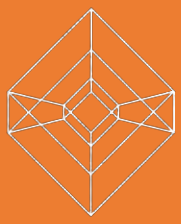


HARVEST4D

HARVESTING DYNAMIC 3D WORLDS FROM COMMODITY SENSOR CLOUDS

Publications for Task 3.3

Deliverable 3.31

Date:	27.06.2016
Grant Agreement number:	EU 323567
Project acronym:	HARVEST4D
Project title:	Harvesting Dynamic 3D Worlds from Commodity Sensor Clouds

Document Information

Deliverable number	D3.31
Deliverable name	Publications for Task 3.3
Version	0.1
Date	2016-06-27
WP Number	3
Lead Beneficiary	PARISTEC
Nature	R
Dissemination level	PU
Status	Draft
Author(s)	PARISTEC

Revision History

Rev.	Date	Author	Org.	Description
0.1	2016-06-27	Tamy Boubekeur	PARISTEC	Initial Draft

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This deliverable contains original unpublished work except where clearly indicated otherwise.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 3.3, and how they fit into the work plan of the project.

The objective of Task 3.3 is to provide the Harvest4D project with fast and scalable geometric operators. These operators may be the result of the research conducted in WP3 or linked the many applications scenarios targeted by the project. The main contributions for this task relate in particular to the computation of efficient level-of-details of the geometry to adapt the data to a number of computing capabilities, algorithm complexities and application needs. We developed new approaches to the fast adaptive resampling of geometric data, including fast simplification of surface and medial meshes, fast decimation of point clouds, fast remeshing of volume meshes and fast point-sampling of a 3D scene with application to global illumination approximation. Just such as extreme approximation was key to T3.2, fast simplification -- and the dual ability to still reconstruct data from fewer sample -- appeared to be a critical task at every step of global "process and analysis" pipelines such as the ones that Harvest4D aim at feeding. This research work has led to a number of publications and two awards: the second prize for the Wolfgang Strasser Award at the ACM SIGGRAPH/Eurographics HPG 2015 and the Best Paper Award at Shape Modeling International 2016. In the following, we give an overview of the main contributions.

1.2 PUBLICATIONS

So far, there are 3 publications that are mainly associated to Task 3.3, and these can be found in the appendix of this deliverable:

- Noura Faraj, Jean-Marc Thiery, and Tamy Boubekeur
  Progressive Medial Axis Filtration
  Proceedings ACM SIGGRAPH Asia 2013, Technical Brief program
- Beibei Wang, Jing Huang, bert Buchholz, Xiangxu Meng, and Tamy Boubekeur
  Factorized Point-Based Global Illumination
  Computer Graphics Forum (Proc. EGSR 2013)
- Hélène Legrand and Tamy Boubekeur
  Morton Integrals for High Speed Geometry Simplification
  ACM SIGGRAPH/Eurographics High Performance Graphics, 2016, **Second prize for the Wolfgang Strasser Award**
- Stefan Ohrhallinger, Scott A. Mitchell and Michael Wimmer
  Curve Reconstruction with Many Fewer Samples
  Computer Graphics Forum (Proc. SGP 2016)
- Noura Faraj, Jean-Marc Thiery, and Tamy Boubekeur

Multi-Material Adaptive Volume Remesher
Computer & Graphics Journal, Proceedings of the Shape Modeling International 2016, **Best Paper Award**

- Gilles Laurent, Cyril Delalandre, Jean-Baptiste de La Rivière and Tamy Boubekeur
  Forward Light Cuts: a Scalable Approach to Real-time Global Illumination
  Computer Graphics Forum (Proc. EGSR 2016)

Additionally, a number of other papers are primarily attached to other tasks and deliverables but exhibit contributions on scalable implementations of 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
- Christian Kehl, Tim Tutenel and Elmar Eisemann
  Smooth, interactive rendering techniques on large-scale, geospatial data in flood visualisations
  ICT Open 2013
- 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
- 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
- Mohamed Radwan, Stefan Ohrhallinger, and Michael Wimmer
  Efficient collision detection while rendering dynamic point clouds
  Proceedings of Graphics Interface 2014 (GI '14), 2014
- Sebastien Ochmann, Richard Vock, Raoul Wessel and Reinhard Klein
  Automatic Reconstruction of Parametric Building and Models from Indoor Point Clouds
  Proceedings of CAD/Graphics, 2015
- Gianpaolo Palma, Paolo Cignoni, Tamy Boubekeur and Roberto Scopigno
  Detection of Geometric Temporal Changes in Point Clouds
  Computer Graphics Forum (Presented at EUROGRAPHCIS 2016)
- Bas Dado, Timothy R. Kol, Pablo Bauszat, Jean-Marc Thiery, Elmar Eisemann
  Geometry and Attribute Compression for Voxel Scenes
  Computer Graphics Forum (Proc. of EUROGRAPHCIS 2016)
- Leonardo Scandolo, Pablo Bauszat, Elmar Eisemann
  Compressed Multiresolution Hierarchies for High-Quality Precomputed Shadows
  Computer Graphics Forum (Proc. of EUROGRAPHCIS 2016)

- Jean-Marc Thiery, Emilie Guy, Tamy Boubekeur and Elmar Eisemann
  Animated Mesh Approximation With Sphere-Meshes
  ACM Transactions on Graphics (presented at SIGGRAPH 2016)

2 DESCRIPTION OF PUBLICATIONS

2.1 OVERVIEW

The main contributions of this task can be decomposed into two categories: application-oblivious fast methods for geometry processing and rendering-oriented geometric methods. While for the former, no assumption on the final application is made beyond the input parameters of the algorithms, the latter is motivated by the growing need of geometry/scene processing and analysis which are required to achieve high quality fast rendering. In the following, we first go over the different shape representations, from surface to volume through medial axis (which link both kinds of models) and summarize different fast down-/re-sampling strategies. Then, we explain how efficient algorithm can benefit from the geometric structure of the scene for both offline and real time approximation of global illumination.

2.2 MORTON INTEGRALS FOR ADAPTIVE SURFACE MESH DOWNSAMPLING

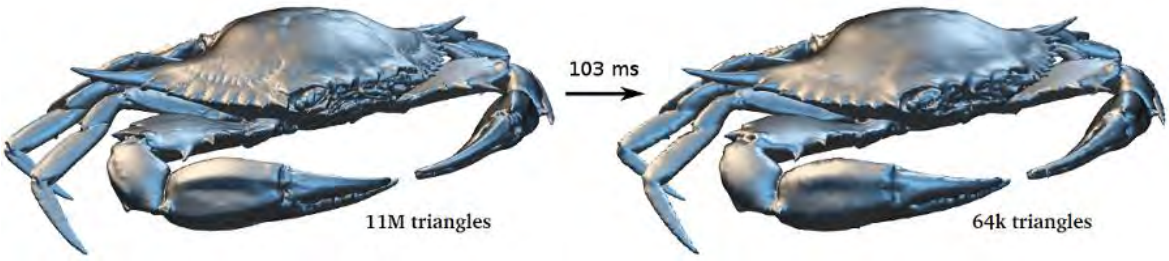


Figure 1 HSGS can reduce this model by a factor of 20 in about 103 ms on a single Nvidia GTX680, 43ms on a more recent GTX 980 ti one, without any preprocessing.

Real time geometry processing has progressively reached a performance level that makes a number of signal-inspired primitives practical for on-line applications scenarios. This often comes through the joint design of operators, data structures and even dedicated hardware. Among the major classes of geometric operators, filtering and super-sampling (via tessellation) have been successfully expressed under high-performance constraints. The subsampling operator i.e., adaptive simplification, remains however a challenging case for non-trivial input models. With *HSGS* [Legrand and Boubekeur 2015], we build a fast geometry simplification algorithm over a

new concept: Morton Integrals. By summing up quadric error metric matrices along Morton-ordered surface samples, we can extract concurrently the nodes of an adaptive cut in the so-defined implicit hierarchy, and optimize all simplified vertices in parallel. This approach is inspired by integral images and exploits recent advances in high performance spatial hierarchy construction and traversal. As a result, our GPU implementation can down-sample a mesh made of several millions of polygons at interactive rates, while providing better quality than uniform simplification and preserving important salient features. We present results for surface meshes, polygon soups and point clouds, and discuss variations of our approach to account for per-sample attributes and alternatives error metrics.

This work received the second prize for the Wolfgang Strasser Award at the ACM SIGGRAPH/EUROGRPHICS High Performance Graphics 2015 conference in Los Angeles, United States, and is used within the integrated prototype of Harvest4D and also part of the prototype deliverable D3.32

2.3 PROGESSIVE DOWNSAMPLING OF THE MEDIAL MESH

Between surface and volume models, the medial axis is a particular mesh structure that acts as an interface representation that allows characterizing globally a shape for subsequent processing and analysis (see T3.2). Again, simplification and abstraction of such a representation is often to key to its scalable use.

The Scale Axis Transform provides a parametric simplification of the Medial Axis of a 3D shape which can be seen as a hierarchical description. However, this powerful shape analysis method has a significant computational cost, requiring several minutes for a single scale on a mesh of few thousands vertices. Moreover, the scale axis can be artificially complexified at large scales, introducing new topological structures in the simplified model. With *P-MAT* [Faraj 2013], we propose a progressive medial axis simplification method inspired from surface optimization techniques which retains the geometric intuition of the scale axis transform. We compute a hierarchy of simplified medial axes by means of successive edge-collapses of the input medial axis. These operations prevent the creation of artificial tunnels that can occur in the original scale axis transform. As a result, our progressive simplification approach allows to compute the complete hierarchy of scales in a few seconds on typical input medial axes. We show how this variation of the scale axis transform impacts the resulting medial structure.

2.4 ADAPTIVE RESAMPLING OF VOLUME MESHES

When moving to full volumetric representation of models, tet-meshes are often acting as the core structures to approximate object composed of numerous materials for which a 3D cartography is key in simulation and visualization. Here, reducing the size of these models becomes even more important than for surface and medial representations.

With *MAD-Remesher* [Faraj 2016], we propose a practical iterative remeshing algorithm for multi-material tetrahedral meshes which is solely based on simple local topological operations, such as edge collapse, flip, split and vertex smoothing. To do so, we exploit an intermediate implicit feature complex which reconstructs piecewise smooth multi-material boundaries made of surface patches, feature edges and corner vertices. Futhermore, we design specific feature-aware local remeshing rules which, combined with a moving least square projection, result in high quality isotropic meshes representing the input mesh at a user defined resolution while preserving important features. Our algorithm uses only topology-aware local operations, which allows to process difficult input meshes such as self-intersecting ones. We evaluate our approach on a collection of examples and experimentally show that it is fast and scales well.

This work received the Best Paper Award at the Shape Modeling International 2016 conference in Berlin, Germany.

2.5 CURVE RECONSTRUCTION FROM FEW SAMPLES

The previous work demonstrates that fast mechanisms can be design to properly sample complex objects with fewer element, in different dimension. Going back to the simpler 2D case, we conducted a more theoretical study on how very few samples can be enough to reconstruct a complex object.

More precisely, we consider the problem of sampling points from a collection of smooth curves in the plane [Ohrhallinger 2016], such that the Crust family of proximity-based reconstruction algorithms can rebuild the curves. Reconstruction requires a dense sampling of local features, i.e., parts of the curve that are close in Euclidean distance but far apart geodesically. We show that epsilon<0.47-sampling is sufficient for our proposed HNN-CRUST variant, improving upon the state-of-the-art requirement of epsilon<1/3-sampling. Thus we may reconstruct curves with many fewer samples. We also present a new sampling scheme that reduces the required density even further than epsilon<0.47-sampling. We achieve this by better controlling the spacing between geodesically consecutive points. Our novel sampling condition is based on the reach, the minimum local feature size along intervals between samples. This is mathematically closer to the

reconstruction density requirements, particularly near sharp-angled features. We prove lower and upper bounds on reach rho-sampling density in terms of lfs epsilon-sampling and demonstrate that we typically reduce the required number of samples for reconstruction by more than half.

This work is part of the prototype deliverable attached to this task (D3.32) and its source code is publically available.

2.6 REAL –TIME GEOMETRY RESAMPLING FOR LIGHTING

After studying how to factorize the search for light cuts in the hierarchical model of 3D scene [Wang 2013], we more recently studied how to transform the geometric stages of the modern graphics shader-based pipeline into a goal-driven resampling machinery, that aims at extract an hierarchy-less multiscale light cache directly from a raw, dense and dynamic geometry such as the ones we encounter in capture or CAD scenarios. With *Forward Light Cuts* [Laurent 2016], the global illumination of such complex scene can be approximated with a one-bounce diffuse instant radiosity solution in real time, using the geometry shader (resp. tessellation unit) to reduce (resp. increase) the sampling rate of the scene.

2.7 OTHER RELATED WORK

As listed previously, numerous publications have contributed to various degree to WP3 production. In particular, one can notice that the Sphere-Mesh method, which contributed both to Harvest4D operators (T3.1) and algorithms (T3.2), was at the center of our more recent work on 3D+time data. With *Animated Sphere-Meshes* [Thiery 2016], we can now approximate at very coarse scale a performance (4D) capture data set using a small number of stable spheres, linked

together with a mesh structure onto which the final geometry of the approximation is obtained with a simple linear interpolation.

This work is described more in details in WP8.

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.).

Progressive Medial Axis Filtration

Noura Faraj Jean-Marc Thiery Tamy Boubekeur
Telecom ParisTech - CNRS LTCI - Institut Mines-Telecom

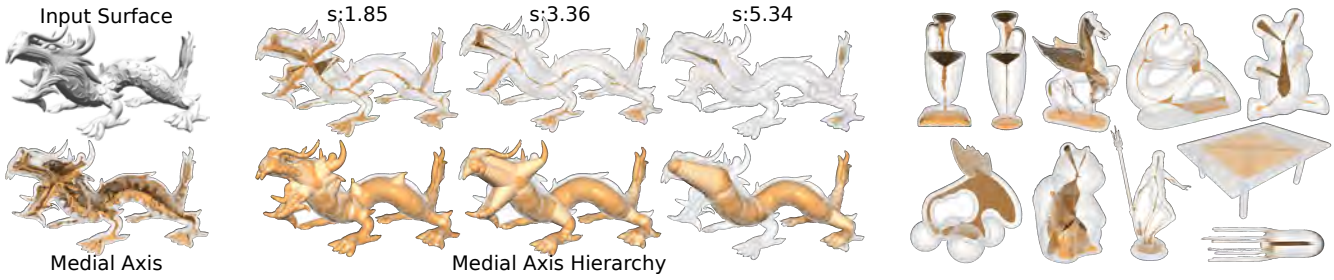


Figure 1: ***Left – Dragon Model:*** *Our progressive simplification of the medial axis filters the input shape at low scales (scale* 1.85*), and provides an efficient ordering of its features at large scales (scales* 3.36 *and* 5.34*). Top row shows the filtered medial axis, bottom row shows the polar (resp. interpolated) spheres in dark (resp. light) orange.* ***Right:*** *additional filtered medial axes.*

Abstract

The Scale Axis Transform provides a parametric simplification of the Medial Axis of a 3D shape which can be seen as a hierarchical description. However, this powerful shape analysis method has a significant computational cost, requiring several minutes for a single scale on a mesh of few thousands vertices. Moreover, the scale axis can be artificially complexified at large scales, introducing new topological structures in the simplified model. In this paper, we propose a progressive medial axis simplification method inspired from surface optimization techniques which retains the geometric intuition of the scale axis transform. We compute a hierarchy of simplified medial axes by means of successive edge-collapses of the input medial axis. These operations prevent the creation of artificial tunnels that can occur in the original scale axis transform.As a result, our progressive simplification approach allows to compute the complete hierarchy of scales in a few seconds on typical input medial axes. We show how this variation of the scale axis transform impacts the resulting medial structure.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Hierarchy and geometric transformations;

Keywords: medial axis, scale axis, shape filtering

1 Introduction

In computer graphics applications, a 3D shape is typically modelled by its boundary, for which a number of representations exist and can be classified in either explicit (e. g., meshes, splines, points) or implicit (e. g., level sets, radial basis function) schemes. Medial structures such as the Medial Axis Transform [Blum 1967] (MAT) are located at the frontier between these two main classes of representations: the shape boundary is described by an inner structure together with a function conveying locally the volume of the shape. Such medial representations are particularly useful for shape analysis, see the work of Siddiqi et al. [Siddiqi and Pizer 2008] for more details.

The MAT $\mathcal{M}_\mathcal{S}$ of a 3D surface $\mathcal{S}$ is probably the most popular medial structure. This transform computes the set of "medial" spheres, contained in $\mathcal{S}$, with at least two contact points with $\mathcal{S}$. The spheres' center form the *medial axis*, which is made of 2-dimensional sheets, curves and single points, while a radius function describes, at each point, the maximally inscribed sphere. In this paper, we focus on polygonal surface meshes for which each connected component of $\mathcal{M}_\mathcal{S}$ is composed of triangles and/or edges.

The lack of stability of the MAT (i. e., small changes in the surface usually result in drastic changes in the medial axis), prevents from using it *directly* in applications (e. g., shape matching) and a specific filtering step is usually necessary. One popular method for this filtering operation is to simply remove spheres based on the angle formed by their two closest boundary points w.r.t. their center [Attali and Montanvert 1996]. Alternatively, some methods rely on the circumradius of the two closest boundary points to define the importance of a sphere [Chazal and Lieutier 2005].

The Scale Axis Transform [Giesen et al. 2009] [Miklos et al. 2010] (SAT) is a third approach targetting the filtering of the medial axis and relies on a spatially-varying *importance* measure of the features of the input shape. The SAT can be summarized as follow: **(i)** computation of the medial axis $\mathcal{M}_\mathcal{S}$ of the input surface $\mathcal{S}$; **(ii)** scaling of the medial spheres of $\mathcal{M}_\mathcal{S}$ by a factor s (the main input parameter); **(iii)** extraction of the corresponding surface $\mathcal{S}'$; **(iv)** computation of the medial axis $\mathcal{M}_{\mathcal{S}'}$ of $\mathcal{S}'$; **(v)** scaling of the medial spheres of $\mathcal{M}_{\mathcal{S}'}$ by a factor $1/s$. Intuitively, a sphere of $\mathcal{M}_\mathcal{S}$ is likely to be filtered during the scaling step if a significantly bigger sphere is close-by. The importance of a feature is then defined relatively to the size of the nearby geometry. In practice, the SAT generates simplified medial axes, with coarseness controlled by the scale parameter s. The topological events that can occur during the scaling step of the SAT are of two kinds: either a scaled sphere S_1 with radius sr_1 absorbs a smaller scaled sphere S_2 of radius sr_2 (*Simplification*, see Fig. 3 left), or two distincts spheres S_1 and S_2

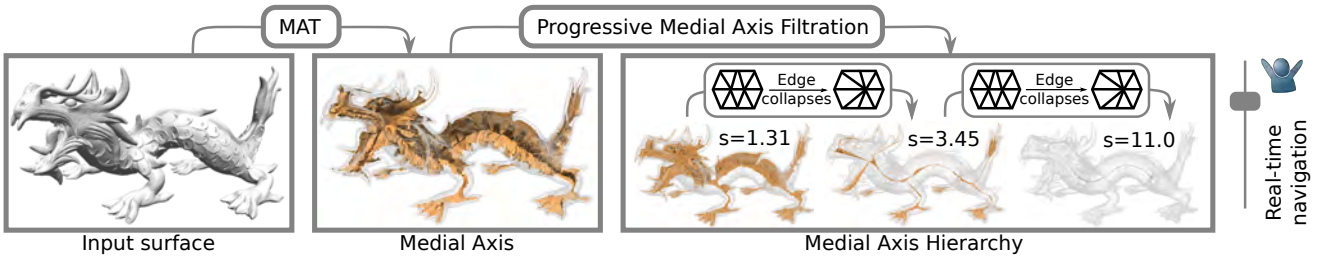


Figure 2: ***Overview:*** *Starting from a closed input surface (left), its medial axis is extracted (middle left) and simplified progressively by collapsing its edges iteratively (middle to right), constructing a medial axis hierarchy in a bottom-up fashion which can be quickly browsed.*

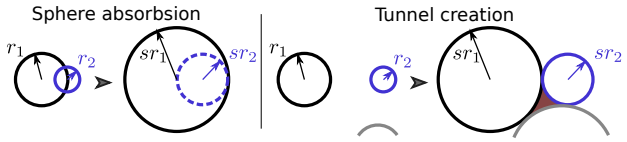


Figure 3: ***Left: Simplification.*** *Scaling the spheres by a factor s results in the absorbsion of S_2.* ***Right: Enrichment.*** *The scaling results here in the creation of a tunnel between S_1 and S_2.*

create a tunnel in the reconstructed surface when they touch each other (*Enrichment*, see Fig. 3 right). Formally, the various topological events occuring during the computation of the scale axis cannot be computed pair-wise only. Their exact computation involves all spheres to detect precisely if a grown sphere is covered (even partially) by the others, or if the point tangent to the two spheres at the creation of the tunnel is covered by another sphere (in which case no tunnel is created).

The main limitation of the SAT is threefold: first, computing a single scale requires the construction of an entire surface and the extraction of a specific medial axis, which prevents navigating easily the derived hierarchy in a reasonable amount of time; second, tunnels are likely to appear when the scale factor is too large, thus complexifying the topology of the medial axis instead of simplifying it; third, SAT spheres are not a subset of the medial axis ones.

We address these three problems by proposing an efficient medial axis simplification process which is based on the SAT importance metric, but which decimates the medial axis iteratively using an *edge-collapse* operator inspired from surface optimization [Garland and Heckbert 1997], without requiring any intermediate surface reconstruction/MAT during the process. As a result, a full hierarchy of nested medial structures can be generated in few seconds on typical inputs, and browsed interactively. As the edge-collapse is a decimation operator, no tunnels are created and each level can be expressed relatively to the next finer one and vise-versa (see Fig. 1).

