the subset of O_1 which is clipped by the depth range of – and between – $\hat{\Omega}$ of $l_{0,0}$ and $l_{0,1}$, construct its layers and compare it against those of O_0. The clipping is repeated similar between subsequent layer pairs $l_{0,i-1}$, $l_{0,i}$, for $i \geq 2$, until the backmost layer of O_0 has been reached or a collision is detected.

6.2 Early Rejection Test

Since entire TLDIs have to be constructed in case of non-collision, we add a quick rejection test in the beginning, where we simply compare the front layer of O_0 to the back layer of O_1, and vice versa. Since the front layer is the closest to the camera, a non-collision is reported if for all pixels d_{near} of the front layer of O_0 is greater than d_{far} of the back layer of O_1. If the depth intervals of those layers intersect, a collision is detected early as well. This test can be performed quickly inside the standard PBR pipeline, as only one depth interval is required to be blended. If the test does not deliver any result, we continue the normal procedure of comparing the layers, in which the already computed layers can be reused.

6.3 Integration into Existing PBR Pipeline

The overlap between the standard PBR pipeline and TLDI construction suggests an integration between those two. The early rejection test already uses the same standard pipeline of rendering to create the two front layers. In fact, the only difference is what is being blended, color or depth. However, in rendering the two objects are processed and z-culled together, while TLDI construction processes one object at a time. The preferred integration scenario is to blend both colors and depth while creating the front TLDIs of O_0 and O_1. While the TLDIs are used for collision detection, the two frames with the blended colors can be merged into one by performing z-culling at each pixel using the corresponding TLDIs to choose which color value is copied to the merging frame pixel.

We note that TLDI construction profits from the reuse of PBR pipeline calculations for point-cloud data located inside the view frustum. For many application scenarios, only these data are of interest anyway, e.g., an avatar moving with the camera.

Point clouds are almost always perspectively projected onto the screen by rendering pipelines, whereas the cylinders in sections 4, 5, 6 are assumed to be orthogonally projected. The integrated pipeline has to use the same projection for both rendering and TLDI and so we use perspective projection in our experiments. This results in perspective foreshortening of the cylinders and thus turns them into truncated cones. However, since the resulting surface is only an approximation, accuracy is not affected significantly as the results in Section 7 confirm.

6.4 Distance Queries

In addition to collision queries, our method can also incidentally answer distance queries in case of non-collision. Since we do not consider $\hat{\Sigma}$ as orientable, the distance function we calculate with respect to it is unsigned. We can therefore not decide if the distance between two objects is one of separation or of penetration. When layers $l_{0,i}, l_{1,j}$ are compared to determine a collision, their absolute depth difference per pixel is calculated as follows:

$d(\hat{\mathbf{x}}) = min(|\hat{F}_i(near) - \hat{F}_j(far)|, |\hat{F}_j(near) - \hat{F}_i(far)|)$

We keep track of $d_{min}(\hat{\mathbf{x}})$ at each pixel and then report its overall minimum in case of non-collision, which yields the separation distance in view direction.

7 Results

The algorithm is implemented in C++, OpenGL and GLSL. Tests were run on Core2 Quad processor, 2.4 GHz, with 4 GB RAM, and GeForce GTX 680 graphics processor.

We used the benchmark proposed by [21] for testing, in which two copies of the same model are tested for collision against each other. Both objects are normalized to fit in a 2^3 cube. The center of one object is positioned at the origin, and the center of the second is positioned at a distance d_0 from the origin along the +x direction. The second object is compositely rotated about the y-axis and the z-axis, with a number of small steps. The composite rotations are iteratively repeated, each iteration starting from a position at a distance $d_i = d_0 - i\Delta d$ from the origin. We initialized d_0 with 3.0, 31 iteration/distance, and 30 steps per rotation, which makes 900 cases per iteration/distance, and 27900 total cases. The camera is positioned at coordinates $(0,0,+4)$, and facing the origin. Point clouds are perspectively projected with view angle 90° at planes $z = +1$ and $z = +6$ from the camera position.