2 Progressive Medial Axis Filtration

We summarize our approach in Fig. 2: starting from a closed surface $\mathcal{S}$, we extract its medial axis $\mathcal{M}_\mathcal{S}$ using the same strategy as Miklos et al. [2010] and then simplify it progressively using of successive edge-collapses. The medial axis is stored in the data structure proposed by De Floriani et al. [2004], that allows to collapse edges of a non-manifold mesh efficiently. The resulting nested hierarchy guarantees that each of its levels is a simplification of the previous one and can be browsed interactively by the user.

The metric that guides the filtration focuses on the *Simplification* events introduced in Sec. 1 and omits the *Enrichment* events.

In the following, $\mathbf{v}_i$ denotes a vertex of $\mathcal{M}_\mathcal{S}$, and represents a medial polar sphere with center $\mathbf{p}_i$ and radius r_i; $\mathbf{e}_{ij}$ denotes an edge of $\mathcal{M}_\mathcal{S}$ linking $\mathbf{v}_i$ and $\mathbf{v}_j$. Our medial axis filtration algorithm is listed in Alg. 1, and has practical runtime and memory complexities of $\mathcal{O}(|\mathcal{M}_\mathcal{S}|\log(|\mathcal{M}_\mathcal{S}|))$.

Algorithm 1 Progressive Medial Axis Filtration.

Require: $\mathcal{M}_\mathcal{S} = \{\{\mathbf{v}_i\},\{\mathbf{e}_{ij}\}\}$ the medial axis of $\mathcal{S}$
Require: $\mathcal{Q}$ a priority queue of edges
Require: $\mathcal{R}$ an empty ordered list of edge-collapses
  for all edge $\mathbf{e}_{ij}$ **do**
    $\mathcal{Q} \leftarrow \mathbf{e}_{ij}$ with cost c_{ij}
  end for
  while $\mathcal{Q}$ **not** empty **do**
    $\mathbf{e}_{ij} \leftarrow \mathcal{Q}.top()$ with $r_i < r_j$; $\mathcal{Q}.pop()$
    if $\mathbf{v}_i$ valid **and** $\mathbf{v}_j$ valid **then**
      collapse $\mathbf{e}_{ij} \rightarrow \mathbf{v}_j$
      $\mathcal{R} \leftarrow [\mathcal{R}, [\mathbf{e}_{ij} \rightarrow \mathbf{v}_j]]$
      mark $\mathbf{v}_i$ as invalid
      for all neighbor $\mathbf{v}_k$ of $\mathbf{v}_j$ **do**
        $\mathcal{Q} \leftarrow \mathbf{e}_{jk}$ with cost c_{kj}
      end for
    end if
  end while
  return $\mathcal{R}$

One core idea of this paper is to define the cost that orders the edge collapses by the scale at which the largest sphere absorbs the other one:

$$c_{ij} = \frac{|\mathbf{p}_i - \mathbf{p}_j|}{|r_i - r_j|}$$

After inserting all possible edge collapse in $\mathcal{Q}$, we prune the element with the smallest cost iteratively and collapse the corresponding edge towards the largest sphere. Each time an edge is collapsed, the neighborhood of the created vertex is updated, and corresponding edge-collapses are inserted into the queue. Edges that were incident to the deleted vertex are invalidated in $\mathcal{Q}$ by invalidating the collapsed vertex. We stop when no edge remains in $\mathcal{M}_\mathcal{S}$.

Similar to progressive meshes [Hoppe 1996], we record the set of deleted triangles and edges at each step of the simplification, constructing the nested hierarchy as an ordered list $\mathcal{R}$ where a medial axis at level k can be updated to level $k+1$ (resp. $k-1$) using the k^{th} (resp. $k-1^{th}$) element of $\mathcal{R}$. As a result, the hierarchy can be traversed in real time, in both directions, using $\mathcal{R}$ only to obtain a specific scale s.

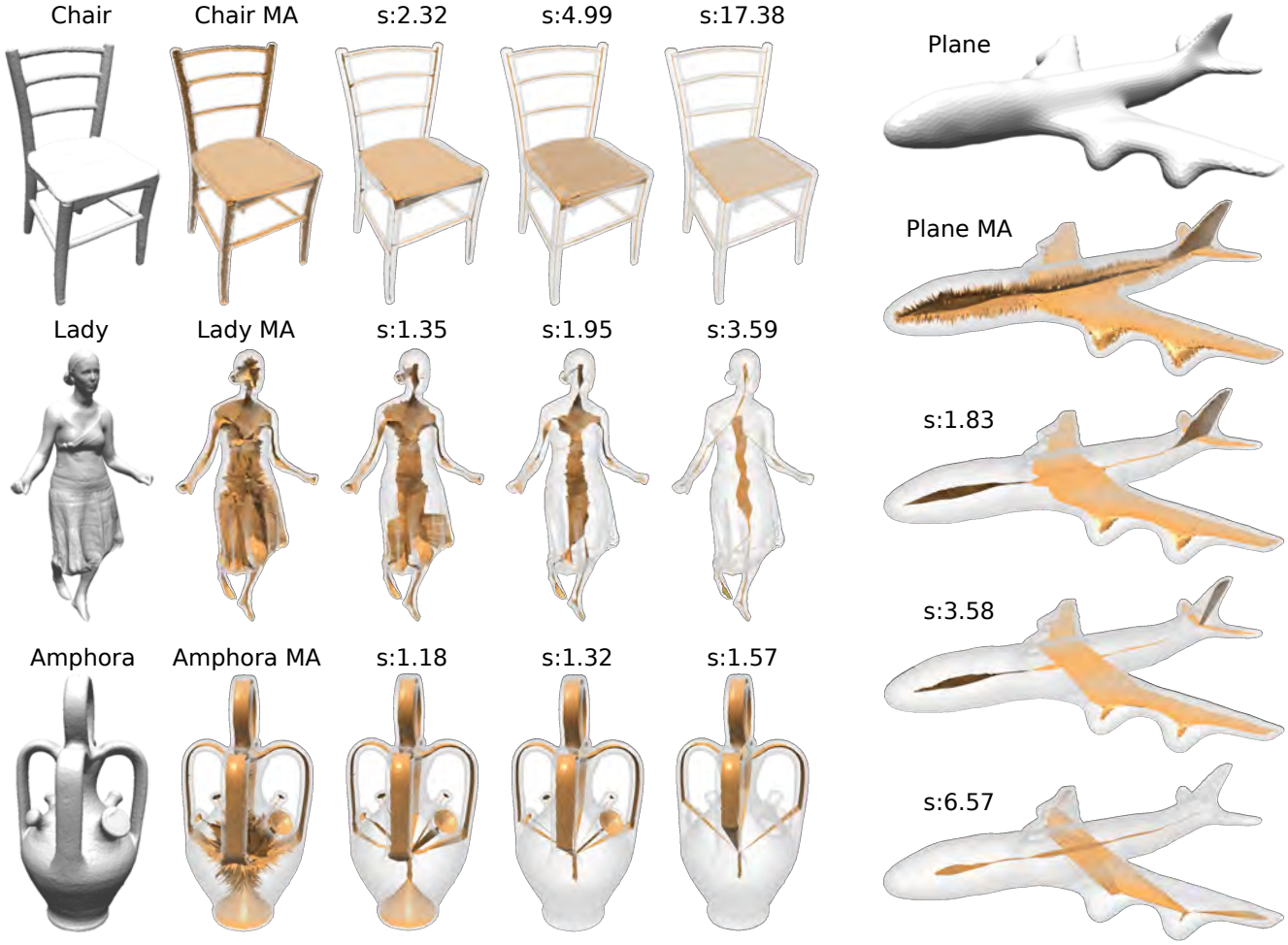


Figure 4: *Medial axis hierarchy extracted from various 3D shapes, with the input in grey, the medial axis (MA) and three different scales (s) obtained with our progressive simplication method. For illustration purpose, some medial axes are shown in wireframe.*

Input Surface (#V / #T)		Medial Axis (#V / #T / #E)	secs.	**Ours** – All Scales (secs.)	scale	**Ours** H	**SAT** H	(secs.)
Chair	(9935 / 19894)	(1014760 / 2289594 / 0)	627.90	68.23	2.32	1.02	1.20	(51.22)
					4.99	1.61	3.30	(13.1)
					17.38	2.50	12.77	(0.33)
Lady	(19990 / 39976)	(243400 / 532503 / 0)	75.70	15.20	1.35	1.65	2.11	(16.18)
					1.95	5.18	5.17	(5.39)
					3.59	7.95	7.66	(0.43)
Amphora	(14859 / 29734)	(141918 / 309993 / 0)	42.87	11.15	1.18	1.39	3.15	(13.84)
					1.32	3.72	4.52	(9.18)
					1.57	6.05	6.77	(5.43)
Plane	(6797 / 13590)	(253578 / 562181 / 0)	71.78	15.10	1.83	0.52	1.32	(17.38)
					3.58	2.46	3.59	(3.27)
					6.57	9.54	9.90	(0.77)

Table 1: *Timings for the computation of the complete simplification hierarchy, and Hausdorff distances* (H) *between surfaces reconstructed from various scales to the input surface. Timings for the computation of the Scale Axis Transform and Hausdorff distance to the input surface are given for each scale. Distances are expressed in percentages of the object's bounding box diagonal. All timings are expressed in seconds. The corresponding models are shown in Fig. 4.*

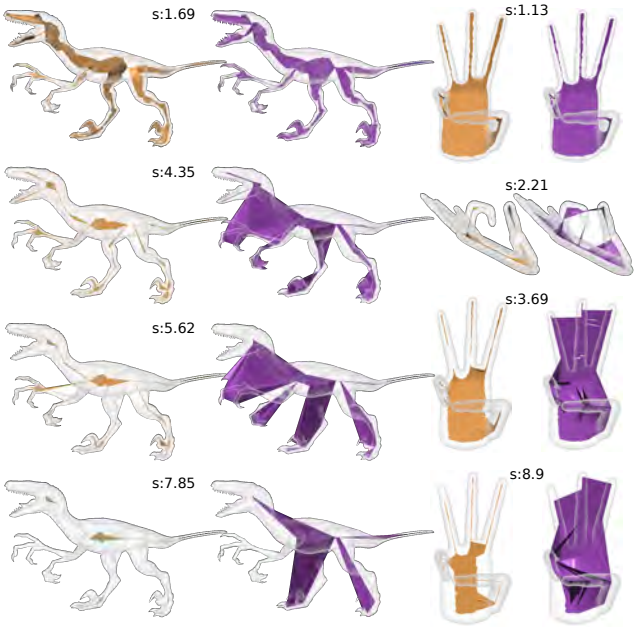


Figure 5: *Side by side visual comparison with the Scale Axis Transform (in purple).*

3 Results

Fig. 4 shows various medial axes simplified at several scales. Timings of computation of the whole hierarchy are reported in Tab. 1, along with the timings of computation of the Scale Axis Transform for the visualized scales. We also provide Hausdorff distances to the input surface for both methods. Since our technique is meant to filter large parts of the medial axis at large scales, these are given not to assess the quality of the simplification but rather to describe the size of the features that were removed.

The ability to navigate through the complete hierarchy in real-time allows the user to identify the key scales at which large features are filtered. Those values are impossible to predict beforehand, as illustrated by their variability in the presented examples.

As shown in Tab. 1, the computation of our all-scales simplification nested hierarchy is of the same order than the computation of a SAT for a single scale. Traversing the hierarchy and updating the medial axis simplification level is done in real-time on an Intel Core2 Duo running at 2.5 GHz with 4GB of main memory: going from 1 to 100.000 spheres takes a few milliseconds.

In Fig. 5 we illustrate the main differences with the SAT. Our progressive medial axis filtration behaves similarly to the SAT at low scales, but allows to filter features of the input medial axis at very large scales (10-30), for which the SAT does not provide useful information on the input shape [Miklos et al. 2010]. Even at medium scales, the SAT complexifies the shape instead of simplifying it (e. g., unwanted tunnels in the Hand model in Fig. 5–2^{nd} row).

The spheres contained in our simplified medial axes are a subset of the polar spheres of the input medial axis. The primitives linking them (triangles, edges) are however not part of the input medial axis. Similarly to the SAT, our simplified medial axes can cross the input surface at very large scales. Nonetheless, this behavior is reduced with the proposed approach (see legs of the Raptor model in Fig. 5 – scales 5.62 and 7.85). Last, on the contrary to the SAT, our approach is free of computational parameters.

4 Conclusion

We have presented a technique for computing a progressive filtration of the medial axis, building upon the spatially-varying importance classification of the medial axis features introduced by the Scale Axis Transform [Miklos et al. 2010]. Our simplification process requires the computation of a single medial axis only, and progressively simplifies it using iterative edge-collapses, ordered by this importance classification, until no edge remains. The output of our technique is a nested hierarchy of medial structures that can be browsed interactively. Compared to the scale axis transform, a large number of level-of-detail medial structures can be quickly extracted, while we ensure the simplification of the medial axis at each step. Last, the information carried out by large simplification scales is pertinent and the algorithm is free of computational parameter.

Acknowledgements We thank Bálint Miklós for providing his implementation of the SAT. The input models are provided by the Max Planck Institute, Princeton and Aim-at-Shape. This work has been partially funded by the European Commission under contract FP7-323567 Harvest4D, by the *Chaire MODIM* of Telecom ParisTech and by the ANSES "ACTE" project.

References

ATTALI, D., AND MONTANVERT, A. 1996. Modeling noise for a better simplification of skeletons. In *Image Processing, 1996. Proceedings., International Conference on*, vol. 3, IEEE, 13–16.

BLUM, H. 1967. A Transformation for Extracting New Descriptors of Shape. In *Models for the Perception of Speech and Visual Form*, W. Wathen-Dunn, Ed. MIT Press, Cambridge, 362–380.

CHAZAL, F., AND LIEUTIER, A. 2005. The λ-medial axis. *Graphical Models 67*, 4, 304–331.

DE FLORIANI, L., MAGILLO, P., PUPPO, E., AND SOBRERO, D. 2004. A multi-resolution topological representation for non-manifold meshes. *Computer-Aided Design 36*, 2, 141–159.

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.

GIESEN, J., MIKLOS, B., PAULY, M., AND WORMSER, C. 2009. The scale axis transform. In *Proceedings of the Symposium on Computational geometry*, 106–115.

HOPPE, H. 1996. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, 99–108.

MIKLOS, B., GIESEN, J., AND PAULY, M. 2010. Discrete scale axis representations for 3d geometry. *ACM Transactions on Graphics (TOG) 29*, 4, 101.

SIDDIQI, K., AND PIZER, S. M. 2008. *Medial representations: mathematics, algorithms and applications*, vol. 37. Springer.

Author's Draft. The final version will be presented at the Eurographics Symposium on Rendering and published a spcial issue of the Computer Graphcis Forum journal. 2013

Volume 32 (2013), Number 4

Factorized Point Based Global Illumination

Beibei Wang[+*] Jing Huang[*] Bert Buchholz[*#] Xiangxu Meng[+] Tamy Boubekeur[*]

[*]Institut Mines-Telecom; Telecom ParisTech; CNRS LTCI [+]Shandong University [#]NYU Polytechnic Institute

Abstract

The Point-Based Global Illumination (PBGI) algorithm is composed of two major steps: a caching step and a multiview rasterization step. At caching time, a dense point-sampling of the scene is shaded and organized in a spatial hierarchy, with internal nodes approximating the radiance of their subtrees using spherical harmonics. At rasterization time, a microbuffer is instantiated at the unprojected position of each image pixel (receiver). Then, a view-adaptive level-of-detail of the scene is extracted in the form of a tree cut and rasterized in the receiver's microbuffer, solving for visibility using a local variant of the z-buffer. Finally, the pixel color is computed by convolving its filled microbuffer with the surface BRDF. This noise-free indirect lighting method is widely used in the industry and captures several critical lighting effects, including ambient occlusion, color bleeding, (indirect) soft-shadows and environment lighting. However, we observe a large redundancy in this algorithm, both in cuts and receivers'microbuffers, which stems from their relatively low resolution. In this paper, we propose an evolution of PBGI which exploits spatial coherence to reduce these redundant computations. Starting from a similarity-based variational clustering of the receivers, we compute a single tree cut and rasterize a single microbuffer for each cluster. This per-cluster microbuffer provides a faithful approximation of the incident radiance for distant nodes and is composited over a receiver-specific microbuffer rasterizing only the closest nodes of the cluster's cut. This factorized approach is easy to integrate in any existing PBGI implementation and offers a significant rendering speed-up for a negligible and controllable approximation error.

1. Introduction

The visual impact of global illumination (GI) in a synthesized picture is the sum of a number of lighting effects stemming from indirect light bounces. Among them, one-bounce diffuse effects, such as ambient occlusion, directional occlusion, color bleeding and indirect soft shadows, carry a large portion of the visual realism that typical GI solutions bring. Point-based global illumination (PBGI) is a popular rendering technique which captures such a subset of GI effects for a moderate amount of time and is intensively used in special effects and computer animation productions. This GI approximation model can be seen as a generalized forward rendering method which combines a fast adaptive approximation of the scene with a multiview rasterization. The resulting algorithm is noise-free, amenable to a parallel implementation and can even be extended to other GI effects (e.g., multiple bounces), although still away from a full GI solution, in particular when it comes to specular indirect phenomena (i.e., caustics).

1.1. Basic Algorithm

PBGI [Chr08] runs in a two-step process: a caching step and a multiview rasterization step. At caching time, the scene is densely point-sampled (e.g., using Poisson disks), the points are shaded from the light sources – accounting for direct shadows only – and structured in a hierarchical data structure (e.g., octree, BSH). This tree is constructed bottom-up from the shaded points, with internal nodes carrying approximations of their related sub-trees (e.g., bounding sphere, normal cone, low-degrees spherical harmonics modeling the outgoing diffuse radiance).

At rasterization time, each pixel of the final picture is shaded using a so-called *microbuffer*, which is a small hemispherical RGBZ image instantiated at the unprojected position of the pixel (or *receiver*) in the scene. For each microbuffer, a specific level-of-detail (LoD) of the scene is extracted in the form of an adaptive cut in the PBGI tree. The resulting nodes are rasterized in the microbuffer using a local variant of the z-buffer algorithm to solve for visibility. The filled microbuffer is finally convolved with the point's BRDF to shade the pixel.

The two key ideas of this algorithm are (i) the point's hierarchy, which acts as an economic substitute to the actual scene when it comes to the many adaptive LoDs which have to be extracted; (ii) the microbuffers, which extend the concept of rasterization to a per-pixel/receiver level.

1.2. Redundancy Issue

Looking back at the rasterization step, we observe that a specific cut is computed from the entire scene for each single receiver. However, the resolution of their microbuffers is typically low (from 4x4 to 64x64 in practice) which immediately translates into tree cuts having a large number of coarse nodes, therefore being highly similar for nearby receivers. As we will show later, this abundant redundancy has a significant impact on the overall rendering time.

1.3. Overview

We tackle this problem by exploiting the microbuffers'spatial coherence to factorize both cut computations and rasterizations. Our factorized PBGI technique (or FPBGI) works in three steps (see Fig. 1):

1. we cluster the receivers based on their similarity and select a per-cluster *active* receiver,
2. for each cluster, we compute a single (coarser) cut from the active receiver and rasterize it in a microbuffer shared by all receivers of the cluster
3. for any receiver, we start the tree traversal from its cluster cut and rasterize only the newly added (i.e., closer) nodes in a receiver-specific microbuffer, which is composited with the cluster one before final BRDF convolution.

As a result, a large part of tree traversals and cut rasterizations are factorized among nearby receivers, which leads to an overall rendering speed-up ranging from 2x to 4x on the typical scenes illustrating this paper.

2. Previous Work

PBGI. PBGI was first introduced by Christensen [Chr08], who proposed the idea of microbuffers and exploited the notion of point-based substitutes introduced by Bunnell [Bun05] for real time ambient occlusion and indirect illumination. Ritschel et al. [REG*09] then replaced cube microbuffers with 2D Lambert-warped ones, introducing importance sampling to PBGI together with an efficient GPU implementation. Holländer et al. [HREB11] later improved on the fine-grained parallelism of the adaptive cut computation by pairing nodes and receivers in a low-scale GPU data amplification mechanism. The cut definition itself has been addressed by Maletz and Wang [MW11] who used an importance-driven point projection based on an initial clustering, by Wang et al. [WMXS11] who grouped together close points with similar normals and computed average cuts for a subset of the receivers, and by Tabellion [Tab12] who recently exposed a set of cut picking algorithms suitable for HDR imaging. The PBGI accuracy entirely depends on the density of the initial sampling and the related memory issue has been tackled by Kontkanen et al. [KTO11], who proposed an out-of-core framework for PBGI with cache-coherent tree construction and traversal. Buchholz and Boubekeur [BB12] proposed an in-core solution to this problem, learning a reduced set of node data vectors in high dimension and quantizing all tree nodes against the resulting look-up table.

Coherence in rendering. Coherence through some form of "reuse" mechanism has been widely studied in GI research. Such techniques try to avoid redundancy at different levels of the GI solution computation, including irradiance, radiance, shading and even tree-cuts in a closer context to ours. Ward et al. [WRC88] reused illumination computation by computing scalar (diffuse) irradiance on a subset of pixels and interpolating for the others, eventually using gradients [WH92] for smoother results. Wang et al. [WWZ*09] used k-means to subsample receiving points and interpolate irradiance, reaching interactive framerates but missing small geometric details. *Radiance Caching* [KGPB05] overcomes the limitation to diffuse reflectance by storing incoming radiance as a directional function, interpolating it between pixels and convolving with the BRDF for every pixel. Closer to PBGI methods, Holländer et al. [HREB11] proposed a **time**-coherent cut update, together with a lazy scheme bounding the amount of time dedicated to this update. Our approach is inspired by this method, but acts in the spatial domain.

Near-far decomposition. The idea of near-far irradiance decomposition has been previously studied in the context of hardware ambient occlusion [SA07] and final gathering [AFO05]. Acting in a PBGI context, our approach differs in the sense that the near-far split is entirely formulated through the cluster/receiver cut, the far component being shared by numerous receivers.

3. Factorized Point Based Global Illumination

3.1. Variational Receiver Clustering

Our basic assumption is that receivers with similar positions and normals have similar cuts: we propose to model this position/normal similarity by computing a variational clustering of the receivers based on a specific metric $\mathcal{D}$. To ease parallel computation, we start by regularly tiling the image space and work independently on each tile. Within a tile, we group spatially coherent receivers in k clusters using a variant of the *k-means* algorithm:

1. we initialize k centers from randomly selected receivers in the tile,
2. we cluster the tile's receivers by associating each of them to its *closest* center w.r.t. $\mathcal{D}$
3. we update clusters' centers and restart in (2).

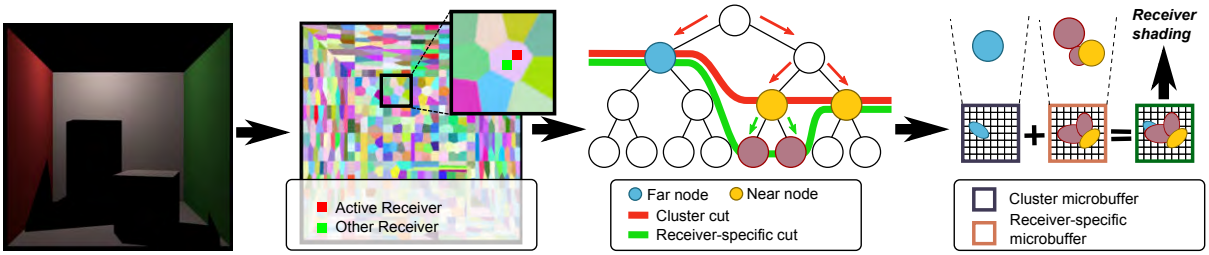


Figure 1: ***Principle***. *Starting from a tiled set of image pixels/receivers (left), we perform a variational clustering based on positional and normal similarity (middle left). For a given cluster, we compute a shared cut (middle right, red) later reused by each individual receiver to further refine their own cuts (middle right, green). The far nodes of the cluster are rasterized into a shared cluster microbuffer (right, purple) and refined nodes (added on a per-receiver basis) are rasterized in a receiver-specific microbuffer (right, orange), which is composited into one cluster for final BRDF convolution (pixel indirect shading).*

We perform this procedure for a prescribed number of iterations and search, for each cluster, the closest receiver to the resulting center: in the following, we call it the *active* receiver of a cluster.

Following Cohen-Steiner et al. [CSAD04], we define our position/normal metric $\mathcal{D}$ as a Sobolev summed metric:

$$\mathcal{D}(\mathbf{x},\mathbf{c}) = ||\mathbf{p_c} \quad \mathbf{p_x}||^2 + \alpha||\mathbf{n_c} \quad \mathbf{n_x}||^2$$

with $\mathbf{x}$ being a receiver, $\mathbf{c}$ a cluster center, $\mathbf{p}$ (resp. $\mathbf{n}$) their position (resp. normal) in $\mathbb{R}^3$. The weight α trades cluster flatness for spatial extent. We typically set it to the length of the tile's receivers' bounding box diagonal. Last, at each iteration, the center position and normal of a cluster $\mathcal{C}$ are updated as follows:

$$\mathbf{p_c} = \frac{\sum_{\mathbf{x}\in\mathcal{C}} \mathbf{p}_x}{card(\mathcal{C})} \quad \mathbf{n_c} = \frac{\sum_{\mathbf{x}\in\mathcal{C}} \mathbf{n_x}}{||\sum_{\mathbf{x}\in\mathcal{C}} \mathbf{n_x}||}$$

3.2. Cluster Cut and Microbuffer

Within a cluster $\mathcal{C}$, the factorized workload among receivers takes the form of a single shared cut and a single microbuffer which are computed w.r.t. the *active* receiver $\mathbf{x}_\mathcal{C}$.

In the next step of the rasterization phase, we start by traversing the PBGI tree from the root for $\mathbf{x}_\mathcal{C}$ but stop early to produce a cut which is coarser than required in the vicinity of $\mathbf{x}_\mathcal{C}$. Indeed, we assume that the significant difference between two nearby microbuffers only appears at fine scale (i.e., closer nodes) and deal with it later.

During the top-down PBGI tree traversal, we use a *far/near* classification of the tree's nodes based on a measure γ for each node/receiver pair: far nodes ($\gamma > \varepsilon$) are traversed as usual, while near nodes ($\gamma \leq \varepsilon$) stop the traversal immediatly. The node/receiver measure is defined as $\gamma = \frac{r}{d}$ with r being the cluster's radius r and d the distance between the node and $\mathbf{x}_\mathcal{C}$. The resulting cluster cut contains two types of nodes: *far* nodes, which are rasterized in the shared cluster microbuffer, and *near* nodes, which will be concurrently refined for each individual receiver in the next step. At this stage, the cluster microbuffer already carries the distant irradiance shared by all cluster receivers.

3.3. Receiver Cut, Microbuffer and Shading

In the last part of our algorithm, we process each individual receiver in parallel. For a given receiver, we compute its specific cut starting from the cluster cut (instead of the tree's root) and traversing the hierarchy down to the classical microbuffer-dependent solid angle threshold. Only the newly added nodes to the cut are marked as *refined*. Once the cut is completed, we rasterize its *refined* and *near* nodes into a receiver-specific microbuffer. Basically, only the closer nodes are rasterized and we obtain a sparse microbuffer.

Last, we composite this receiver microbuffer with the corresponding cluster one, using the depth component of both microbuffers to properly cull the microbuffer pixels which are hidden by this combination. The resulting composited microbuffer is finally convolved with the receiver's BRDF to shade the receiver/pixel.

4. Results

We implemented our technique in the Mitsuba Renderer [Jak10], with the initial point set being generated using Poisson Disk sampling. Comparisons are performed against the original PBGI algorithm [Chr08] and performances are measured on a 2.67GHz Intel i7 (8 cores) with 9GB of main memory. Images are rendered with one-bounce indirect illumination at a 1280×1000 pixels resolution (except for the Cornell Box, at 1024×1024) with 32×32 tiles.

In all comparisons, we measure numerical differences with the Mean Squared Error (MSE) and visual differences by counting the number of Perceptually Different Pixels (PDP), as proposed by Yee [Yee04]. This perceptual error metric acts in the Lab space and is plotted in black and blue.

Scene	Points	Individual Timings					Total Time		Error	
		Clustering	Cut Computation		Micro-Rasterization		Full Rendering			
		Time(s)	*PBGI*	*FPBGI*	*PBGI*	*FPBGI*	*PBGI*	*FPBGI*	*PDP*	*MSE*
CornellBox	88.88K	0.68s	2.46m	0.65m	1.73m	1.19m	5.72m	2.60m	103	8.79e-6
Bunny	1.00M	0.87s	2.38m	0.56m	1.55m	1.08m	4.49m	2.03m	61	1.31e-4
ItalianCity	8.38M	0.87s	3.65m	1.08m	1.23m	0.64m	5.52m	2.34m	235	8.15e-5
Sponza	16.19M	0.87s	19.71m	3.77m	5.40m	1.68m	26.72m	7.04m	15	2.51e-5

Table 1: *Performance measures.*

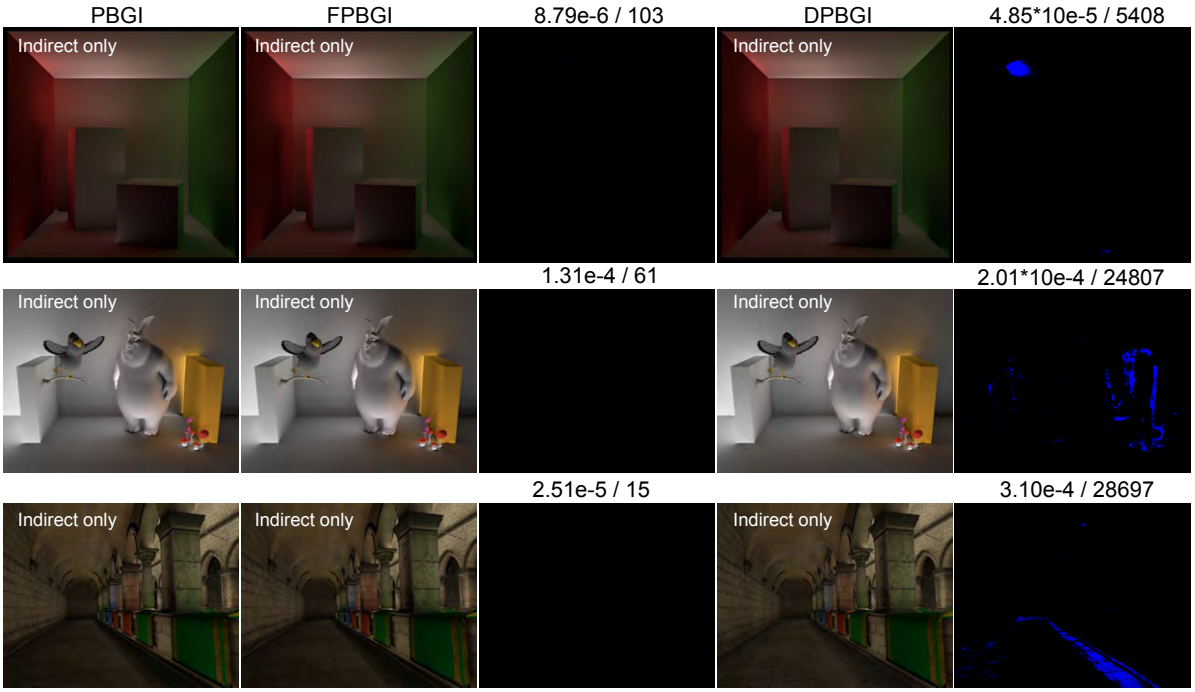


Figure 2: *Error analysis on the indirect lighting contribution for FPBGI and DPBGI against PBGI. Perceptual differences [Yee04] are plotted in black (no visible difference) and blue (visible difference). The MSE between RGB images and the number of Perceptually Different Pixels [Yee04] (PDP) are indicated in the format <MSE>/<PDP> on top of difference images.*

In Fig. 2, we compare FPBGI with the original PBGI algorithm on three different scenes. Overall, we observe a negligible error, both from a perceptual and numerical point of view. The original PBGI algorithm can indeed be trivially sped-up by reducing the resolution of the microbuffers (i.e., higher solid angle threshold in the tree traversal), which immediately translates into coarser cuts for each receiver and reduced rasterization time. Therefore, we also compare to such a *degraded* PBGI setting (DPBGI), with microbuffer resolution decreased so that the total rendering time is as close as possible to our FPBGI. In this case, DPBGI produces significantly stronger errors, with noticeable aliasing appearing.

In Table 1, we report timings and errors for the four different examples shown in Fig. 2 and Fig. 3. Here, we can assess the benefit of our factorized approach, with a speed-up ratio for the total rendering time (including BRDF evaluation and initial set up) ranging from 2.2 to 3.8 compared to the original PBGI algorithm. Looking at the specific portion of the algorithm that we target (rasterization), the speed-up ratio ranges from 3.4 to 5.2 for the cut computation and from 1.4 to 3.2 for the micro-rasterization. In all cases the receiver clustering time is negligible.

In Fig. 3, we provide a visual comparison of the final rendering (direct+indirect illumination) between a fully path-traced solution (PTS), PBGI and FPBGI. We can observe that PBGI and FPBGI provide similarly good approxima-

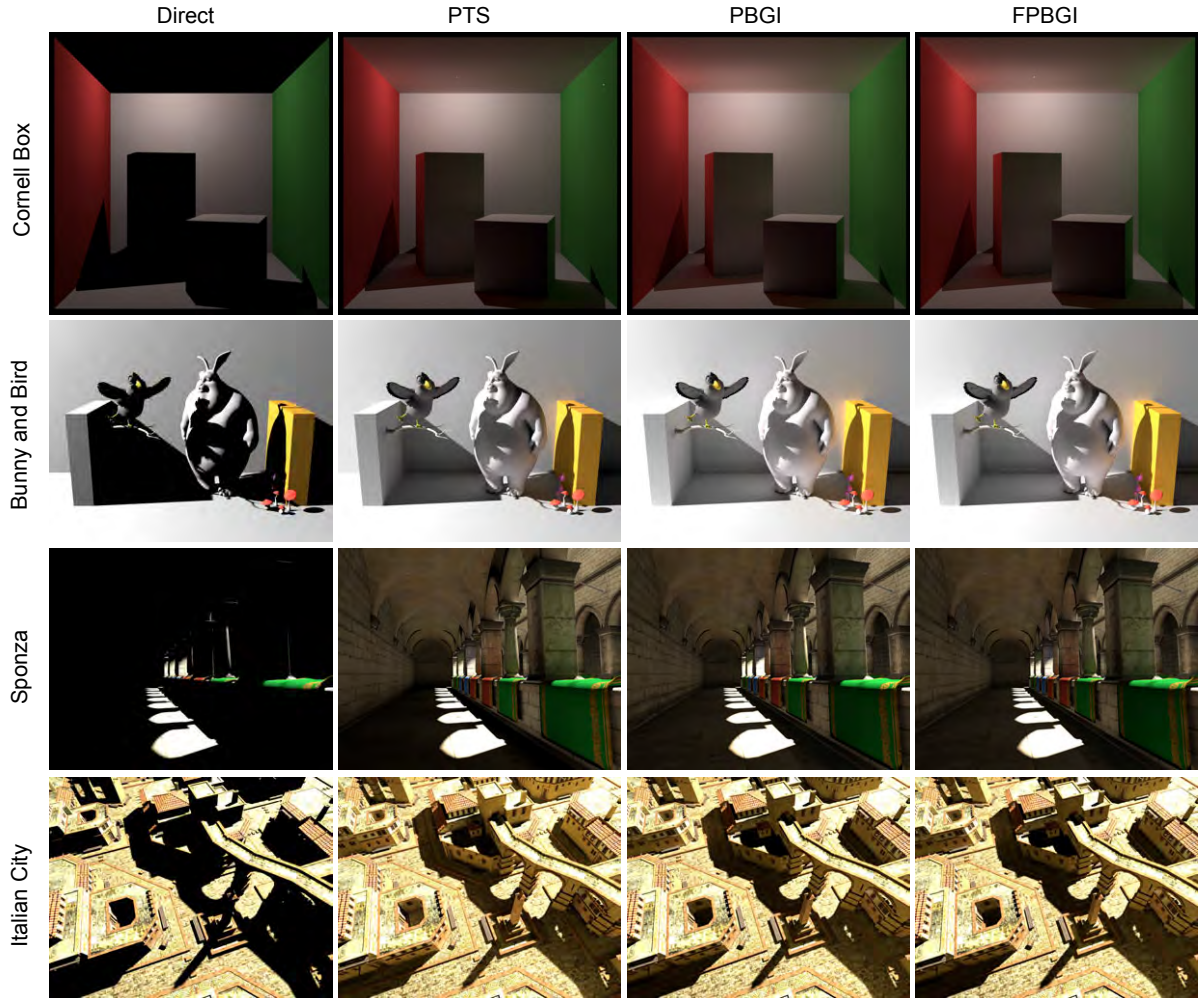


Figure 3: *Visual comparison of final renderings.*

tions of the PTS, which is typically an order of magnitude slower than FPBGI.

We also analyze the influence of the two main parameters of FPBGI: the number of clusters per-tile k and the far-near threshold ε. In Fig. 4, we illustrate their influence on the Sponza scene. We observe that the influence of k clearly dominates on the approximation quality, as measured by numerical and perceptual errors. However, looking closely at the result, we can see that, under a very small k value, large values of ε cause large visible artifacts. In Fig. 5, we plot the speed-up evolution under variations of these two parameters. We empirically determine $k = 100$ and $\varepsilon = 10e^{-2}$ as good default values for all the scenes we experimented with. Last, with visually invisible differences, FPBGI inherits the temporal coherence of PBGI: we illustrate this behavior in an accompanying video with three sequences showing animated lighting, camera and models.

Discussion. Alternatively to our approximation technique, recent approaches [REG*09,HREB11] propose to maximize the fine-grained parallelism of the PBGI algorithm in order to map it efficiently on GPU architectures. Clearly, our approach is orthogonal to such methods, but preserves the natural parallelism of PBGI. However, compared to such methods, an additional specific preliminary pass would be required to gather the shared microbuffers. As future work, at least two alternative solutions may be further developed to combine our factorization with an efficient GPU implementation: first, the typical number of clusters is large enough to load the numerous GPU computing units with cluster cuts computations using a naive implementation (i.e., one thread per-cluster first, then one thread per-receiver); sec-

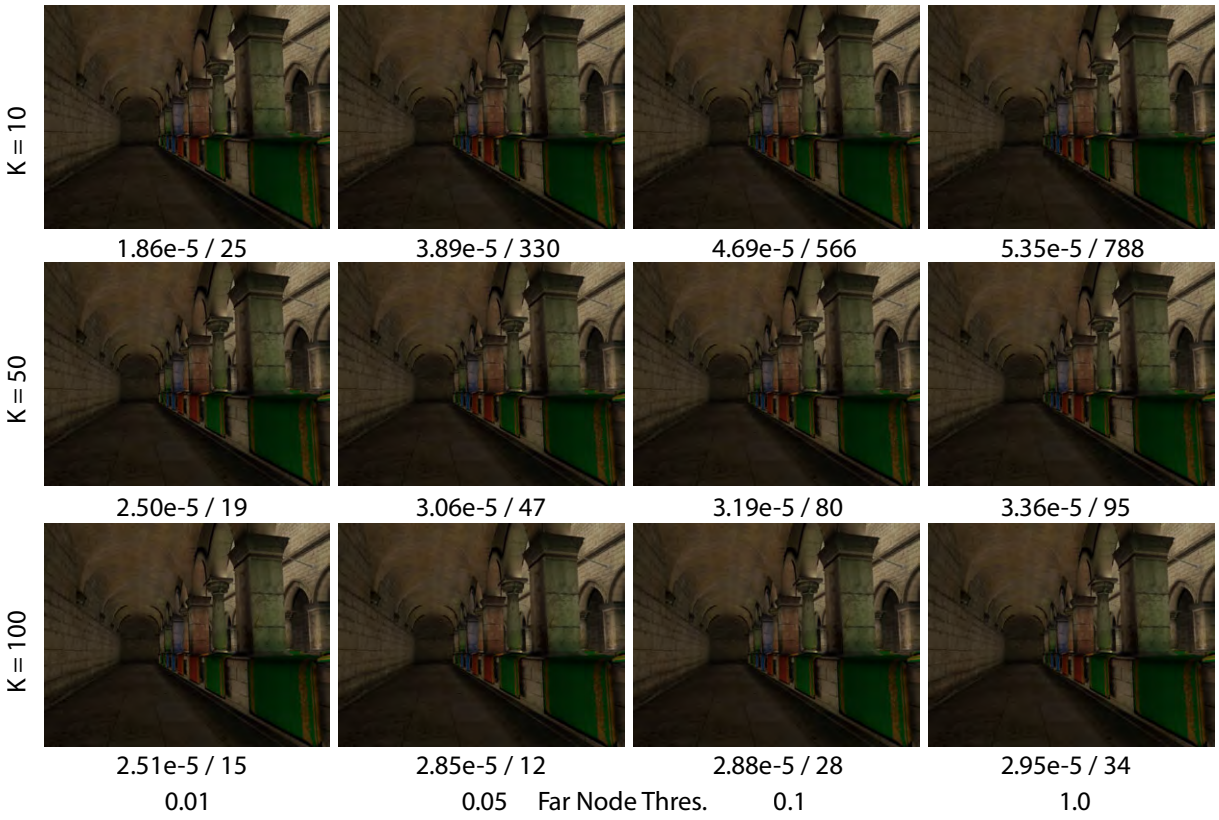


Figure 4: *Parameter influence with <MSE>/<PDP> to the PBGI solution for each pair.*

ond, a more evolved solution could use the two-layer GPU computing model (blocks and threads) to make threads belonging to the same block define concurrently the cluster cut and microbuffers in shared memory before synchronizing them and letting them processing their receiver-specific components, using the ManyLoDs algorithm [HREB11] at both stages.

Interestingly, Holländer et al. [HREB11] proposed an acceleration exploiting *temporal* coherence only, the *lazy* scheme which reuses cuts over consecutive frames, while our factorized approach exploits *spatial* coherence. Combining both approaches could help exploiting *spatio-temporal* coherence to its full extent.

Our approach has two main limitations. First, the cluster cut may be over-conservative and the resulting per-receiver cut can be too refined. Although this does not influence the rendering quality, this remains sub-optimal. A solution could be to allow receivers to "walk-up" the tree while refining their cut. Second, the two main parameters of the algorithm have fixed values. These values could be optimized dynamically and vary spatially by using the PBGI tree to perform a quick scene analysis. Last, our approach can be seen as a simplified *hierarchy* of receivers. It would then be interesting to determine how to reformulate the PBGI algorithm to rasterize, adaptively, the PBGI tree against the receiver/pixel one to reach a fully adaptive solution.

5. Conclusion

We have proposed a factorized evolution of the PBGI algorithm which exploits spatial coherence to significantly speed up the computation of indirect diffuse illumination effects. By combining an initial variational clustering with per-cluster cuts and microbuffers, we showed that the individual receiver workload boils down to a local geometry rasterization followed by a microbuffer compositing. As a result, we obtain a speed-up ranging from 2x to 4x, without any visible image degradation. Our approach is easy to implement in any PBGI framework and has a reduced set of intuitive control parameters.