Accuracy and runtime results are computed for each distance by averaging the results of all steps in the corresponding rotation. Accuracy tests are performed by comparing our outcomes with those of an exact mesh collision procedure. We call this percentage the accuracy error, but it should be noted that *it is not actually an error*. The polygonal mesh of a point cloud is an approximation of the surface, but not the surface itself. So, we consider the mesh collision test results as an *approximation of the ground truth*.

Four models were used for testing: Stanford BUNNY, ARMADILLO, DRAGON, and HAPPY polygonal models. For each model, the point set has been extracted from the mesh and the r_i for its points determined by *kNN* with $k = 7$ for testing purposes, where r_i is set to 0.65 of the computed distance. An efficient screen-space method to determine a radius containing *kNN* has been demonstrated in [18]. They project cylinders onto a rather large frame, 1024×1024, to achieve accurate results.

Besides synthetic models, we also tested collision detection between the huge data set of the houses of EPHESUS (> 5M points) captured by a laser scanner (see Figure 6), and a single HAPPY model. We tried to imitate an interactive navigation experience by considering the HAPPY model a human discovering the big model.

Figure 6: The houses of EPHESUS. Left: View from above of the point cloud. Right: Closer view with splat rendering.

Table 1: The percentages of false negatives and false positives with various compression ratios ρ for: **B**UNNY, **A**RMADILLO, **D**RAGON, **H**APPY.

ρ	false negatives (%)				false positives (%)			
	B	A	D	H	B	A	D	H
1.0	0	0	0	0	0.92	0.47	0.47	0.12
0.5	0	0	0	0	0.68	0.29	0.29	0.08
0.25	0	0	0	0	0.52	0.23	0.21	0.03
0.1	0	0.1	0.01	0	0.43	0.18	0.15	0.02
0.05	0	0.01	0.02	0.01	0.39	0.17	0.14	0.02
0.01	0.01	0.03	0.05	0.05	0.33	0.16	0.13	0.01
0.005	0.05	0.04	0.06	0.06	0.3	0.16	0.13	0.01

The same benchmark described above is used, where EPHESUS keeps its size, HAPPY scaled to the size of a human – relative to EPHESUS – and the view emanates from the eyes of HAPPY and directed forward. HAPPY is initially positioned at coordinates $(0,0,+D)$, where D is the x-extent of the EPHESUS bounding box. Number of iterations and number of steps per iteration are the same as above. A polygonal mesh of EPHESUS is not available, so only runtime is measured.

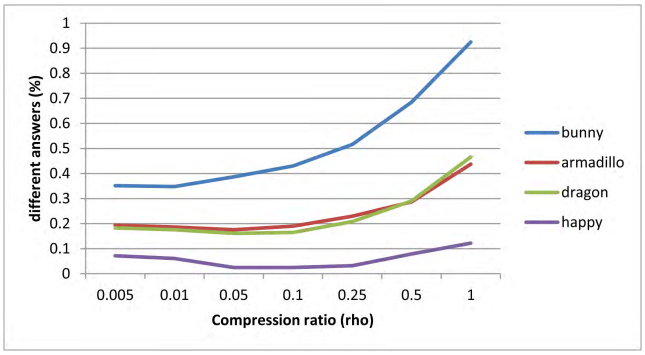


Figure 7: Accuracy error of collision detection compared with the sampled mesh as approximate ground truth. Based on these results we chose $\rho = 0.05$ for subsequent tests.