Acknowledgements. This work has been partially supported by the China Scholarship Council and 863 Program of China under Grant No. 2012AA01A306, the European Commission under contract FP7-287723 REVERIE and the ANR iSpace&Time project.

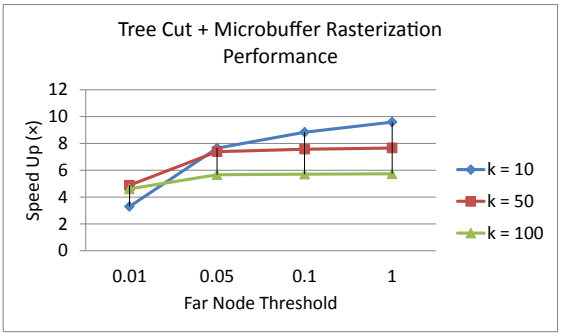


Figure 5: *Parameters influence on the speed-up.*

References

[AFO05] ARIKAN O., FORSYTH D. A., O'BRIEN J. F.: Fast and detailed approximate global illumination by irradiance decomposition. In *ACM SIGGRAPH 2005* (2005), pp. 1108–1114. 2

[BB12] BUCHHOLZ B., BOUBEKEUR T.: Quantized point-based global illumination. *Comp. Graph. Forum (Proc. EGSR 2012) 31*, 4 (2012), 1399–1405. 2

[Bun05] BUNNELL M.: Dynamic ambient occlusion and indirect lighting. *GPU Gems 2* (2005), 223–233. 2

[Chr08] CHRISTENSEN P.: *Point-based approximate color bleeding*. Tech. Rep. 08-01, Pixar Technical Notes, 2008. 1, 2, 3

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. *ACM Trans. Graph. 23*, 3 (2004), 905–914. 3

[HREB11] HOLLÄNDER M., RITSCHEL T., EISEMANN E., BOUBEKEUR T.: Manylods: Parallel many-view level-of-detail selection for real-time global illumination. *Comp. Graph. Forum (Proc. EGSR 2011) 30*, 4 (2011), 1233–1240. 2, 5, 6

[Jak10] JAKOB W.: Mitsuba renderer. http://www.mitsuba-renderer.org/, 2010. 3

[KGPB05] KRIVANEK J., GAUTRON P., PATTANAIK S., BOUATOUCH K.: Radiance caching for efficient global illumination computation. *IEEE TVCG 11*, 5 (2005), 550–561. 2

[KTO11] KONTKANEN J., TABELLION E., OVERBECK R. S.: Coherent out-of-core point-based global illumination. In *Comp. Graph. Forum (Proc. EGSR 2011)* (2011), pp. 1353–1360. 2

[MW11] MALETZ D., WANG R.: Importance point projection for GPU-based final gathering. *Comp. Graph. Forum 30*, 4 (2011), 1327–1336. 2

[REG*09] RITSCHEL T., ENGELHARDT T., GROSCH T., SEIDEL H.-P., KAUTZ J., DACHSBACHER C.: Micro-rendering for scalable, parallel final gathering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia 2009) 28*, 5 (2009). 2, 5

[SA07] SHANMUGAM P., ARIKAN O.: Hardware accelerated ambient occlusion techniques on GPUs. In *ACM I3D* (2007), pp. 73–80. 2

[Tab12] TABELLION E.: Point-based global illumination directional importance mapping. In *ACM SIGGRAPH Talk* (2012). 2

[WH92] WARD G. J., HECKBERT P.: Irradiance gradients. In *Eurographics Workshop on Rendering* (1992). 2

[WMXS11] WANG B., MENG X., XU Y., SONG X.: Fast point based global illumination. In *Computer-Aided Design and Computer Graphics* (2011), pp. 93–98. 2

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A ray tracing solution for diffuse interreflection. In *ACM SIGGRAPH Computer Graphics* (1988), vol. 22, pp. 85–92. 2

[WWZ*09] WANG R., WANG R., ZHOU K., PAN M., BAO H.: An efficient GPU-based approach for interactive global illumination. In *ACM Trans. Graph. (Proc. SIGGRAPH 2009)* (2009), pp. 91:1–91:8. 2

[Yee04] YEE H.: A perceptual metric for production testing. *Journal of Graphics Tools 9*, 4 (2004), 33–40. 3, 4

Morton Integrals for High Speed Geometry Simplification

Hélène Legrand & Tamy Boubekeur
Telecom ParisTech - CNRS LTCI - Institut Mines-Telecom

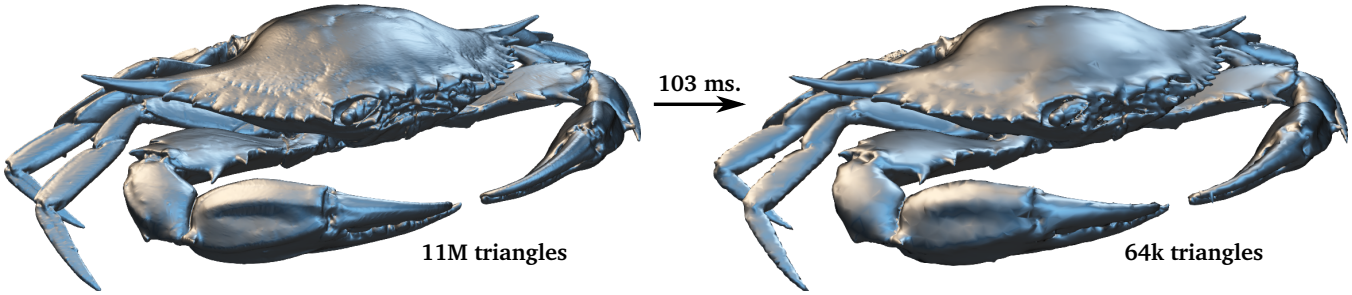


Figure 1: *Adaptive simplification using our algorithm: even beyond 10M tri., our parallel approach remains nearly interactive.*

Abstract

Real time geometry processing has progressively reached a performance level that makes a number of signal-inspired primitives practical for on-line applications scenarios. This often comes through the joint design of operators, data structure and even dedicated hardware. Among the major classes of geometric operators, filtering and super-sampling (via tessellation) have been successfully expressed under high-performance constraints. The subsampling operator i.e., adaptive simplification, remains however a challenging case for non-trivial input models. In this paper, we build a fast geometry simplification algorithm over a new concept: Morton Integrals. By summing up quadric error metric matrices along Morton-ordered surface samples, we can extract concurrently the nodes of an adaptive cut in the so-defined implicit hierarchy, and optimize all simplified vertices in parallel. This approach is inspired by integral images and exploits recent advances in high performance spatial hierarchy construction and traversal. As a result, our GPU implementation can downsample a mesh made of several millions of polygons at interactive rates, while providing better quality than uniform simplification and preserving important salient features. We present results for surface meshes, polygon soups and point clouds, and discuss variations of our approach to account for per-sample attributes and alternatives error metrics.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry & Object Modeling—Geometric Algorithms I.3.5 [Computer Graphics]: Computational Geometry & Object Modeling—Hierarchy and Geometric Transformations; I.3.1 [Computer Graphics]: Hardware Architecture—Parallel Processing; I.3.1 [Computer Graphics]: Hardware Architecture—Graphics Processors;

Keywords: mesh simplification, GPU algorithms, adaptive clustering, Morton code.

1 Introduction

Modern 3D capture pipelines allow acquiring real world geometry quickly and accurately. The increasing data generation speed has motivated a large number of research projects, to provide visual computing systems with geometric operators able to process this data in real time. In these constrained scenarios, approximations and a controlled loss of quality are preferable to exceeding the time limit. This trend in high performance geometry processing had a number of successes, with now solutions for performing some of the most critical processing steps on-the-fly, for non-trivial input and on commodity hardware. This includes for instance adaptive feature-preserving filtering [Adams et al. 2009] and adaptive smooth tessellation [Pixar 2013] (i.e., upsampling) which are now supported by high-performance data structures and even dedicated hardware and programmable graphics stages.

The case of adaptive geometry simplification (e.g. mesh simplification or point cloud subsampling) remains however challenging. The key problem in this case lies in the data compaction mechanism, which does not cope naturally with a fine-grained parallel computing environment. Although two decades of research have progressively led to simplification algorithms offering a controllable trade-off between output surface quality and computational effort, we are still far from high quality, real time simplification for non-trivial (e.g., multi-millions) sample sets, and current aggressive simplification methods have a limited range of applications. However the interactive and real time scenarios we target may benefit from higher quality simplifications, with applications such as interactive display on mobile devices, dynamic multiview rendering or live broadcast of 3D captured data (e.g., 3D camera). In such applications, the resulting geometry may be discarded after a short amount of time, required immediately after full resolution data generation or needed on-demand, in multiple versions under a wide variety of simplification ratios. In this case, it is crucial to provide instantly visually good simplifications.

Indeed, a key aspect of high performance mesh processing, largely exploited for filtering and tessellation, is the parallel scalability of the operators and their ability to run on the graphics processor unit (GPU). In the context of adaptive simplification/downsampling, we make two observations. First, fast adaptive sampling often relies on an underlying hierarchical data structure which is costly to generate and maintain in a parallel environment. Second, with such a hierarchy in hand, simplifying the mesh often means extracting a particular, error-driven "cut" in the tree structure. This operation, either with a bottom-up of a top-down approach, does not map triv-

ially on a fine-grained parallel architecture.

In this paper, we propose a step toward higher quality real time surface optimization, in the form of a fast simplification method which provides adaptive error-driven mesh samplings, while staying within the real time rates required by our target applications for typical input/output sizes (see Fig. 1). We address the hierarchy problem using the *Morton order* of the input surface samples, over which we compute a one-dimensional sum of the geometric cost associated with each surface sample. This intermediate representation allows for an efficient kd-tree construction and later for concurrent evaluation of all the geometric errors to be estimated on all the nodes of the tree. It also enables the parallel processing of both leaves and inner nodes, for a constant and predictable per-node cost. We use the Quadric Error Metric [Garland and Heckbert 1997] as our basic cost measure for two reasons: first, it remains the state-of-the-art measure for general simplification. Second, quadrics have the nice property to sum to quadrics, which allows us to address the problem of multi-level error computation using the same principle as *integral images* (a.k.a. summed area tables [Crow 1984]). The hierarchy defined by the Morton codes of the samples requires a single sort to be performed at the beginning of the algorithm. Although the resulting space clustering is less accurate than a pure error driven tree refinement/aggregation (e.g., BSP tree), it significantly improves over uniform GPU clustering techniques. As a result, our algorithm can simplify large meshes, polygons soups and even point clouds in real time, accounting for the geometric features, but also for additional attributes on the surface. We present experiments on a collection of models and discuss possible evolutions of our approach.

2 Previous work

Most mesh simplification methods define an objective optimization criterion, with a metric which measures the error caused by the simplification in the form of some distance between the original object and the simplified output. Simplification algorithms usually fall into two categories : *iterative simplification* and *vertex clustering*.

Iterative methods [Hoppe et al. 1993; Garland and Heckbert 1997] progressively reduce the number of primitives of the mesh by performing, at each step, a local simplification operation causing the smallest error according to the chosen metric. These methods usually lead to high quality output meshes but are difficult to parallelize efficiently due to their sequential nature. They also usually require a clean mesh connectivity which, in the context of instantaneous capture and processing, can hardly be guaranteed. Although their method does not reach real time rates, Grund et al. [2011] propose a parallel simplification algorithm based on this approach.

We focus on clustering methods, which optimize for a simplified mesh at a coarser grain, by defining a partition of the mesh, computing a representative vertex/polygon for each cluster and meshing the resulting (smaller) geometric set. The choice of a particular partitioning structure has a strong impact on the overall performance of the process, with solutions including simple grids [Rossignac and Borrel 1993; Lindstrom 2000], octrees [Schaefer and Warren 2003][Lindstrom 2003][Shaffer and Garland 2005], BSP-trees [Shaffer and Garland 2001] and k-means partitions [Cohen-Steiner et al. 2004]. The representative element is again chosen to optimize a certain metric, for which popular choices include the Quadric Error Metric (QEM) [Garland and Heckbert 1997] which models the simplification cost as the sum of the squared distances from the representative point to the planes defined by the triangles of the cluster; or the $L^{2,1}$ [Cohen-Steiner et al. 2004] metric which uses the normal information to grow large flat clusters whenever possible. The final meshing step is performed by either reindexing input triangles

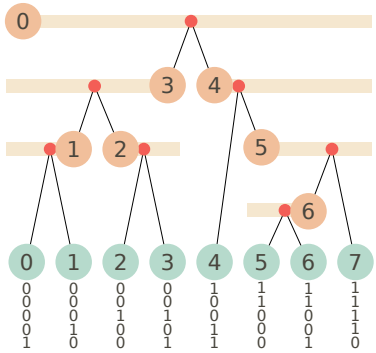


Figure 2: ***Tree layout*** *in the method of Karras. The yellow bars represent the range of leaf nodes (in green, with their Morton code) covered by the internal nodes (in orange). The red dots indicate the binary split position.*

intersecting three different clusters of the partition to the related representatives [Rossignac and Borrel 1993; Boubekeur and Alexa 2009] or generating polygons tracking clusters boundaries.

Since each cluster is (mostly) processed independently, vertex clustering methods are more adapted to parallel computing and GPU architectures. In particular, Decoro and Tatarchuk [2007] have proposed a GPU implementation of QEM-based grid simplification [Lindstrom 2000] as well as a probabilistic octree structure providing adaptivity.

Regarding the type of simplification obtained, our method falls in the same category as [Schaefer and Warren 2003], [Lindstrom 2003] and [Shaffer and Garland 2005], with an axis aligned hierarchy depending on a spatially coherent ordering of the primitives. Most of our contribution is about providing very similar results in terms of quality, while performing a fully parallel simplification at interactive to real time rates.

Hierarchical space subdivision structures provide a good trade-off between full adaptivity and grid clustering to create the initial partition. The efficient generation of such structures has been mostly studied in the context of rendering applications. In particular, Morton curves (or z-order curves) have proved to provide an efficient space parametrization supporting the hierarchy construction [Lauterbach et al. 2009; Pantaleoni and Luebke 2010; Garanzha et al. 2011]. More precisely, a Morton code is calculated for each primitive by interleaving the bits of its binary coordinates. Sorting them by their Morton code then groups the primitives in a spatially coherent manner. This allows for the parallel construction of a binary tree, level by level, starting from the root, each node representing a contiguous range of primitives. A similar idea is exploited by Zhou et al. [2010] to build octress in the context of surface reconstruction.

Our underlying GPU structure is based on the work of Karras [2012], who maximize the tree construction parallelism reaching real time performances on models beyond 1M polygons. Instead of generating one level of the tree at a time, all the nodes are processed in parallel thanks to a particular tree layout which allows finding the range covered by a node and its children independently from the other nodes.

More precisely, the leaf nodes and the internal nodes are stored in two distinct arrays: L (size n) and I (size $n-1$). By assigning the right indices to the internal nodes, Karras finds the range of leaf nodes they cover and the indices of their children, without processing their ancestors or descendants first. In practice, the index of the

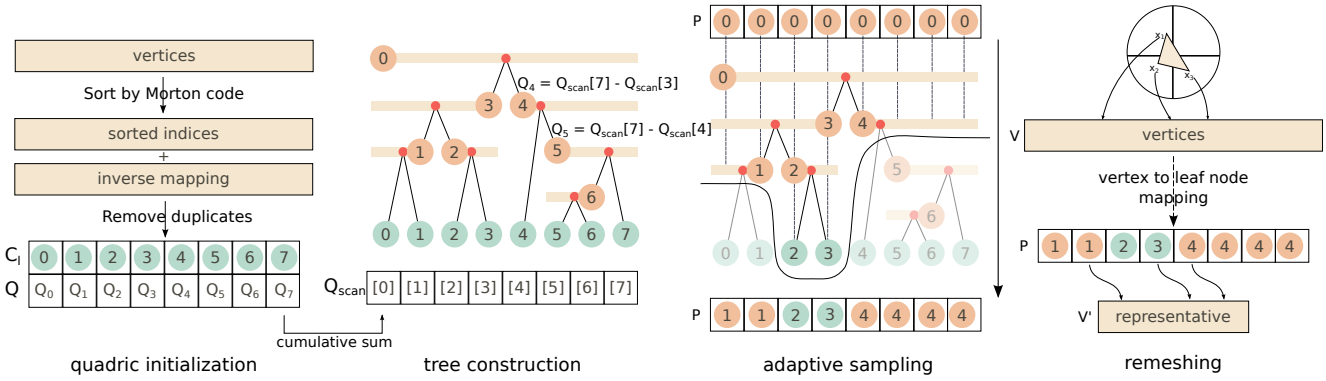


Figure 3: ***Overview***. *We start by sorting the input vertices in the Morton order and generate a table of leaves (left) onto which we accumulate per-leaf error quadrics in the Morton order. We then generate a kd-tree, using the Morton integral to compute the internal nodes in parallel (middle left). Finally, an error-driven simplified geometry is generated by extracting an adaptive cut in the tree in parallel (middle right) and meshing its representative vertices using the input connectivity (right).*

root is set to 0. Then, for every internal node, the indices of its children directly depend on the split position: if the range is split at the position p, the index of the left child will be p (in L if it's a leaf, in I otherwise) and $p+1$ for the right child. Consequently, the index of a node corresponds to one end of the range of leaves it covers. The other end of the range and the split position are found using a binary search, detecting the first differing bit of the Morton code in the range (see Fig. 2). We refer to [Karras 2012] for details on the construction of the structure.

3 Algorithm

Overview Our algorithm (illustrated in Fig. 3) takes as input an indexed triangle mesh M with an error threshold θ and runs entirely on GPU. It provides a simplified indexed mesh M' as an output and consists in several stages:

1. we sort the vertices of M by their Morton code and define a list of leaf nodes with error quadrics,
2. we compute a *Morton integral*, which is a cumulative sum of the quadrics in the Morton order,
3. we compute the internal nodes of a kd-tree K in a parallel, non-hierarchical fashion using the Morton integral,
4. we gather the space partition S as the highest nodes of K with an error lower than θ; the simplified mesh is formed by the representative vertices of S computed using the per-node quadric,
5. we re-index the input triangles shared among three *different* clusters on their representative point [Rossignac and Borrel 1993] to define the simplified connectivity.

In the following, we describe the main steps of our algorithm, essentially structured by the tree construction and traversal, and point to relevant parallel primitives.

Morton sorting During the first step of our algorithm, we sort the vertices by their Morton code. Instead of directly sorting the vertex array (as well as eventual color or normal arrays), we sort by key an array of integer indices, ranging from 0 to the number of vertices of the input mesh, using the Morton code of the vertices as key. We also maintain a look-up table of the inverse mapping from the intial vertex order to the Morton one. Once sorted, we eliminate the duplicates in the Morton order – caused by multiple close-by vertices discretized to the same Morton code. As our array is sorted, duplicates are contiguous and we can simply:

1. mark the first occurence of each Morton code,
2. perform a prefix sum over the marking, to generate the mapping from the sorted vertex array to the compact leaf node array
3. allocate a compact array C_l and store the duplicate-free leaf nodes in it.

As a result, we obtain a mapping from the initial vertex array V to the leaf node array: $V \rightarrow C_l$.

Quadrics initialization For every leaf node l, we compute a 4x4 symmetric quadric matrix Q_l following Garland and Heckbert [1997]. To do so, we compute the face quadric Q_t of each triangle t of M and sum it to the leaf quadrics of the vertices of t: $Q_l = \sum_{t \in l} Q_t$. Assuming the Morton code is computed with a fine enough discretization, performing this sum in parallel using *atomics* induces only a low number of actual collisions between the threads. Additionally, we store the mean vertex of each leaf node, its number of vertices and, optionally, the color and/or the normal vectors. As usual with quadric-based optimization, we consider that Q_l locally models the shape through its minimizer, a representative point obtained by either inverting Q_l, or falling back to the previously stored mean position when the determinant of Q_l is below a numerical stability threshold [DeCoro and Tatarchuk 2007].

Morton integrals Beyond the Morton-based hierarchy, one key aspect of our approach lies in the ability to compute concurrently the error and representative point for all internal node, regardless of the current state of their descendants. We solve this issue by computing a *cumulative sum* of attributes (quadric matrix, mean position, etc) along the Morton ordered leaves, defining in particular an additional node attribute Q_{scan} summing all the quadrics stored in the preceding nodes in the Morton order. The computation of this sum is performed in parallel using an *inclusive scan*. Similar to *integral images* [Viola and Jones 2001], this "Morton integral" allows to compute any sum of attributes for consecutive leaves with only two memory accesses; for instance in the case of the quadric of a node n covering leaves r_1 to r_2:

$$Q_n = Q_{scan}[r_2] - Q_{scan}[r_1 - 1]$$

Parallel tree construction We build our kd-tree K by processing internal nodes independently following Karras [2012], using our Morton-ordered leaf node array C_l, enhanced by the Morton

integral. For each internal node n of K, we need to compute its quadric matrix Q_n and its average vertex (position, color, etc) to figure out its own representative point x_n and approximation error E_n w.r.t. the original geometry of M. Q_n is the sum of the quadrics associated with the children of n, which is equivalent to the sum of the quadrics of all the leaves having n as an ancestor. Thanks to the Morton ordering, they are all contiguous and by construction [Karras 2012] trivial to determine (from index r_1^n to index r_2^n). As a result of our Morton integration, this sum is available in constant time, using 2 quadric accesses and one 4x4 matrix subtraction per node. Therefore, it can be performed during the parallel generation of the tree nodes, independently of the node's children. From Q_n, we extract x_n [Garland and Heckbert 1997; Lindstrom 2000] and the quadric error:

$$E_n = (x_n, 1)Q_n(x_n, 1)^T$$

Algorithm 1 Parallel tree traversal.

```
for each leaf node i in parallel do
    c ← 0
    error ← +∞
    while error > θ and c ∈ internal nodes do
        r ← right[c]
        l ← left[c]
        if i > l then
            c ← r
        else
            c ← l
        end if
        error ← errorNode[c]
    end while
    P[i] ← c
end for
```

Adaptive sampling and remeshing Our error-driven simplification gathers an adaptive space partition of M as a cut in K, formed by the highest nodes of K with an approximation error lower than θ (see Alg. 1). In our parallel context, we formulate this cut extraction as the computation of a mapping, associating each element of C_l to its corresponding node in the target cut. To do so, we initialize an array P, as large as C_l, filled with zeros i.e., the root index. Then, we launch a thread i for each element of C_l that performs a top-down traversal of the tree toward the i-th leaf cell in C_l. When passing a node with index $P[i]$, if $E_{P[i]} < \theta$, we stop the traversal. Otherwise, we set $P[i]$ to one of its child nodes, depending on the value of i: again, by construction, the index of the child nodes correspond to the splitting position in the leaf node array (see Fig. 3). Therefore if i is smaller or equal to the index of the left child, we set $P[i]$ to its value, otherwise we set it to the index of the right child. Consequently, the traversal is always done toward the element of C_l (or leaf node) of index i.

At the end of this procedure, any vertex x of V can be mapped to a node of the target adaptive cut of K in constant time using our $V \rightarrow C_l$ mapping, since it is equivalent to $V \rightarrow P$. We collect in parallel the representative vertices for the nodes of the cut to form the vertex set V' of M' by marking the nodes of the tree that appear in the cut and scanning the marking array. This provides the size to allocate for V' as well as for each node of the cut, the index where its representative vertex should be written in V' (mapping $P \rightarrow V'$).

Last, we generate the connectivity T' of the simplified vertex set by classifying all input triangles of M in parallel according to P. We mark the triangles shared by three different clusters and use a parallel prefix sum to allocate the output triangle array T'. We fill this array with the marked triangles, reindexed over the representative vertices thanks to our $V \rightarrow P \rightarrow V'$ mapping. At this stage, for each triangle, we can optionally check if its normal orientation has flipped and reorder its vertices if necessary.

Variations Our simplification method can account for per-vertex attributes, alternative error metrics or input data in different formats.

For instance, the per-vertex color value can be maintained for each node similarly to quadric matrices. Indeed, it is often desirable to weight color averages by the area of incident triangles. This requires two additional arrays: one used to accumulate the weighted color information (Col) and for accumulating areas (A). After the Morton integral computation, the color for any node is given by:

$$Col_n = \frac{Col_{scan}[r_2] - Col_{scan}[r_1 - 1]}{A_{scan}[r_2] - A_{scan}[r_1 - 1]}$$

The exact same process can be used for other attributes, such as normals. In the last part of the algorithm, this extra per-node value influences the cut extraction by, for instance, bounding its standard deviation in all cut nodes. In this case, one more array (Col^2) is needed, to accumulate the squared weighted color information. After computing its Morton integral, the standard deviation for the color of a node is:

$$S_n = \sqrt{\frac{Col^2_{scan}[r_2] - Col^2_{scan}[r_1 - 1]}{A_{scan}[r_2] - A_{scan}[r_1 - 1]} - [Col_n]^2}$$

with Col_n the mean color for the node.

Unorganized point clouds with normals can also be simplified with our approach by (i) computing the quadric matrices directly from point normals, (ii) propagating the normal for each node as we do for color and (iii) omitting the final meshing step. Note however that, unless per-sampled area/radius is provided, this method requires data with relatively uniform sampling.

4 Implementation and results

We implemented our simplification algorithm in C++/CUDA, using the Thrust library [Nvidia 2011] for the prefix sums and sort, and measured performances on a PC equipped with a GeForce GTX 680 and a 3.6 GHz Intel Xeon E5-1620 CPU. We limited our Morton codes to 30 bits as we store them as 32 bits integers in GPU memory. For comparison, we also implemented a GPU regular grid simplification, similar to the method of Decoro and Tatarchuk [2007]. For quality check, we report high quality offline results obtained with QSlim [Garland and Heckbert 1997].

In Table 1, we report the detailed timings of the different steps of our algorithm for a collection of models. We can see that real time performances are reached for meshes up to several millions of polygons and remain interactive beyond 10 millions. While the tree construction and sampling parts add up to a very small part of the total time, the current bottleneck appears at the initial phase. Note however that for application scenarios implying multiple simplifications of the same model (e.g., many-users remote visualization, view-dependent rendering), this stage is performed once for all. In terms of visual quality, as we can see on the Lucy model for instance (Fig. 4), our adaptive space partition preserves visually important features, with small triangles around features and larger ones in flatter parts.

Model	#T	Out #T	Sort	Dupl	Leaves	Scan	Cons	Sampl	Mesh	Total
Bunny	70K	4,300	0.8	0.2	1.3	1.0	1.8	0.1	1.2	6.4
Dragon	100K	9,300	1.1	0.4	3.9	1.2	2.8	0.1	1.5	11.0
Horse	225K	10,000	1.4	0.7	1.9	1.6	3.0	0.1	1.6	10.3
Buste	510K	19,200	1.7	0.3	5.4	1.6	3.5	0.5	1.9	14.9
Caesar	770K	18,500	2.3	0.7	5.6	2.5	4.4	0.9	2.1	18.6
Grog	1M	41,000	2.8	1.1	5.0	3.5	5.1	0.9	2.3	20.8
Gargoyle	1.7M	44,500	3.4	1.0	7.1	3.4	4.1	0.6	2.5	22.1
Raptor	2M	22,500	3.6	1.2	8.8	2.8	1.7	0.2	2.2	20.6
Neptune	4M	24,500	4.6	1.3	30.1	1.8	2.4	0.3	6.6	47.0
Crab	11M	64,200	12.8	3.3	69.8	2.4	6.0	0.9	11.4	106.5
Lucy	28M	116,500	29.7	5.6	151.2	6.1	5.5	0.8	23.9	222.7

Table 1: ***Performance measures*** *in ms, without CPU-GPU memory transfer.* ***Sort****: Morton sorting,* ***Dupl****: duplicates removal.* ***Leaves****: initialization of the leaves attributes (quadrics and mean).* ***Scan****: Morton integration.* ***Cons****: parallel tree construction.* ***Sampl****: error-driven tree traversal and cut extraction.* ***Mesh****: triangles re-indexing.*

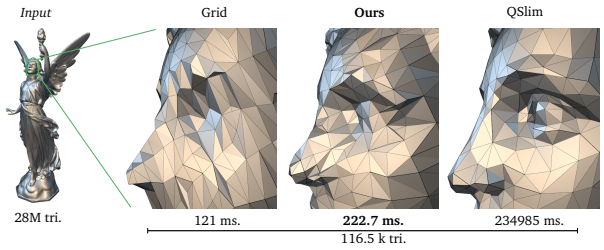


Figure 4: ***Adaptive simplification*** *of the Lucy model. Our approach approximates better features and singular structure than uniform clustering, but still runs in split-second for this large model.*

Model	model size	additional space used 27 bits MC	30 bits MC
Bunny	1	7	7
Dragon	2	8	9
Horse	5	19	20
Buste	11	32	37
Caesar	17	61	63
Grog	22	75	80
Gargoyle	39	98	120
Raptor	45	44	89
Neptune	91	83	138
Crab	259	217	275
Lucy	642	502	557

Table 2: ***Memory usage*** *in Mb for 27 bits and 30 bits Morton codes.*

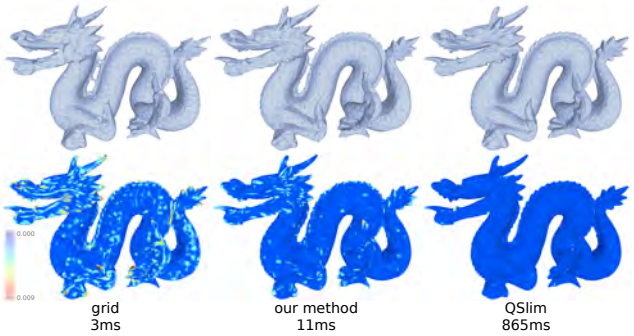


Figure 5: ***Error visualization*** *for the Stanford Dragon model (100K triangles), simplified to 9300 triangles.*

Model	Method	#T	Out #T	Time	H	M12	M21
Bunny	Ours	70K	4,300	6.4	0.013802	0.000500	0.000499
	QSlim			503	0.002425	0.000295	0.000288
	Grid			2.5	0.012880	0.001327	0.001294
Dragon	Ours	100K	9,300	11.0	0.008810	0.000671	0.000585
	QSlim			865	0.009327	0.000370	0.000251
	Grid			3	0.013838	0.001142	0.001039
Horse	Ours	225K	10,000	10.3	0.004485	0.000280	0.000280
	QSlim			1,728	0.001862	0.000104	0.000101
	Grid			4	0.009340	0.000588	0.000555
Buste	Ours	510K	19,200	14.9	0.003257	0.000237	0.000236
	QSlim			4,419	0.001365	0.000095	0.000094
	Grid			6	0.007113	0.000505	0.000490
Caesar	Ours	770K	18,500	18.6	0.013818	0.000220	0.000209
	QSlim			7,544	0.013894	0.000130	0.000124
	Grid			8	0.014448	0.000431	0.000413
Grog	Ours	1M	41,000	20.8	0.005253	0.000255	0.000249
	QSlim			9,101	0.003686	0.000128	0.000124
	Grid			9	0.007022	0.000531	0.000482
Gargoyle	Ours	1,7M	44,500	22.1	0.006374	0.000236	0.000237
	QSlim			15,668	0.004018	0.000120	0.000118
	Grid			13	0.006929	0.000496	0.000469
Raptor	Ours	2M	22,500	20.6	0.012457	0.000280	0.000277
	QSlim			19,057	0.011525	0.000143	0.000137
	Grid			15	0,012629	0.000423	0.000423
Neptune	Ours	4M	24,500	47.0	0.005375	0.000272	0.000282
	QSlim			43,941	0.001851	0.000089	0.000082
	Grid			27	0.008222	0.000487	0.000450
Crab	Ours	11M	64,200	106.5	0.005439	0.000233	0.000244
	QSlim			92,933	0.002617	0.000057	0.000055
	Grid			53	0.004677	0.000319	0.000315
Lucy	Ours	28M	116,500	222.7	0.008423	0.000185	0.000179
	QSlim			234,985	0.000894	0.000035	0.000033
	Grid			121	0.003576	0.000211	0.000203

Table 3: ***Quality and time comparison*** *with* ***H*** *the Hausdorff distance between the original model and the simplification,* ***M12*** *the mean distance from the original model to its simplification and* ***M21*** *the mean distance from the simplification to the original model. Timings are given in ms.*

In Table 2, we present the memory usage for the same set of models. The storage space needed on the GPU only depends on the number of vertices in the input geometry and on the desired Morton code precision. In the worst case scenario, if the Morton code is chosen precise enough to be different for every single input vertex (for example with the bunny model), there will be as many leaf nodes, and thus as many quadrics, average vertices , colors... However, as the size of the model increases, there will be more and more duplicated morton codes, and consequently not as many leaf nodes. For this reason, the GPU memory used by the algorithm does not increase as quickly as the number of triangles in the input model.

We compare our method with GPU grid clustering, which is very fast but not adaptive: for a comparable number of triangles, our method preserves visually important features that disappear with regular clustering while, although slower, keeping timings in a similar range. We also compare our results with QSlim, which favors quality over performances. We report timings and objective error measures in Table 3, with in particular *mean* and *Hausdorff* distances between the original and simplified meshes. Measures are performed using the Metro tool [Cignoni et al. 1998]. We plot visually the simplification error for the three approaches in Fig. 5. While for grid clustering, the error is concentrated around details, our method gives a globally lower error, with lower damages on features. This reflects in the error measures, with an improved Hausdorff distance and a significantly better mean distance to the original model. Of course, the quality of the approximation provided by our algorithm cannot compete with the QEM-based progressive reduction of QSlim. However, as illustrated in the Fig. 11, the visual quality remains overall good, for an execution time which is three orders of magnitude faster.

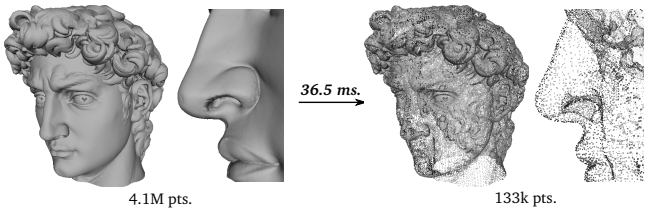


Figure 6: ***Point-based simplification*** *using our approach.*

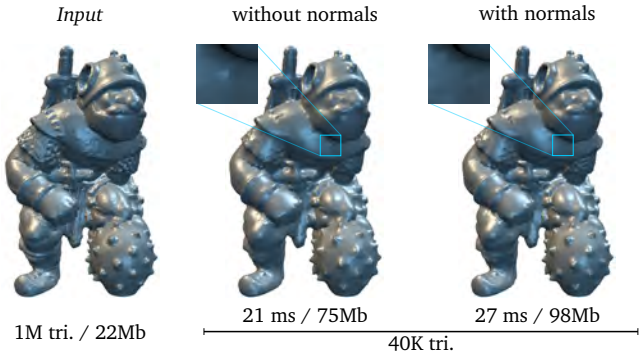


Figure 7: ***Influence of the normals*** *on the visual aspect, timings and GPU memory usage. On the Grog model, small clustering artifacts disappear when the normals are maintained, for example under the beard, on the leg or on the shoulder.*

In Fig. 6, we show an example of point cloud simplification using our approach on the Michaelangelo David's head model. Again, while the geometric resolution is drastically reduced, the important features, in particular the sharp edges, are captured in the simplified point sampling. The generation time is comparable to the mesh case, as the triangle reindexing step is not a bottleneck of our approach. In Fig. 7 and Fig. 8, we show examples of simplifications preserving the normal and color information. As we can observe on the Grog model, maintaining normals improves the visual aspect of the approximation by helping preserving details and visually reducing small clustering artefacts. We also show an example of simplification accounting for the color information provided on a per-vertex basis in the input. Visually important features that mostly exist through the color distribution are better preserved in this case. This is particularly useful for stereovision data, which often exhibits more features in the color than in the geometry. Since no significant change to the algorithm is needed to preserve an additional attribute (we just maintain it along with the quadrics, means, etc), the cost in time and GPU memory is relatively small.

In Fig. 9, we provide an example of simplification for a scene exhibiting a strongly varying vertex density. We can observe a proper behavior of our approach, with large polygons remaining intact with small ones being correctly simplified. Last, in Fig. 10, we show experiments performed on animated data. In particular, we focus on performance capture data [de Aguiar et al. ; Beeler et al. 2011], both for full body (medium resolution) and face (high resolution) models. We illustrate the range of possible applications for our method with on one side a rather simple body model (40k triangles) simplified by a factor of 5, and on the other side a high resolution face model (2.3M triangles) simplified by a factor 100. We show in this case the resulting mesh structure, obtained in real time as well.

Limitations Although adaptive, our approach falls in the *clustering* category, which induces at least two main limitations. First, such methods lack guarantees on the topology preservation At coarse scale, under extreme simplification rates, the input topology frequently changes, collapsing nearby layers. Although this is not always a weakness [Cohen-Steiner et al. 2004], more control on this behavior would be desirable. Second, we do not provide an absolute (polygon-wise) control over the output model size: although the user can set the desired level of simplification by ruling θ, guaranteeing an exact number of polygons is tedious. Our Morton integration is also bounded by the machine precision: when computing cumulative sums on very large arrays, imprecisions accumulate and can cause inaccurate quadric matrices, which can lead to instability in the output mesh. We reduce the impact of this stability problem by scaling the input mesh to the unit cube (scaling back the output), and by using double precision when computing cumulative sums. However, the error may still be significant when processing very large (giga polygons) models. Such data requires a different class of simplification algorithms (out-of-core/streaming), at least as a first pass. Once the model has reached a first appropriate (dense yet in-core) simplification, it is possible to chain one or several other in-core algorithms. Addressing these issues while preserving high performances is clearly one of the main direction for future work.

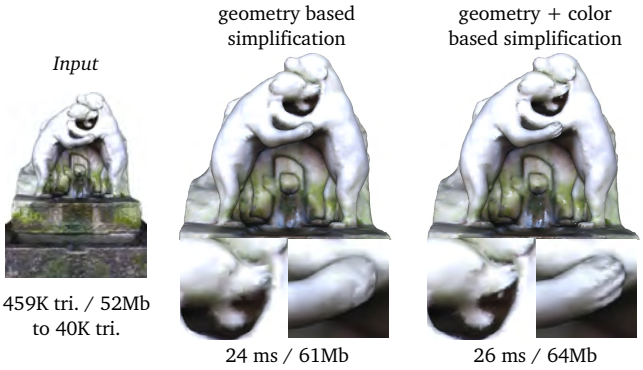


Figure 8: ***Influence of the color****, used to drive the simplification (on the right) or not (on the left), on the visual aspect, timings and amount of GPU memory used by the algorithm.*

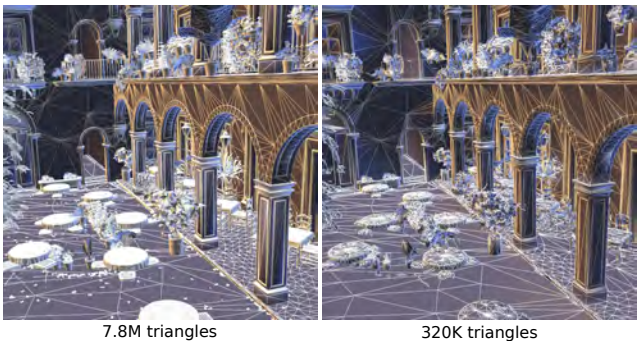


Figure 9: ***Simplification of a 7.8M triangles model with strongly varying vertex density*** *in 54ms using our approach.*

5 Conclusions

We have introduced a high performance adaptive geometry simplification algorithm which can process objects made of millions of polygons in real time and on commodity hardware. We achieve such a performance level by introducing *Morton integrals*, which are cumulative sums of samples attributes or error measures performed along their Morton enumeration. This intermediate object enables the parallel construction of a hierarchical approximation structure and its error-driven parallel traversal to extract a cut of nodes tailoring the simplified geometry. As a result, we obtain adaptive simplified meshes on-the-fly, with better quality than state-

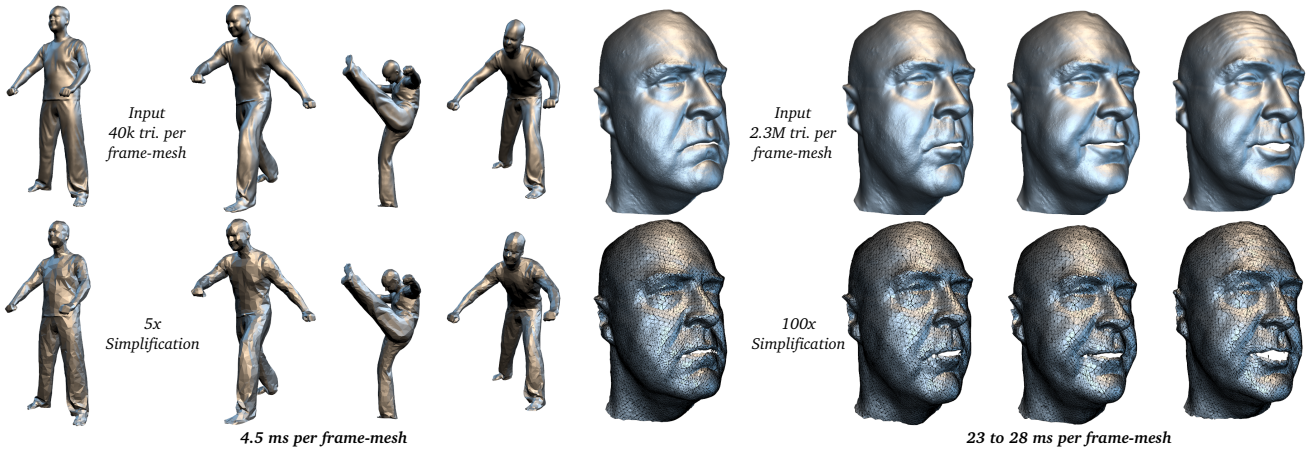


Figure 10: ***Adaptive simplification*** *of performance capture data.* ***Left:*** *500 frames are processed independently at about 200 Hz.* ***Right:*** *the three orders of magnitude downsampling is performed in real time on this dense face model.*

of-the-art high performance methods. We also showed that our method can be extended to account for surface attributes and meshless input. Our approach completes the high performance GPU geometry processing pipeline, which now features *live and adaptive* filtering, refinement and simplification. Beyond alternative metrics and additional surface attributes, we believe that the concept of Morton integration can be useful to other kinds of applications, providing a parallel scalable support for various flavors of multiresolution geometry processing and analysis methods.

Acknowledgments. Animated meshes are courtesy DRZ and MPI. This work has been partially supported by the European Commission under contracts FP7-323567 HARVEST4D and FP7-287723 REVERIE, and by the ANR iSpace&Time project.

References

ADAMS, A., GELFAND, N., DOLSON, J., AND LEVOY, M. 2009. Gaussian kd-trees for fast high-dimensional filtering. *Trans. Graph. 28*, 3, 21:1–21:12.