When querying the intersection between thickened bounding intervals as an indicator of collision, we have encountered very few cases of false negatives in our experiments (none for most object pairs tested), but the ratio of false positives is rather high (see Table 1). Compressing the intervals as mentioned in Section 6.1 with a factor ρ decreases this number effectively while yielding only an insignificant number of false negatives. The accuracy resulting from different compression values is plotted in Figure 7, which shows that accuracy peaks for ρ between 0.01 and 0.05. Smaller values of ρ yield less false negatives, but more false positives, which results in less overall accuracy. Based on that, we chose $\rho = 0.05$ for all

Table 2: Point clouds with total TLDI construction time (in millisec). CD runtime, time overlap with rendering, and error from mesh ground truth are averaged following the benchmark of Section 7.

Model	Size	TLDI	CD	Render	Error
Bunny	36k	13.9	2.2	0.9	0.39 %
Armadillo	173k	43.7	8.1	3.1	0.18 %
Dragon	438k	122.5	15.8	6.7	0.16 %
Happy	544k	134.9	18.3	8.2	0.03 %

following tests, to maximize accuracy (= minimizing sum of false negatives and false positives).

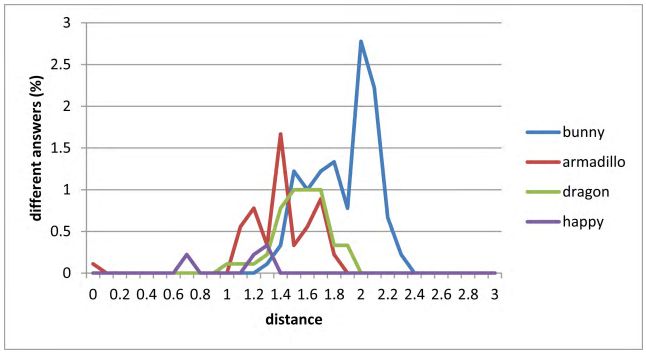


Figure 8: Accuracy error vs distance for all models.

Accuracy

Figure 8 shows a plot of accuracy error against distance d_i. For all models, the error is zero when the two objects centers are either far or close, and increases in between, where the object surfaces collide. The accuracy error stays below 3% for all models for any distance. Table 2 shows accuracy error averaged over distances, and is always below 0.4%. Note that false negatives result from $\hat{\Omega}$ not covering $\hat{\Sigma}$ entirely. This can occur if either ρ is too small, overly compressing $\hat{\Omega}$, or if radii are estimated too small.

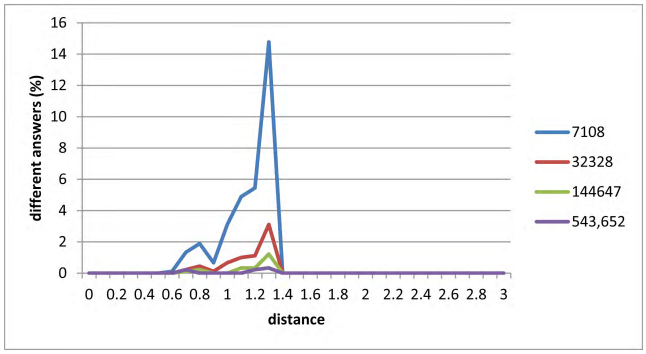


Figure 9: Accuracy error for different resolutions of HAPPY.

We are particularly satisfied by the accuracy results of our method. In Figure 10, [21] showed accuracy error of different resolutions of HAPPY. We reproduce this plot based on the same benchmark in Figure 9, albeit with the different resolutions of HAPPY which were available to us. Interestingly their accuracy do not improve much when increasing sampling density (always $< 7\%$). Our results improve significantly with increasing sampling density as we expect the TLDI to approximate the surface better, down to $< 0.3\%$ for the original resolution.