BEELER, T., HAHN, F., BRADLEY, D., BICKEL, B., BEARDSLEY, P., GOTSMAN, C., SUMNER, R. W., AND GROSS, M. 2011. High-quality passive facial performance capture using anchor frames. *Trans. Graph. 30*, 75:1–75:10.

BOUBEKEUR, T., AND ALEXA, M. 2009. Mesh simplification by stochastic sampling and topological clustering. *Computer & Graphics (Proc. Shape Modeling International) 33*, 3, 241–249.

CIGNONI, P., ROCCHINI, C., AND SCOPIGNO, R. 1998. Metro: measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2, 167–174.

COHEN-STEINER, D., ALLIEZ, P., AND DESBRUN, M. 2004. Variational shape approximation. *Trans. Graph. 23*, 3, 905–914.

CROW, F. C. 1984. Summed-area tables for texture mapping. In *SIGGRAPH*, 207–212.

DE AGUIAR, E., STOLL, C., THEOBALT, C., AHMED, N., AND SEIDEL, H. Performance capture from sparse multi-view video. *Trans. Graph. 27*, 3, Art. 98.

DECORO, C., AND TATARCHUK, N. 2007. Real-time mesh simplification using the GPU. In *ACM I3D*, 161–166.

GARANZHA, K., PANTALEONI, J., AND MCALLISTER, D. 2011. Simpler and faster HLBVH with work queues. In *HPG*, 59–64.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. *SIGGRAPH*, 209–216.

GRUND, N., DERZAPF, E., AND GUTHE, M. 2011. Instant level-of-detail. In *Vision, Modeling and Visualization*, 293–299.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1993. Mesh optimization. In *SIGGRAPH*, 19–26.

KARRAS, T. 2012. Maximizing parallelism in the construction of bvhs, octrees, and k-d trees. In *High Performance Graphics*, 33–37.

LAUTERBACH, C., GARLAND, M., SENGUPTA, S., LUEBKE, D., AND MANOCHA, D. 2009. Fast BVH Construction on GPUs. *Computer Graphics Forum 28*, 2 (Apr.), 375–384.

LINDSTROM, P. 2000. Out-of-core simplification of large polygonal models. *SIGGRAPH*, 259–262.

LINDSTROM, P. 2003. Out-of-core construction and visualization of multiresolution surfaces. In *I3D*, 93–102.

NVIDIA, 2011. Thrust. https://developer.nvidia.com/Thrust.

PANTALEONI, J., AND LUEBKE, D. 2010. HLBVH: hierarchical LBVH construction for real-time ray tracing of dynamic geometry. In *High Performance Graphics*, 87–95.

PIXAR, 2013. Opensubdiv.

ROSSIGNAC, J., AND BORREL, P. 1993. Multi-resolution 3d approximations for rendering complex scenes. In *Modeling in Computer Graphics*, 455–465.

SCHAEFER, S., AND WARREN, J. 2003. Adaptive vertex clustering using octrees. In *Geom. Design & Computing*, 491–500.

SHAFFER, E., AND GARLAND, M. 2001. Efficient adaptive simplification of massive meshes. In *Visualization*, 127–551.

SHAFFER, E., AND GARLAND, M. 2005. A multiresolution representation for massive meshes. *TVCG 11*, 2, 139–148.

VIOLA, P. A., AND JONES, M. J. 2001. Robust real-time face detection. In *ICCV*, 747.

ZHOU, K., GONG, M., HUANG, X., AND GUO, B. 2010. Data-Parallel Octrees for Surface Reconstruction. *TVCG 17*, 5, 669–681.

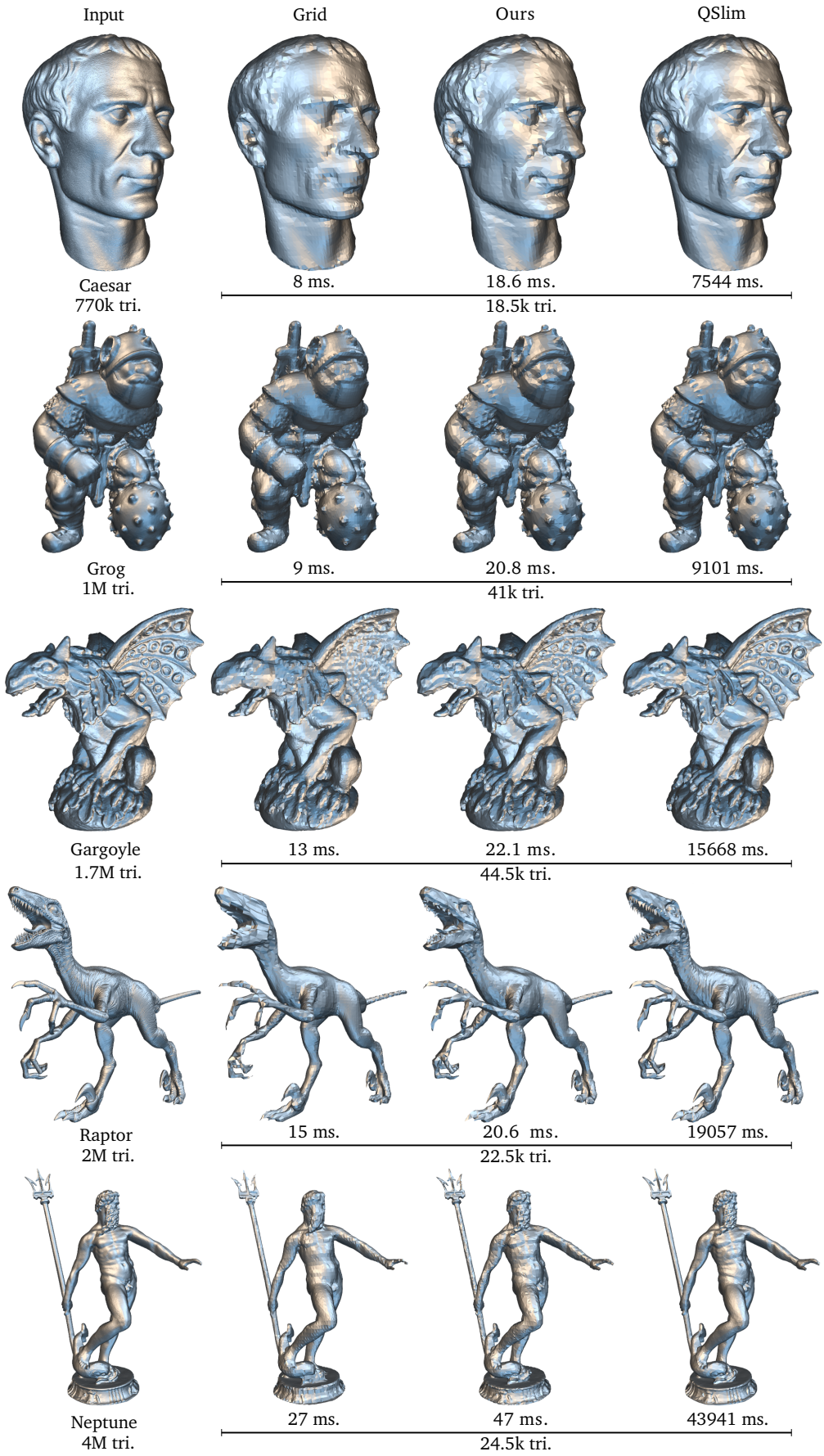


Figure 11: *Examples.*

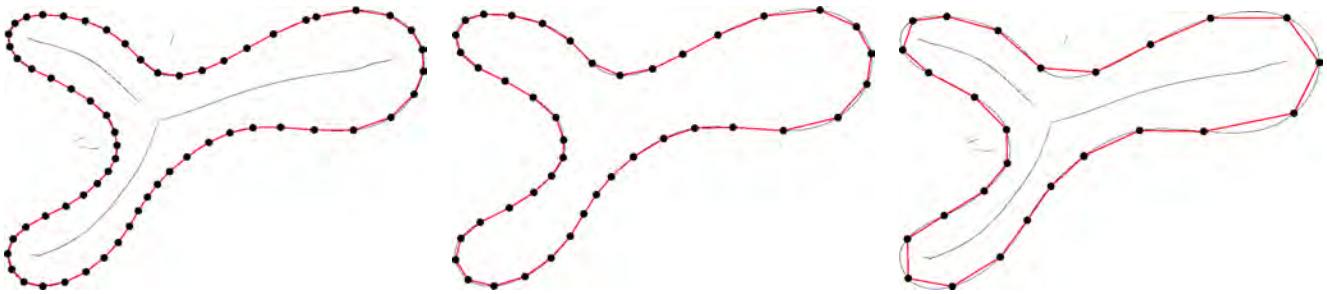
Eurographics Symposium on Geometry Processing 2016
Maks Ovsjanikov and Daniele Panozzo
(Guest Editors)

Volume 35 (2016), Number 5

Curve Reconstruction with Many Fewer Samples

S. Ohrhallinger[1], S.A. Mitchell[2] and M. Wimmer[1]

[1]Institut für Computergraphik und Algorithmen, TU Wien, Austria
[2]Center for Computing Research, Sandia National Laboratories, U.S.A.



(a) State of the art [DK99], $\varepsilon < \frac{1}{3}$: 61 points. (b) We prove $\varepsilon < 0.47$ to reduce to 43 points. (c) Our *reach*-based $\rho < 0.9$ needs just 26 points.

Figure 1: *A smooth curve (black) with the relevant subset of its medial axis and its reconstruction (red) with the proposed* HNN-CRUST *algorithm. We first tighten the state-of-the-art sampling condition (a) from* $\varepsilon < \frac{1}{3}$ *to* $\varepsilon < 0.47$ *(b). Then we show that our new sampling condition based on the* reach *reduces samples even further (c). For the shown example, the state-of-the-art sampling condition requires 135% more samples than ours, which are irrelevant for homeomorphic reconstruction.*

Abstract

We consider the problem of sampling points from a collection of smooth curves in the plane, such that the CRUST *family of proximity-based reconstruction algorithms can rebuild the curves. Reconstruction requires a dense sampling of local features, i.e., parts of the curve that are close in Euclidean distance but far apart geodesically. We show that* $\varepsilon < 0.47$*-sampling is sufficient for our proposed* HNN-CRUST *variant, improving upon the state-of-the-art requirement of* $\varepsilon < \frac{1}{3}$*-sampling. Thus we may reconstruct curves with many fewer samples. We also present a new sampling scheme that reduces the required density even further than* $\varepsilon < 0.47$*-sampling. We achieve this by better controlling the spacing between geodesically consecutive points. Our novel sampling condition is based on the reach, the minimum local feature size along intervals between samples. This is mathematically closer to the reconstruction density requirements, particularly near sharp-angled features. We prove lower and upper bounds on reach* ρ*-sampling density in terms of lfs* ε*-sampling and demonstrate that we typically reduce the required number of samples for reconstruction by more than half.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

The *connect-the-dots* game without numbers on the dots corresponds to the problem of reconstructing the connectivity of a planar curve from a set of unstructured points sampled on that curve.

More formally, our problem is to sample points from a curve, throw away the curve, then connect points to those nearby. For the reconstruction to be correct, the points should be connected in the same order as on the curve. A sparser sampling is valuable whenever placing points, storing them, or reconnecting them is expensive. But it must not be too sparse because the connectivity must be restorable from just the points.

The samples capture *the essential shape information*, topological and geometric. The *Human Visual System* is able to complete the connectivity based on the *Gestalt principles of Proximity and Continuity*. Familiar examples are planting flower bulbs to form a shape, or animating patterns in the night sky by lit drones. Recon-

struction algorithms are also built on these proximity and continuity principles. Potential applications include generating efficient shape descriptors based on points (as opposed to curve-based descriptions), and compressing or progressively streaming point sets. These can be used to decide whether to request additional samples from a sensor, or that the sample set is of sufficient quality.

If the curve is sampled densely, connecting nearby points will reconstruct the correct curve. The less dense the sampling, the more challenging it is to reconstruct the curve, especially at *features* where two intervals of the curve come close to each other, or where the curvature is high. Reconstruction algorithms require some *sampling conditions* on the input in order to guarantee a correct output. The particular algorithm determines the required density.

Sampling algorithms also guarantee some *sampling conditions* on the output. However, these are rarely of exactly the same form, and it is non-trivial to describe the reconstruction algorithm's requirements in terms of the sampling algorithm's guarantees. This leads to a mismatch between the minimum local density required for reconstruction, and the maximum local density a sampling algorithm produces. Typically we choose some local measure of a curve, and sample density is guaranteed to be some parameterized fraction of that measure. The closer the guarantees match the requirements, and the tighter we can describe the necessary and sufficient parameter values, the more efficient we can make our sampling. This leads to our goal: to sample curve features as sparsely as possible, yet still guarantee that the reconstructed curve is correct.

We describe the reconstruction algorithm HNN-CRUST, a variant of NN-CRUST [DK99]. Many sampling algorithms use the ε-sampling condition, which is based on comparing ε times the local feature size (lfs) at a point to the distance to its nearest sample [ABE98]. The known parameter bounds for this combination, $\varepsilon < 1/3$-sampling, appear weak, and we show a better one, $\varepsilon < 0.47$-sampling. Furthermore, we provide a *better sampling condition* based on a different measure of the curve, the *reach*, ρ. The reach is bounded by the minimum local feature size at all points between two samples. The reach is more suitable for HNN-CRUST, and we believe for proximity-based reconstruction in general.

Our first contribution is the tightening of $\varepsilon < 1/3$-sampling to $\varepsilon < 0.47$-sampling.

Our second and main contribution is the new reach-based ρ-sampling condition, with the following properties:

- ρ-sampling is simple, with a single parameter like ε-sampling.
- $\rho < 0.9$-sampling guarantees that HNN-CRUST *correctly* reconstructs the curve.
- The polygonal reconstruction *geometrically approximates* the original curve, similar to $\varepsilon < 0.47$-sampling.
- $\rho < 0.9$-sampling has *only half the samples* when lfs is constant, and never more than $\varepsilon < 0.47$-sampling.
- The same condition holds when *limiting the Hausdorff distance* from the polygonal reconstruction to the original curve.
- Thus, $\rho < 0.9$-sampling permits much *sharper angles*: up to 73°, compared to 120° for $\varepsilon < \frac{1}{3}$-sampling.

Programs for sampling smooth curves under both sampling conditions are provided online as open source. One can explore varying ε and ρ parameters, as well as Hausdorff distance limits.

2. Related Work

We briefly review curve reconstruction algorithms and their associated sampling conditions. Early methods guaranteed curve reconstruction from uniformly dense samples, where the maximum distance between consecutive samples is a global constant [EKS83, KR85, FMG94, Att97]. However, since the sampling density is constant, it depends on the maximum curvature, which is inefficient for flat parts of the curve. Those methods work well for curves whose curvature is limited above by a global constant, such as for r-regular sets [DT14, DT15], for which guarantees are given for non-noisy [Ste08] and noisy point sets [ST09].

Sampling framework: To get rid of this over-sampling, the seminal paper by [ABE98] proposed the CRUST algorithm. It filters edges from the Delaunay triangulation. Sampling density varies according to both curvature and Euclidean distance between geodesically-far curve intervals. They also introduced a non-uniform sampling condition based on local feature size, called ε-sampling, and proved that CRUST reconstructs a manifold boundary; [Dey06] proved $\varepsilon < 0.2$ is sufficient. Many subsequent methods use this sampling framework. [Gol99] optimized and simplified CRUST to a single-step algorithm. This family of algorithms constrain their output to edges of the Delaunay triangulation.

Proximity-based algorithms: [DK99] introduced the simple proximity-based algorithm NN-CRUST for general dimensions. It guarantees reconstruction of closed curves for $\varepsilon < 1/3$. [Alt01] improved the condition to $\varepsilon < 0.5$, but required $\alpha > 151^\circ$. [Len06] claims a better bound for NN-CRUST: $\varepsilon < 0.4$, or $\varepsilon < 0.48$ with additional angle restrictions, but does not show proof. He also noted shortcomings of ε-sampling, e.g. for sharp corners, as open problems. These investigations show that there is still room for improvement. Without angle restrictions, the best proven bound is $\varepsilon < 1/3$-sampling, and this is not tight.

Extensions: NN-CRUST was extended to CONSERVATIVE-CRUST [DMR99] to handle open curves, and later to *GathanG* [DW02], which modified the sampling condition to handle sharp corners, but requires $\alpha > 150^\circ$ otherwise. [FR01] introduced the notion of curve reconstruction as requiring a homeomorphism between the polygonal reconstruction and the curve, but not geometric closeness. They also presented their own sampling condition, requiring several parameters, in order to reconstruct collections of open and closed curves with sharp corners. Other approaches proposed a sampling condition using a vision function based on human perception and some empirically established parameters [ZNYL08, NZ08]. [OM13] presented a three-step method which is able to reconstruct very sparsely-sampled features, for closed curves, by considering it as a global problem. The first step guarantees reconstruction for $\varepsilon < 0.5$, but in order to handle the sharp angles of 0°–60° it requires an additional constraint, slowly varying density as a maximum ratio between adjacent edge lengths.

Sampling: [LKvK*14] generate connect-the-dot puzzles from curves which vary in the criterion of connectivity, using different sampling criteria. Their variant *connect-the-closest-dot* corresponds closely to our problem, but our sampling condition neither requires encoding of topology indicators nor a minimum distance between points.

3. Overview

We describe a variant of NN-CRUST [DK99] that we call HNN-CRUST, which permits reconstruction of angles sharper than $< 90°$, as small as $60°$. While it improves the reconstruction, it is mostly a vehicle to compare our ρ-sampling condition to the widely used ε-sampling condition [ABE98]. We consider only these two sampling conditions for comparison because the others are highly tailored to specific reconstruction algorithms and require careful adjustment of many parameters.

The HNN-CRUST reconstruction algorithm implies that two edges meet at an angle of at least $60°$. The reconstruction is correct for $\varepsilon < 0.47$. The angle between consecutive edges is related to curvature and sampling density: the flatter the curve and the denser the sampling, the larger the angle. (In the limit, for an infinite sampling of a regular curve, we get $180°$.)

The essence of our paper is a new sampling condition that samples more sparsely where possible, closer to the minimum tolerated by the reconstruction. The weakness of ε-lfs sampling is that, in essence, the sampling condition's output guarantee is that the maximum distance between consecutive samples is limited by the lfs at a point half way between them. The sampling condition is less sensitive to the lfs at other points, and the lfs at the samples themselves are completely irrelevant. *In contrast, the reconstruction algorithm's input requirements are sensitive to small lfs at the samples themselves.* This mismatch leads one to select an ε small enough that the algorithm is correct even when the lfs changes rapidly between the midpoint and the sample. The sampling density is driven by this worst case, and is much denser than necessary when the lfs is not changing rapidly. The strength of our new measure, the reach, is that it is sensitive to small lfs at the samples, and so the sampling condition is more closely matched to the reconstruction requirements.

The rest of the paper is organized as follows. In Section 4 we introduce the required background and definitions. We explain the reconstruction algorithm in Section 5 together with some properties. In Section 6 we give our improved $\rho < 0.9$-sampling condition based on the *reach* rather than *local feature size*. In Section 7 we prove that $\rho < 0.9$-sampling suffices. We also prove bounds relating ρ-sampling to ε-sampling, which indirectly proves $\varepsilon < 0.47$-sampling suffices. We compare the results of our reconstruction algorithm and sample density for our sampling condition in Section 8. In Section 9 we give our conclusions along with potential extensions.

4. Definitions

We give the following definitions, most of which have been introduced by [ABE98]:

The domain is a collection of *smooth curves* C, by which we mean bounded 1-manifolds embedded in $\mathbb{R}^2$, which are twice-differentiable everywhere except perhaps at boundaries. This permits C to consist of multiple connected components, such as a circle and a closed segment, but without crossings, T-intersections or sharp angles. The boundary of a closed segment consists of two *terminus* points. Note that each connected component of C induces a natural geodesic ordering of its points, which can be traversed in one of the two possible directions. Based on such a directed ordering, we say that a curve point lies *before* or *after* another, or *between* two curve points. The *interval* $I(p) \equiv [s_0, s_1]$ is the set of points $p \in C$ *between* s_0 and s_1. A *chord* is the straight edge between two points of an interval.

The set of samples is S. Samples s_0 and s_1 are *adjacent* or *consecutive* if there is no other sample on their interval. Let $\|\overrightarrow{n}\|$ denote the Euclidean L_2-norm. We measure distances in the Euclidean metric, except where we specifically denote geodesic distance.

The *nearest neighbor* s_0 to sample point s_1 is $\operatorname{argmin}_{s_j \in S \setminus s_1} \|s_1, s_j\|$. The *half neighbor* s_2 is the closest sample in the half-space H which is partitioned by the perpendicular bisector of the edge $\overline{s_0 s_1}$ and does not contain s_0: $\operatorname{argmin}_{s_j \in S \setminus s_1, s_j \in H} \|s_1, s_j\|$. We often order all neighbors by Euclidean distance: let n_i be the i-th nearest sample to s_1.

We define the *manifold boundary* B as the correct piece-wise linear reconstruction of C, which connects the samples of each connected component in the same order as on C and adds no other edges.

The *medial axis* M of C is the closure of all points in $\mathbb{R}^2$ with two or more closest points in C [Blu67].

We define the *local feature size* $\operatorname{lfs}(p)$ for a point $p \in C$ as the Euclidean distance from p to its closest point m of M. This definition is loosely based on [Rup93], but simplified because we are only considering smooth curves. Note $\operatorname{lfs}(p)$ is slowly varying, 1-Lipschitz continuous with $|\operatorname{lfs}(p_0) - \operatorname{lfs}(p_1)| \leq \|p_0, p_1\|$.

Definition 1 is the widely used lfs sampling condition [ABE98]:

Definition 1 A smooth curve C is ε-*sampled* by point set S if every point $p \in C$ is closer to a sample than an ε-fraction of its local feature size: $\forall p \in C, \exists s \in S : \|p, s\| < \varepsilon \operatorname{lfs}(p)$.

In contrast, the *reach* [Fed59] for a set $\mathcal{S}$ is the largest "radius" r such that points closer than r to $\mathcal{S}$ have a unique closest point of $\mathcal{S}$. The reach is similar to the smallest distance to the medial axis. This inspires our definition of the *reach* of a curve interval I as $\inf \operatorname{lfs}(p) : p \in I$, where the lfs is defined by all of C.

5. Our Improved Reconstruction Algorithm HNN-CRUST

HNN-CRUST simply connects each sample $s \in S$ to its nearest and half neighbor. (If s is a terminus of a curve, then only the nearest neighbor gets an edge. If the terminus is not specifically marked, then the reconstruction will have an extra edge.) Let h be the perpendicular bisector of the nearest neighbor edge, and H its half-space containing s. Then the half neighbor lives in H but outside the nearest-neighbor radius around s; see Figure 2. In Figure 3 we show how CRUST and HNN-CRUST compare when consecutive samples make sharp angles.

5.1. HNN-CRUST Mimics the Human Vision System

Three *Gestalt principles* are implicitly present in HNN-CRUST. (Since our algorithm does not attempt to reproduce the Human Vision System, some reconstructions will not match typical human

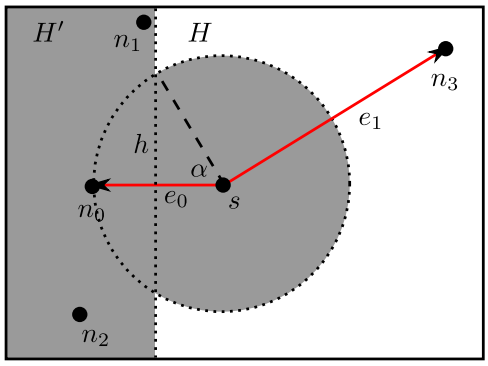


Figure 2: HNN-CRUST *reconstruction of an edge-pair for a sample s. Edge e_0 connects s to its nearest neighbor n_0. The other edge e_1 is the shortest edge connecting s with a vertex in halfspace H. Further, observe that this vertex (here n_3) must lie inside the white shaded area of H, since no sample is closer to s than n_0. This implies the two edges meet at an angle of at least* 60°.

perception.) These principles can be observed in Figure 3, and are as follows:

- *Proximity* is enforced by always connecting the nearest neighbor, and for the second neighbor choosing the nearest neighbor inside the restricted halfspace.
- *Good Continuity* arises from requiring angles between incident edges to be more than 60°.
- *Closure* means we close the curve, unless excessive distance between points implies a hole or an open curve.

6. An Improved Sampling Condition

We will show that HNN-CRUST reconstructs a smooth curve for an ε-sampling with $\varepsilon < 0.47$. For higher values of ε, [ABE98] observed some interesting properties. Theorem 12 noted that for $\varepsilon < 1$, the reconstruction $B \subset DT$ (Delaunay Triangulation). Theorem 13 showed that the distance from any point $p \in C$ to the polygonal reconstruction B is bounded above by $\varepsilon^2 \text{lfs}(p)/2$. However, we have not seen any attempts to guarantee reconstruction for $0.47 \leq \varepsilon < 1$, so we will investigate why this is hard.

6.1. Large ε Do Not Keep Geodesically Distant Intervals Away

Lfs ε-sampling (Definition 1) just requires a sample to be within an ε-fraction of the lfs at that point. Thus, as $p \in C$ approaches a sample point, $\text{lfs}(p)$ may be arbitrarily small, and the sampling condition is still satisfied. The only thing keeping geodesically distant curves sections separate is the ε-lfs condition at points farther away, such as the point $x \in C$ midway between samples often used in proofs. Therefore, for an ε-sampling with $0.47 \leq \varepsilon < 1$, HNN-CRUST may connect non-adjacent samples and fail.

6.2. The Solution for Keeping Them at the Proper Distance

To sample more sparsely where samples are not needed, but still ensure samples are dense enough where the curve approaches itself, we must have a sampling condition that depends more strongly on the lfs near samples. Our sampling condition replaces $\text{lfs}(p)$ by the *reach,* the minimum lfs on an interval.

Definition 2 The *reach* [Fed59] of interval I is $\inf_{p \in I} \text{lfs}(p)$.

Definition 3 A smooth curve C is ρ-sampled by point set S if every point $p \in C$ is closer to a sample than a ρ-fraction of the reach of the interval $I(s_0, s_1)$ of consecutive samples containing it. That is, $\forall p \in I = [s_0, s_1]$ with $s_0, s_1 \in S : \|p, s_0\| < \rho\,\text{reach}(I)$ or $\|p, s_1\| < \rho\,\text{reach}(I)$.

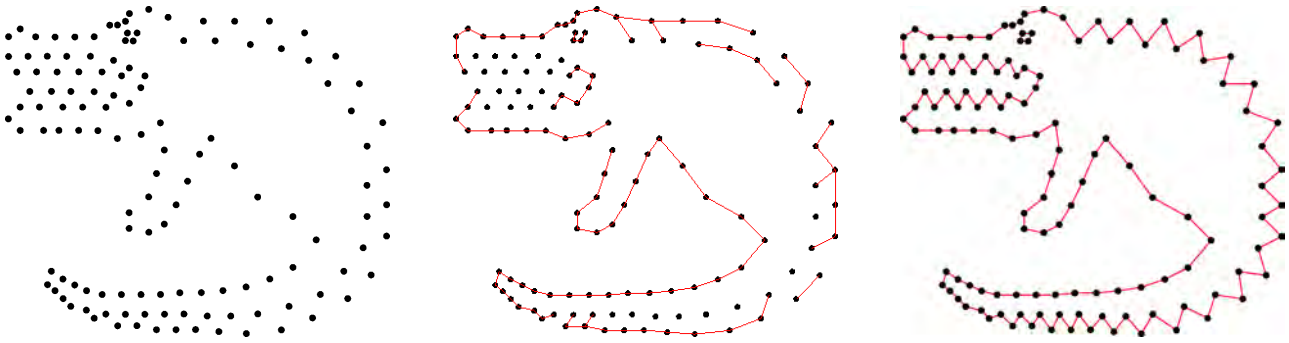
7. Correctness of HNN-CRUST for $\rho < 0.9$ and $\varepsilon < 0.47$.

The goal of this section is to show reconstruction provides correct output for certain ρ. Indeed, we will show that every ε-sample is also a ρ-sample, so this implies correctness for certain ε. The idea is to show that consecutive samples are close together, that geodesically close samples are farther, and geodesically distant samples are farther as well. We establish a series of geometric preliminaries relating distances between samples, the curve, and its medial axis. Most are similar to previous observations, but in some cases we provide stronger results or more elegant proofs.

The first lemma is useful for geodesically close samples. Theorem 2 in [OM13] shows, amongst other things, that Euclidean chord length increases monotonically with geodesic distance, as long as chords do not intersect M. In particular, for $I = [p_0, p_2]$, as x advances on C from p_0 to p_2, chord length $\|\overline{p_0 x}\|$ is strictly increasing, and has no local maxima. Here we show something stronger, with a more elegant proof.

Lemma 1 Let $p_0, p_2 \in C$. If the chord $h \equiv \overline{p_0 p_2}$ does not cross the medial axis M of C, the interval $I = [p_0, p_2]$ lies inside the smallest circle O_{02} containing $\overline{p_0 p_2}$. Moreover, for $t \in I$, distances $\|p_0 t\|$ and $\|p_2 t\|$ are strictly monotonic in t's ordering on I.

Proof See Figure 5 left. For each point x on segment h, consider the largest radius disk O centered at x with no points of C in its interior. Let t be a point of C on the boundary of O. Then we have the function $T(x) = t$ with $t \in C$ and $x \in h$. Note $T(p_0) = p_0$ and $T(p_2) = p_2$, with radius zero. If T is discontinuous (multivalued) at some x, then O touches C at two or more points, and $x \in M$. Hence $T(x)$ must be continuous. Thus $\{t\}$ must lie on a single connected component of C, an interval, and h is a chord. Since O can never contain p_0 or p_2 in its interior, O lies inside the diameter disk, and hence so must all t. Observe O has strictly higher curvature (i.e. smaller radius) than O_{02}.
The continuity and curvature limit of T implies I can not be perpendicular to h: if it were, then $t_\perp = T(\{x\})$ for some continuous range of x. Continuity of T at the boundary of this range implies the curvature of I at $t_\perp$ is at most that of O_{02}, a contradiction. Hence the $\{x\}$ where $T(x) = t$ is a single point for all t. Hence T is monotonic. This leads to the range of T being I. For curves that are topological circles, the range might instead be I', where $I' = [p_2, p_0]$. Since here the orientation of I is arbitrary, we will label the enclosed interval "I".
Besides $t = T(x)$ being monotonically ordered on I, the distance $\|p_0 t\|$ is also monotonic. It two points t_1 and t_2 of I are equidistant from p_0, then they lie on a circle O_{p0} centered at p_0. Let t_1 be the



(a) CROCODILE point set. (b) CRUST reconstruction. (c) HNN-CRUST reconstruction.

Figure 3: CRUST *only guarantees correct reconstruction for flat angles: consecutive samples must make angles* $> 90°$. *In contrast,* HNN-CRUST *succeeds for sharper angles, requiring only angles* $> 60°$.

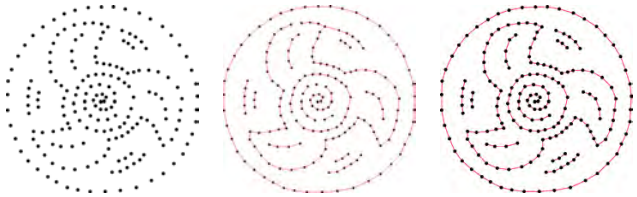


Figure 4: *Open and closed curves. Left: Sample points. Center:* CRUST *reconstruction. Right:* HNN-CRUST *reconstruction.*

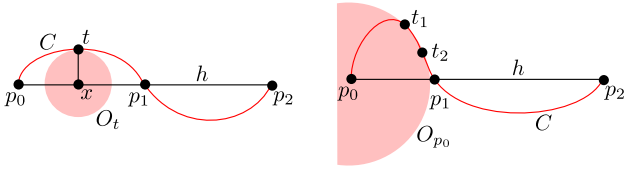


Figure 5: *If a chord does not cross a medial axis point, then the curve interval must lie in the diameter disk. Left, tangent point t varies continuously and monotonically with circle center x along* $\overline{p_0p_1}$. *Right, distance from* p_0 *to t is strictly increasing, else* t_2 *is unreachable.*

one closer to h. (They cannot be equidistant because T is single-valued.) Then any circle in O touching t_2 has t_1 in its interior, a contradiction. By symmetry, $\|p_1t\|$ is also monotonic in x. □

The next two lemmas quantify the fact that consecutive samples are close. We exploit the principle that an ε-sampling ensures that an adjacent sample s_2 is close to s_1 in terms of lfs. From [ABE98] Lemma 1's proof:

Lemma 2 Let $s1, s2$ be adjacent samples in C. For an ε-sampled curve $\exists y \in I[s_1, s_2]$ such that

$$\|s_1s_2\| \leq 2\|s_2y\| = 2\|s_1y\| < 2\varepsilon \text{lfs}(y)$$

$$\text{lfs}(s_1)/(1+\varepsilon) < \text{lfs}(y) < \text{lfs}(s_1)/(1-\varepsilon)$$

Proof ε-sampling ensures $2\|s_1y\| < 2\varepsilon\text{lfs}(y)$ and the triangle inequality provides $\|s_1s_2\| \leq 2\|s_1y\|$. Since lfs is 1-Lipschitz, $\text{lfs}(y) \leq \text{lfs}(s_1) + \|s_1y\|$ replaced with above inequality for $\|s_1y\|$ yields $\text{lfs}(y) < \text{lfs}(s_1)/(1-\varepsilon)$. Also from the 1-Lipschitz property, $\text{lfs}(y) \geq \text{lfs}(s_1) - \|s_1y\|$ replaced with $\|s_1y\|$ from above provides $\text{lfs}(y) > \text{lfs}(s_1)/(1+\varepsilon)$. □

Lemma 3 For adjacent samples s_0, s_1, s_2, let $x \in I[s_0, s_1]$ with $\|s_0x\| = \|s_1x\|$ and $y \in I[s_1, s_2]$ with $\|s_2y\| = \|s_1y\|$. Then,

$$\text{lfs}(x) > \frac{1-\varepsilon}{1+\varepsilon}\text{lfs}(y).$$

For the reach, the situation is considerably simpler.

Lemma 4 For a ρ-sampled curve with consecutive samples s_0 and s_1, $\|s_0s_1\| < 2\rho\ \text{reach}(I_{01}) \leq 2\rho\ \text{lfs}(s_1)$. Moreover, for midpoint x,

$$\text{lfs}(s_1)/(1+\rho) < \text{lfs}(x) < (1+\rho)\text{lfs}(s_1).$$

Proof $\exists x \in I[s_0, s_1]$ such that $\|s_0, x\| = \|x, s_1\| < \rho\ \text{reach}(I_{01}) \leq \rho\ \text{lfs}(s_1)$. The bound on $\text{lfs}(x)$ follows from $\text{reach}(I_{01}) \geq \text{lfs}(x)$ and 1-Lipschitz. □

The next two lemmas show that geodesically distant samples are also far in Euclidean distance. We then relate ρ- and ε-sampling. Finally we provide additional restrictions on the interval between consecutive samples, quantifying how close it must be to a straight line, and additional lower bounds on Euclidean distance.

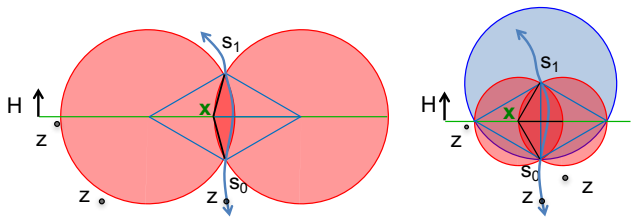
We call a disk with no point of C in its interior "C-free", and a disk with no point of M in its interior "M-free" Recall [ABE98] Lemma 7:

Lemma 5 A disk tangent to a smooth curve C at a point p with *radius* at most $\text{lfs}(p)$ is C-free.

We generalize Lemma 5 to the following.

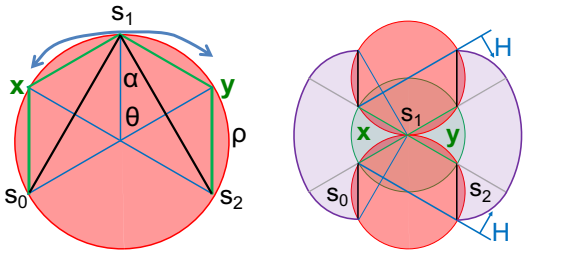
Lemma 6 A rolling tangent circle Rty with center interior to circle $O(y, \text{lfs}(y))$ touches C at a single point p in interval $O(y, \text{lfs}(y)) \cap C$.

Proof By definition, $O(y, \text{lfs}(y))$ is M-free. Following the proof of Lemma 5, growing a tangent disk at y with continuously increasing radius cannot intersect another point of C before the radius reaches $\text{lfs}(y)$, else the center would be a point of M. By the same argument,



(a) $\|x,s_1\| < 0.5\text{lfs}(x)$ (b) $\|x,s_1\| < \text{lfs}(x)$

Figure 6: *Forbidden regions: the red circles are C-free except for $I = [s_0,s_1]$, and I lies in their lune-shaped intersection and $x \in I$ on the green line inside that lune. The lunes bound the extreme cases of constant curvature, where lfs = reach = lfs(x). In (a), the black lines have length* 0.5*lfs and the blue lines lfs. In (b), the black lines have length lfs and the blue triangles are equilateral.*



(a) $\angle s_0s_1s_2$ bound. (b) $s_2 \in H$

Figure 7: *Ranges for x,y,s_1,s_2, angles and H for* ρ*-sampling. The red circles are tangent to C at s_1 with radius lfs(s_1), and are C-free and exclude x,y,s_1, and s_2 from their interior. In (b), the green circle is $O(s_1,\text{lfs}(s_1))$, and contains x and y. Sample s_0 lies in the union of the three purple sectors and one green sector to the left of s_1. Hence H contains the purple and green right sectors and s_2.*

we may now continuously vary the center within $O(y,\text{lfs}(y))$, keeping a continuous tangent at p in an interval around y. □

Combining the idea of growing a tangent ball at y with the fact that local curvature is less than $1/\text{lfs}(y)$ results in the forbidden regions from [ABE98]. We summarize the properties we use in the following lemma.

Lemma 7 The two circles through consecutive samples s_0 and s_1 with the maximum curvature allowed by the sampling condition are C-free except for $I = [s_0,s_1]$. Moreover, I lies in the lune of intersection of the two circles. See Figure 6.

In the following sense, our ρ-sampling is at least as good (i.e. as sparse) as ε-sampling:

Theorem 1 Any $\varepsilon < r$-sampling is also a $\rho < r/(1-r)$-sampling, for $r < 1$. E.g. an $\varepsilon < 0.5$-sampling is also a $\rho < 1$-sampling, and an $\varepsilon < 1/3$-sampling is also a $\rho < 0.5$-sampling.

Proof The proof is the same as the proof of Lemma 2, combined with using Lemma 1 to show distances are monotonic along $I = [s_0s_1]$. For any ε-sampled interval $I = [s_0,s_1]$, we first show $\text{reach}(I) \geq (1-\varepsilon)\text{lfs}(x)$, then show the condition holds $\forall p \in I$. Let $x \in I$ be equidistant from s_0 and s_1. From Lemma 1, $\|xp\| \leq \|xs_0\|$. By 1-Lipschitz, $\text{lfs}(p) \geq \text{lfs}(x) - \|xp\| > (1-\varepsilon)\text{lfs}(x)$. Thus $\text{reach} = \inf_p \text{lfs}(p) \geq (1-\varepsilon)\text{lfs}(x)$. Again by Lemma 1, $\forall p \in [s_0,x], \|ps_0\| \leq \|xs_0\| \leq \varepsilon/(1-\varepsilon)\text{reach}$. The argument for $p \in [x,s_1]$ is the same. Thus, for $r < 1$, any $\varepsilon < r$-sampling is also a $\rho < r/(1-r)$-sampling. □

Corollary 1 $\rho < r/(1-r)$-sampling does not require more samples than $\varepsilon < r$-sampling.

Lemma 8 For a $\rho < 1$-sampling, $\angle s_0s_1s_2 \geq \pi - 4\arcsin\rho/2$ and $\angle xs_1y \geq \pi - 2\arcsin\rho/2$. This is tight for constant curvature.

Proof Consider the C-free tangent disk to s_1 of radius $\text{lfs}(s_1)$. The reach on each interval containing s_1 is at most $\text{lfs}(s_1)$. In Figure 7(a), this leads to $\|xs_1\| \leq \rho\,\text{lfs}(s_1)$, then $\theta = 2\arcsin(\rho/2)$ and $\angle s_0s_1s_2 \geq 2\alpha = \pi - 2\theta$. □

Lemma 9 For an ε-sampled curve, with $\varepsilon < 0.5$, the angle spanned by three adjacent samples is at least $\pi - 4\arcsin(\varepsilon/(2-2\varepsilon))$.

Proof Combine Lemma 8 with Theorem 1. □

Lemma 8 is a restatement of Lemma 10 from [ABE98], with ε replaced by ρ. This is weaker, but, to our knowledge, Lemma 10 from [ABE98] remains unproven. Our corollary, Lemma 9, is an improvement over the bound of $\angle s_0s_1s_2 \geq \pi - 2\arcsin(\varepsilon/(1-\varepsilon))$ in [Dey06]: here $\varepsilon < 0.5$ gives angles at least $60°$, whereas [Dey06] does not provide a lower bound on the angle.

To bound the distance between the reconstruction and the curve, Lemma 13 from [ABE98] applies, which we reformulate:

Lemma 10 For a ρ-sampling of a curve in $\mathbb{R}^2$, with $\rho < 1$, the distance from a point p to a point on the correct polygonal reconstruction of the samples is at most $(\rho^2/2)\text{lfs}(p)$.

Theorem 2 For a $\rho < 0.9$-sampled smooth curve C, the reconstruction algorithm HNN-CRUST outputs the manifold boundary B.

Proof Consider consecutive samples s_0, s_1 and s_2. We wish to show that edges $\overline{s_0s_1}$ and $\overline{s_1s_2}$ are formed. Without loss of generality, let s_0 be the closer of the two samples to s_1. We further consider a sample $z \neq s_0,s_1,s_2$ to investigate the existence of a counter-example.
We first show that s_0 is the nearest neighbor to s_1. We have two cases, depending on whether $\overline{s_1z}$ intersects M. If it does not, then by Lemma 1, z lies on interval I with s_0 (or s_2) between z and s_1, and s_0 (or s_2) is strictly closer to s_1 than z is. Hence z is not a nearest neighbor.
The second case is $\overline{s_1z}$ intersects M. Let q be the closest point of $I_{02} = [s_0,s_2]$ to z. Suppose $q \in I_{12} = [s_1,s_2]$. If q is s_2, then z is closer to s_2 than s_1, and Lemma 7 demonstrates z is farther from s_1 than s_2. Otherwise q is an interior point of I and segment $\overline{zq}$ is perpendicular to I at q. By Lemma 5 it passes through the diameter of a disk tangent to q with diameter $2\text{lfs}(q)$. Then $\|zs_1\| \geq \|zq\| \geq 2\text{lfs}(q)$. But $\text{lfs}(q) \geq \text{reach}(I_2)$ and by Lemma 4 $2\rho\,\text{reach}(I_2) > \|s_1s_2\|$. Hence $\|zs_1\| > \|s_1s_2\| \forall \rho \leq 1$. Using the same arguments, if $q \in [s_0,s_1]$ then $\|zs_1\| > \|s_0s_1\|$.
We have now shown that s_0 is the nearest neighbor to s_1, and it remains to show that s_2 is the half neighbor. From Figure 7(b), the admissible region for s_1 leads to $s_2 \in H$ as follows. As s_0 varies along the boundary of a red circle, H rotates around the center of the circle, but never contains the admissible region for s_2. As s_0

moves off a red circle, H just retreats farther from s_2's admissible region.