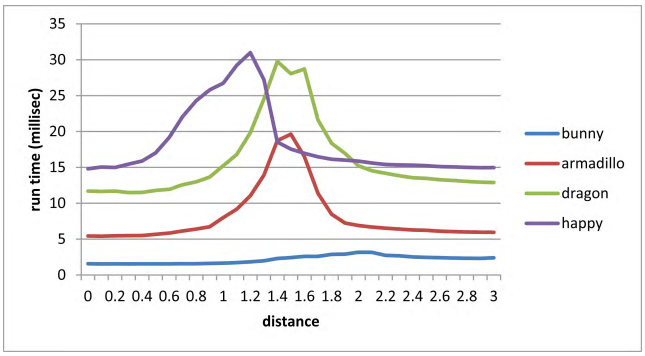


Figure 10: Runtime vs distance.

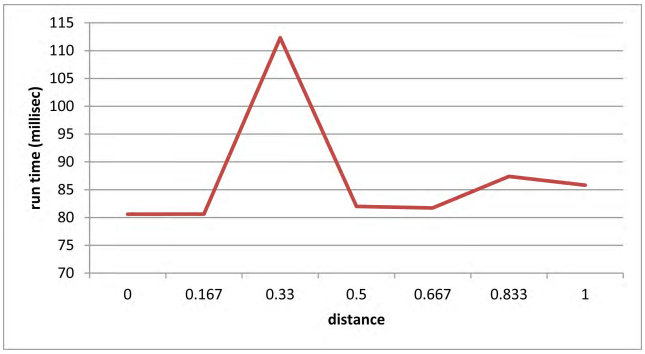


Figure 11: Runtime for colliding large EPHESUS (5M points) with HAPPY averaged over different distances (using benchmark). The numbers on the x-axis are the distances normalized to $[0, 1]$.

Runtime

The result of the benchmark shows that runtime increases approximately linearly with point cloud size (see Table 2). The same table also shows the large overlap of collision detection with the rendering pipeline: about 40% of collision detection runtime is removed if the point clouds are rendered as well. Colliding the large EPHESUS model with HAPPY is still possible at interactive frame rates (see Figure 11). Similar to accuracy, runtime also decreases for near or far object centers and increases in between where surfaces collide, as Figure 10 demonstrates.

Figure 12 confirms that the early rejection test outlined in Section 6.2 reduces runtime significantly as well.

The runtime of TLDI construction is directly proportional to point cloud size, and number of captured layers, whereas the number of layers in turn depends on how tight the TLDI is. The more the TLDI adheres to the actual surface, the more layers are captured and the longer the construction time. TLDI tightness is controllable via scaling the splats radius and the frame resolution. For collision detection, it is necessary to construct a tight TLDI to achieve accurate results, but this is balanced by the fact that it is not necessary to construct the whole TLDI as explained in Section 5. For other applications where the TLDI functions as a bounding volume rather than a surface estimator, the construction time can be traded off with the tightness level. Table 3 shows the runtime of full TLDI construction for HAPPY and DRAGON models, against different frame resolutions and splat scales. The table shows the accuracy error values as well. The smallest error is achieved by a radius scale of 1.0 and frame resolution of 1024×1024. There is no specific rule how accuracy error changes with the two parameters, but the trend is that it decreases as the radius scale increases.

Comparing the runtime of our method to others, e.g. [21], is of

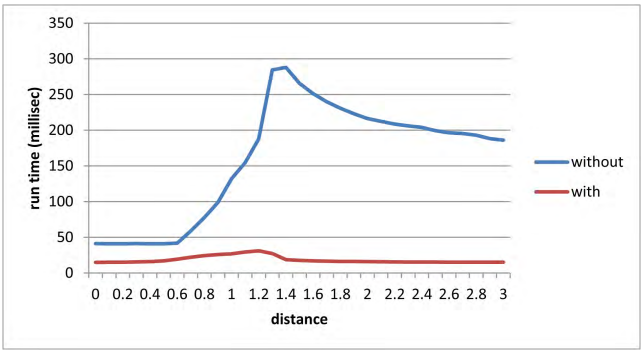


Figure 12: Runtime for HAPPY (using benchmark), without (blue) and with (red) the early rejection test. It clearly shows that it increases efficiency significantly.

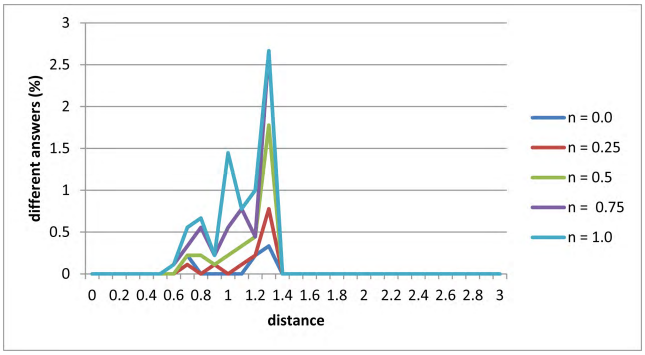


Figure 13: Accuracy error for different levels of uniform Gaussian noise ($\sigma = nr_{avg}$) added to HAPPY.

Table 3: The average TLDI construction time for HAPPY and DRAGON models at different frame resolutions and splat scales. Average number of captured layers are denoted inside brackets, and accuracy error values are denoted below in italic. The values on top of the columns indicate the scaling value of the splat radius. The numbers show how the TLDI construction time decreases as the frame resolution decreases and the splat radius scale increases.

Model	Resolution	1.0	3.0	5.0
Happy	1024×1024	134.9(10.4)	88.2(6.9)	67.5(5.4)
		0.16%	*0.79%*	*1.9%*
	512×512	121.8(9.5)	85.3(6.8)	68.6(5.6)
		0.18%	*0.67%*	*1.7%*
	256×256	113.4(9.0)	75.9(6.2)	63.2(5.1)
		0.71%	*0.57%*	*1.4%*
Dragon	1024×1024	122.5(11.5)	98.2(9.1)	93.4(6.4)
		0%	*0.20%*	*0.24%*
	512×512	116.2(11.1)	94.2(9.0)	64.6(6.2)
		0.16%	*0.19%*	*0.63%*
	256×256	108.3(10.1)	89.9(8.7)	62.5(6.2)
		0.67%	*0.25%*	*0.42%*

limited usefulness. Their algorithm was designed for static objects, as the underlying structure takes considerable time to construct. For those pre-processed data structures it performs collision queries faster than ours (being of $O(logN)$ versus our $O(N)$ complexity), but for dynamically changing objects, hierarchy construction time needs to be added to each query, and for that our algorithm is faster by orders of magnitude, even considering that they were measured on older hardware. The construction of their underlying BVH may become faster if parallelized and performed on modern hardware. The tested models contain about 1M points. For objects of that size, a BVH-based algorithm of $O(NlogN)$ complexity would require an extra time factor $O(logN)$ of 20, which is quite large.

Robustness to Noise

Since the boundary of Ω is thickened to the extent of sampling density, we expect it to smooth noise up to a similar level. We tested the robustness of our approach by adding Gaussian noise with different σ to HAPPY, the most densely sampled synthetic model used in our tests. We set $\sigma = nr_{avg}$, where r_{avg} is the average over r_i, and random $n = [0,1]$. Runtime did not change significantly and accuracy error was always below 3% (see Figure 13).

Real Data and Dynamic Simulation

Point clouds captured with the Kinect often exhibit noise and holes, as ROOM (300k points, captured by Kinect) shown in Figure 14a. Figures 14b-d show snapshots of collision detection between BUNNY and ROOM. Collisions are robustly detected near flat surfaces and small holes. We also simulated a dynamic environment of an animated model (HORSE) (10 frames, 8.5k points each) traversing the EPHESUS model. Figure 15 shows snapshots from the simulation. BUNNY and HORSE are rendered as meshes in the figures for visual plausibility.

7.1 Complexity Analysis

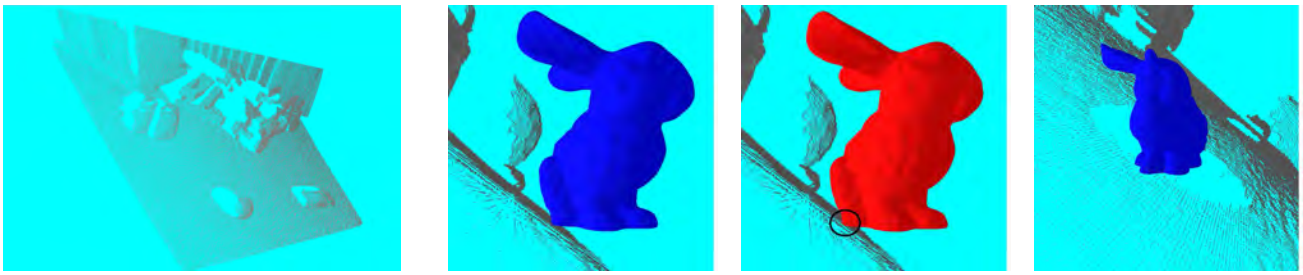
Worst-case time complexity between pairs of objects occurs only if there is no collision and the early rejection test does not detect that. An example is an object that is partially obscured from the view point by a concavity in the other object. This requires construction of all TLDIs per point cloud and comparing all of those for one point cloud against the subsets of TLDIs clipped between them. TLDI construction is linear in the size of the point clouds, with the added factor of depth complexity, as points are processed in order for each layer and therefore $O(LN)$.The collision test is output-sensitive with $O(LXY)$ for screen space resolution $X \times Y$ and scales with the depth complexity of the point cloud being clipped.

For collision detection among a set of more than two point clouds, using the proposed algorithm would make the overall runtime (both construction and collision detection) quadratic, as the construction of a cloud TLDI is dependent on the other cloud TLDI and thus would be reconstructed for each comparison. However, if the number of clouds is large and the size of each cloud is relatively small, we could also construct the TLDI of each point cloud just once and separately. The then linear TLDI construction time has the trade-off that the previously linear time of collision detection becomes $O(L_1L_2XY)$.

If we compare a single large point cloud (environment) against multiple small ones (avatars), we could also use another approach. In that case, all avatars are treated as a single combined point cloud, and the same complexity of a single pair comparison holds. Increasing the number of avatars in that scenario increases N_2 in the above expression, and therefore construction time increases linearly. In order to know which avatars collide with the environment, labels at the points would have to be stored as well, which would result in a small increase in memory storage.

8 CONCLUSION AND FUTURE WORK

We have proposed a novel data structure for representing the surface of dynamic point clouds. We show that it can be constructed efficiently and reuse computation from an existing PBR pipeline. As an application we have demonstrated online collision detection for large models. Our results show that our surface extraction is significantly more precise than for a previous method [12], especially where points are densely sampled, and that it is also robust to noise since the surface underlying the points is thickened.



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

Figure 14: Collision detection between BUNNY and the (a) Kinect captured ROOM. BUNNY is blue in cases of non collision, and turns red in cases of detected collisions. (b) shows BUNNY near a flat surface, and crosses it in the next frame (c). Both cases are correctly detected. BUNNY passes through a wide hole in (d) which is not recognized as part of the surface, and thus no collision is detected.

Figure 15: Animated HORSE inside EPHESUS, passing through a column.

We are currently working to improve our data structure in terms of compactness and efficiency of construction and traversal. Implementation of the more exact surface extraction for stacks would even further increase accuracy, since we currently simply assume the center of the depth range of a layer along a view ray to be the surface intersection. We think that augmenting the TLDI with data from sampling such as normals and uncertainty information could permit even more precise surface extraction. TLDI could also be used instead of a voxelization as a more compact representation, for example to accelerate global illumination computations.

ACKNOWLEDGEMENTS

This research was supported by the EU FP7 project HARVEST4D (no. 323567).

REFERENCES

[1] R. K. Anjos, J. M. Pereira, and J. F. Oliveira. Collision detection on point clouds using a 2.5+d image-based approach. *J. of WSCG*, 20(2):145–154, 2012.

[2] G. V. D. Bergen. Efficient collision detection of complex deformable models using AABB trees. *J. of Graphics Tools*, 4(2):1–14, 1997.

[3] H. Blum. A Transformation for Extracting New Descriptors of Shape. In W. Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, Cambridge, 1967.

[4] B. Curless and M. Levoy. A volumetric method for building complex models from range images. *Proc. SIGGRAPH*, pages 303–312, 1996.

[5] S. A. Ehmann and M. C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *cgforum*, 20:500–510, 2001.

[6] E. Eisemann and X. Dècoret. Fast scene voxelization and applications. *ACM SIGGRAPH Symp. on Interactive 3D Graphics & Games*, pages 71–78, 2006.

[7] C. Everitt. Interactive order-independent transparency. Technical report, NVIDIA, 2001.

[8] S. Gottschalk, M. Lin, and D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. *SIGGRAPH 96 Conf. Proc.*, pages 171–180, Aug 1996.

[9] N. K. Govindaraju, M. C. Lin, and D. Manocha. Fast and reliable collision culling using graphics hardware. *Vis. and Computer Graphics, IEEE Trans. on*, 12(2):143–154, Mar-Apr 2006.

[10] B. Heidelberger, M. Teschner, and M. H. Gross. Detection of collisions and self-collisions using image-space techniques. In *J. of WSCG*, volume 17, pages 145–152, 2004.

[11] T. Hinks, H. Carr, L. Truong-Hong, and D. Laefer. Point cloud data conversion into solid models via point-based voxelization. *Surveying Engineering*, 139(2):7283, 2013.

[12] J. Klein and G. Zachmann. Point cloud collision detection. In *Eurographics 2004*, volume 23, pages 567–576, Sep 2004.

[13] J. T. Kloswski, M. Held, J. S. B. Mitchell, H. Sowrizal, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Trans. on Vis. & Com. Graphics*, 1(4):21–36, Jan 1998.

[14] D. Knott and D. K. Pai. Cinder: Collision and interference detection in real-time using graphics hardware. In *Graphics Interface*, pages 73–80, May 2003.

[15] Y. Ohtake, A. Belyaev, and H.-P. Seidel. An integrating approach to meshing scattered point data. In *Proc. of 2005 ACM symp. on Solid & physical modeling*, pages 61–69. ACM, 2005.

[16] J. Pan, S. Chitta, and D. Manocha. Probabilistic collision detection between noisy point clouds using robust classification. *Int. Symp. on Robotics Research*, 2011.

[17] J. Pan, I. A. Sucan, S. Chitta, and D. Manocha. Real-time collision detection and distance computation on point cloud sensor data. In *IEEE Int. Conf. on Robotics & Automation*, pages 3593–3599, 2013.

[18] R. Preiner, S. Jeschke, and M. Wimmer. Auto splats: Dynamic point cloud visualization on the gpu. In H. Childs and T. Kuhlen, editors, *Proc. of Eurographics Symp. on Parallel Graphics & Vis.*, pages 139–148. Eurographics Association 2012, may 2012.

[19] C. Scheiblauer and M. Wimmer. Out-of-core selection and editing of huge point clouds. *Computers & Graphics*, 35(2):342–351, Apr 2011.

[20] D. Steinemann, M. Otaduy, and M. Gross. Efficient bounds for point-based animations. *Symp. Point-Based Graphics*, pages 57–64, 2007.

[21] G. Zachmann. Minimal hierarchical collision detection. In *ACM Symp. on Vir. Reality Software and Tec.*, pages 121–128, Nov 2002.