Thus, we need only show that no other sample z in H is closer. While showing that s_0 was the nearest neighbor, we already established that any z was farther than $\|s_1s_2\|$ except perhaps when $\overline{zs_1} \cap M \neq \emptyset$ and its closest point of I is $q \in [s_0, s_1]$. For $\rho < 0.5$, the remainder is trivial because $\|zs_1\| \geq \text{lfs}(s_1) > 2\rho\|s_1s_2\|$. For larger ρ, the main idea of the proof is to use rolling tangent balls to cover the part of $O(s_1, \|s_1s_2\|)$ in H. From Lemma 7, $I = [s_0, s_2]$ is restricted to lie in the union of two lunes, which provides a lower bound on the radii of the rolling tangent balls from Lemma 6. Hence the balls are large and cover the portion of the circle $O(s_1, \|s_1s_2\|)$ in H. Unfortunately, we do not have a closed-form algebraic description of this fact. Instead, we have a computer assisted proof. We consider the possible ranges of positions, with ratio=$\|x, s_1\|/\|s_1, y\| \in [0,1]$ and the tangent angles between $\overline{xs_1}$ and $\overline{s_1y} \in [0°, 53.5°]$ (Lemma 8). We divide each of these three ranges into small intervals. For all feasible combinations of intervals, we take the worst case value for each quantity independently when used. For all ranges we construct a collection of rolling tangent circles that covers $O(s_1, \|s_1s_2\|)$. Figure 8 provides a few representative examples. These figures and all other feasible combinations can be reproduced with a matlab script available online. □

Theorem 3 For an ε-sampled smooth curve C, with $\varepsilon < 0.47$, HNN-CRUST outputs the manifold boundary B.

Proof This follows immediately from Theorems 1 and 2. □

8. Results

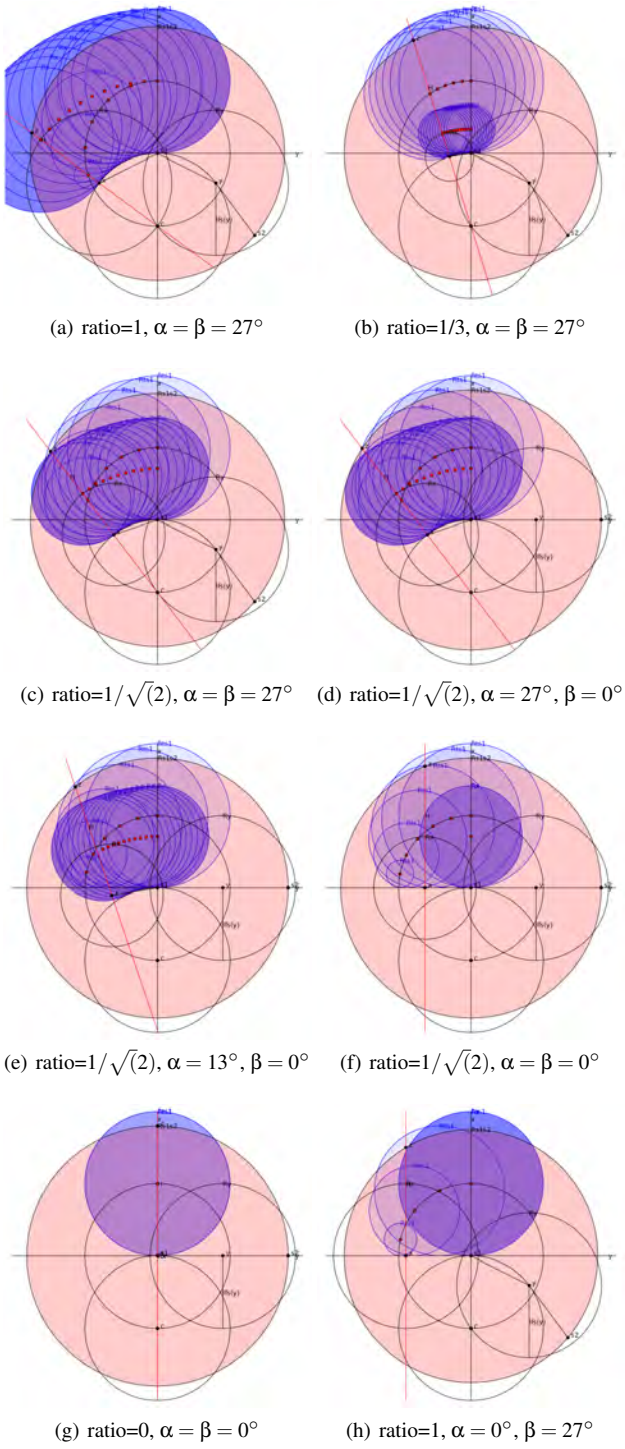
8.1. Comparison of HNN-CRUST

Figure 4 shows that unlike the CRUST [ABE98], our proposed algorithm reconstructs sharp corners up to 60° and handles close curves well. Our reconstruction algorithm is local and therefore scales well to large point sets. HNN-CRUST also handles open curves gracefully. It only outputs edges which are reconstructed bijectively, i.e. are consistent from both end points, in order to avoid catastrophic failure. We provide open source code for this algorithm that reproduces figures and tables of this paper: `https://github.com/stefango74/hnn-crust-sgp16`.

8.2. Comparison of $\rho < 0.9$-sampling

Algorithm	Sampling condition	Bound	min α	circle	par.
GATHANG	$\|p, s_{[0\|1]}\| < \varepsilon\,\text{lfs}(p)$	$\varepsilon < 0.5$	$> 150°$	12	2
CRUST	$\exists s : \|p, s\| < \varepsilon\,\text{lfs}(p)$	$\varepsilon < 0.2$	$> 157°$	15.7	5
NN-CRUST	$\exists s : \|p, s\| < \varepsilon\,\text{lfs}(p)$	$\varepsilon < \frac{1}{3}$	$> 142°$	9.4	3
NN-CRUST*	$\exists s : \|p, s\| < \varepsilon\,\text{lfs}(p)$	$\varepsilon < 0.4$	$> 134°$	7.8	2.5
[Len06]*	—"—	$\varepsilon < 0.48$	$> 124°$	6.5	2.1
HNN-CRUST	—"—	$\varepsilon < 0.47$	$> 126°$	6.6	2.1
HNN-CRUST	$\exists s : \|p, s\| < \rho\,\text{reach}(I(p))$	$\rho < 0.9$	$> 73°$	3.4	1.1

Table 1: *Bounds for differing sampling conditions (*=not proven), guaranteed minimum angles spanned between three adjacent samples for constant curvature and based on those the averaged number of points required to* sample a circle *and* parallel lines with length equal to their distance. *Here, $p \in C$ is in the curve interval $I(p)$ between adjacent samples s_0 and s_1, and s is any sample.*



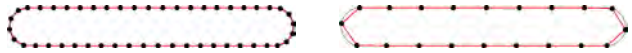
(a) ratio=1, $\alpha = \beta = 27°$ (b) ratio=1/3, $\alpha = \beta = 27°$

(c) ratio=$1/\sqrt{(2)}$, $\alpha = \beta = 27°$ (d) ratio=$1/\sqrt{(2)}$, $\alpha = 27°$, $\beta = 0°$

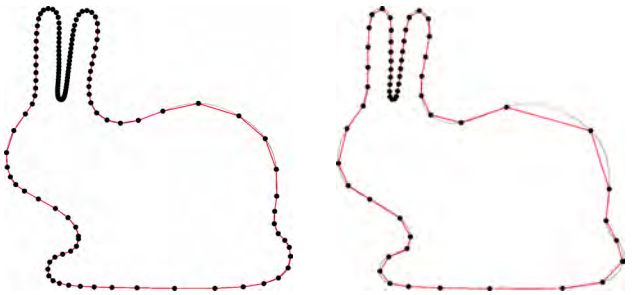
(e) ratio=$1/\sqrt{(2)}$, $\alpha = 13°$, $\beta = 0°$ (f) ratio=$1/\sqrt{(2)}$, $\alpha = \beta = 0°$

(g) ratio=0, $\alpha = \beta = 0°$ (h) ratio=1, $\alpha = 0°$, $\beta = 27°$

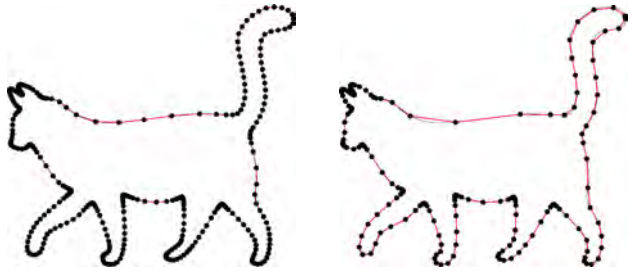
Figure 8: *For ρ-sampling with $\rho = 0.9$, s_2 is the half neighbor because rolling tangent balls cover $O(s_1, \|s_1s_2\|) \cap H$ (red disk right of red line in quadrant II). Their radii are bounded below by the curve (or its lower bound approximation, the x-axis). Here ratio=$\|x, s_1\|/\|s_1, y\| \in [0,1]$ and $\alpha, \beta \in [0°, 27°]$. We assign the tangent of C at s_1 as the x-axis, with α its angle with $\overline{s_1x}$ and β its angle with $\overline{s_1y}$.*



(a) PARALLEL LINES.



(b) BUNNY clipart consisting of cubic bezier curves.



(c) CAT clipart consisting of cubic bezier curves.

Figure 9: *Curves with polygonal reconstruction (red). Left: An* $\varepsilon < \frac{1}{3}$*-sampling. Right:* $\rho < 0.9$ *permits much sparser sampling.*

In Table 1 we compare sampling conditions w.r.t. their minimum angle and how many samples this represents on a circle or on parallel lines. Note that we derive the minimum angle for all conditions from the given bounds, except for GATHANG [DW02], which relies on additional conditions to handle sharp corners. Note especially that for constant curvature (circular arcs, parallel lines), our proposed $\rho < 0.9$-sampling requires just little more than a third of the samples than $\varepsilon < \frac{1}{3}$-sampling.

We implemented a sampling algorithm which can apply both ε-sampling and ρ-sampling and outputs a number of samples on the input curve. The parameters ε, ρ and d (the Hausdorff distance between original curve and polygonal reconstruction) can be varied. To verify whether the edges in the reconstruction are correct, they are output as well (see Figures 9, 10 and 11). As input curves we use cubic Bezier curves and subsample them very densely to approximate the needed lfs closely at these curve points. The implementation is also available as open source online.

Figure 9 visualizes sampling different curves with $\varepsilon < \frac{1}{3}$-sampling and $\rho < 0.9$-sampling. The number of respective samples together with $\varepsilon < 0.47$-sampling are shown in Table 2.

Figure 10 shows the advantage of $\rho < 0.9$-sampling over $\varepsilon < \frac{1}{3}$-sampling when the sampling must also ensure that the reconstructed polygon lies within Hausdorff distance d of the original curve.

Table 2 shows that a $\rho < 0.9$-sampling requires many fewer samples than an $\varepsilon < 0.47$-sampling, while still guaranteeing reconstruction with HNN-CRUST, approaching half of what $\varepsilon < \frac{1}{3}$-sampling produces. Since for curve intervals of constant local feature size the *reach* is equal to this lfs, circular arcs or parallel lines require only exactly half the samples in the limit. The lower bound of $\rho < 0.9$-sampling is therefore $\varepsilon < 0.9$-sampling, the upper bound $\varepsilon < 0.47$-sampling as shown in Corollary 1.

Model	$\rho < 0.9$	$\varepsilon < 0.47$	$\varepsilon < \frac{1}{3}$
PARALLEL	20	35 (75%)	48 (140%)
TEASER	26	43 (65%)	61 (135%)
BUNNY	58	94 (62%)	131 (126%)
CAT	180	254 (41%)	356 (98%)

Table 2: *Number of samples required for the given sampling conditions (* $*$ *= in the limit) for example curves and the % of redundant samples compared with* $\rho < 0.9$ *in brackets (see Figures 1 and 9).*

The more drastically the lfs changes, the more samples have to be placed, approximating the limit of $\varepsilon < 0.47$-sampling.

Hausdorff distance	$\rho < 0.9$	$\varepsilon < 0.47$	$\varepsilon < \frac{1}{3}$
∞	58	94 (62%)	131 (126%)
1%	60	94 (57%)	131 (118%)
0.3%	73	99 (36%)	133 (82%)
0.1%	105	123 (17%)	148 (41%)
0.03%	173	186 (8%)	204 (18%)

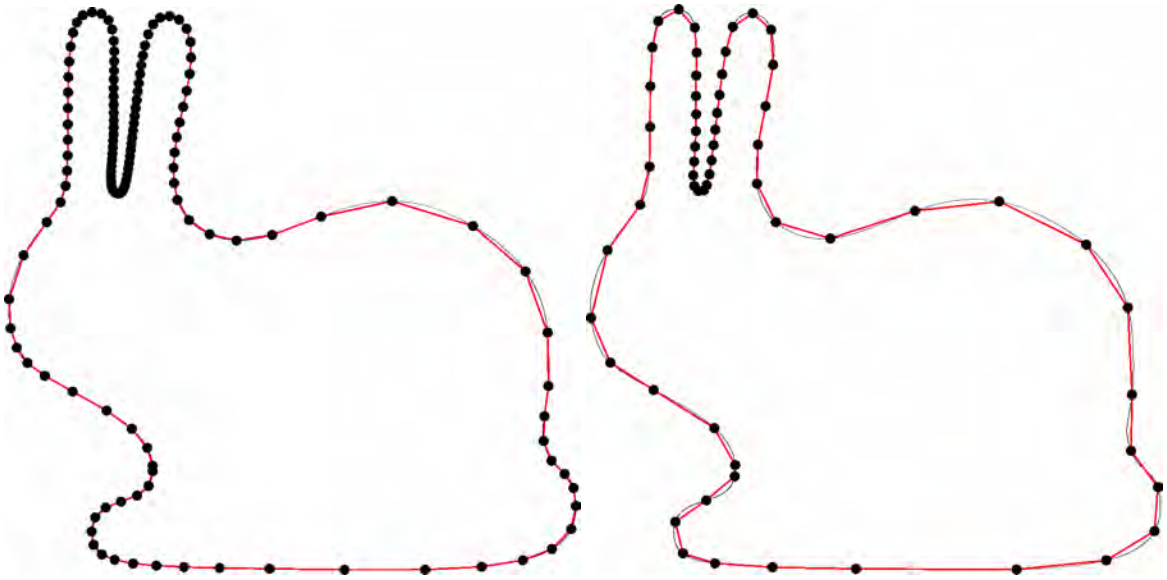
Table 3: *Number of samples required for the given sampling conditions for the* BUNNY *curve and given Hausdorff distance limit in terms of maximum point set dimension, the % of redundant samples compared with* $\rho < 0.9$ *in brackets.*

Table 3 shows how sample redundancy for ε-samplings decreases as the required Hausdorff distance between the reconstruction and original curve becomes smaller than the feature size. Note that for the BUNNY in Figure 9(b), the $\rho < 0.9$-sampling requires just adding 2 samples to achieve the 1% reconstruction error (see Figure 10).

The limits of HNN-CRUST are shown in the lower half of Figure 11, where the sampling condition is violated by too close curves or too sharp corners, while its top half shows that *GathanG* yields for such cases rather arbitrary results due to a lack of an intuitively understandable sampling condition. Those can be handled by specialized algorithms such as GATHANG [DW02], which rely on heuristics or global data structures such as Delaunay triangulation. Their disadvantage is that due to the heuristic criteria, they cannot give as good guarantees w.r.t. angles as ours. Also the required global data structures cannot be well partitioned for local construction, such as is possible for the kd-tree we use for determining nearest neighbors.

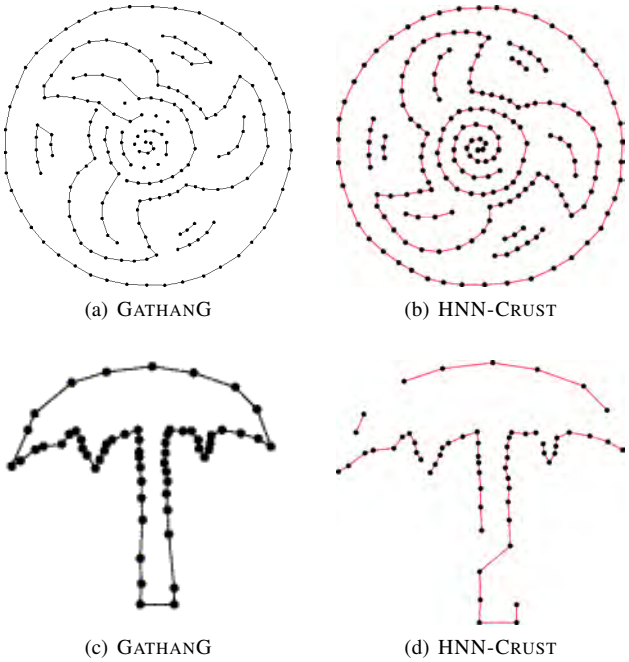
9. Conclusion and Future Work

Both improving the existing bound for ε-sampling from $\varepsilon < \frac{1}{3}$ to $\varepsilon < 0.47$ and introducing a new condition for sampling smooth



(a) $\varepsilon < \frac{1}{3}$ requires 131 samples. (b) $\rho < 0.9$ needs just 60 samples.

Figure 10: *Sampling the original curve and limiting its reconstruction to a Hausdorff distance of 1% of its total extent: Here,* $\varepsilon < \frac{1}{3}$ *requires more than double the samples than* $\rho < 0.9$*, which are redundant since not contributing to the reconstructed geometry within the specified error.*



(a) GATHANG (b) HNN-CRUST

(c) GATHANG (d) HNN-CRUST

Figure 11: *Top left:* GATHANG *connects some edges seemingly arbitrary compared to Local* HNN-CRUST *on the right. Bottom left:* GATHANG *handles very sharp corners and undersampling by exploiting the global context. Right: Local* HNN-CRUST *indicates (by producing leaf vertices on an assumed closed curve) where* $\rho <$ 0.9 *is violated.*

curves, ρ-sampling, has enabled us to prove a much tighter bound in terms of local sampling density. That new bound, $\rho < 0.9$, permits reconstruction of smooth curves with our proposed simple and fast algorithm HNN-CRUST. We believe that 0.9 is close to tight, based on Figure 8(d). The bound allows for much more sparse sampling while keeping the geometric approximation of the reconstructed polygon to the original curve. The improved ε-sampling bound already requires up to 45% fewer samples (in the limit, for constant curvature). Additionally, based on that new sampling condition, smooth curves can be reconstructed from even fewer points, typically half of the state-of-the-art bound, in the limit roughly one third. We are currently working on framing conditions to enhance our sampling framework to support non-smooth curves, as [OM13] shows they can be reconstructed for extremely sparse sampling.

Further we believe that it can be extended to handle noisy samples with outliers in the sense of [DS06]. Another work in progress is the extension of the reconstruction algorithm into $\mathbb{R}^3$ for surface reconstruction with a similar condition for the sampling required on a smooth boundary, together with the above enhancements. While the edge-pairs reconstructed at points in $\mathbb{R}^2$ correspond to closed triangle fans in $\mathbb{R}^3$, the output of the reconstruction algorithm does not match, as shown in [OMW13]. Flat tetrahedra can lie parallel to the surface (slivers) and so an additional condition is required to yield a unique triangulation.

Acknowledgements

We thank Tamal Dey and Marshall Bern for helpful discussions about edge angles. This work has been funded by FWF grant P24600-N23 and FP7-ICT project 323567 (HARVEST4D). San-

dia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[ABE98] AMENTA N., BERN M. W., EPPSTEIN D.: The crust and the beta-skeleton: Combinatorial curve reconstruction. *Graphical Models and Image Processing 60*, 2 (1998), 125–135. 2, 3, 4, 5, 6, 7

[Alt01] ALTHAUS E.: *Curve Reconstruction and the Traveling Salesman Problem*. Doctoral dissertation, Universität des Saarlandes, 2001. 2

[Att97] ATTALI D.: r-regular shape reconstruction from unorganized points. In *Symp. on Computational Geometry* (1997), pp. 248–253. 2

[Blu67] BLUM H.: A Transformation for Extracting New Descriptors of Shape. In *Models for the Perception of Speech and Visual Form*, Wathen-Dunn W., (Ed.). MIT Press, Cambridge, 1967, pp. 362–380. 3

[Dey06] DEY T. K.: *Curve and surface reconstruction: algorithms with mathematical analysis*, vol. 23. Cambridge University Press, 2006. 2, 6

[DK99] DEY T. K., KUMAR P.: A simple provable algorithm for curve reconstruction. In *Proc. 10th ACM-SIAM SODA '99* (1999), pp. 893–894. 1, 2, 3

[DMR99] DEY T. K., MEHLHORN K., RAMOS E. A.: Curve reconstruction: Connecting dots with good reason. *In Proc. 15th ACM Symp. Comp. Geom 15* (1999), 229–244. 2

[DS06] DEY T. K., SUN J.: Normal and feature approximations from noisy point clouds. In *Proceedings of the 26th int'l. conference on Foundations of Software Technology and Theoretical Computer Science* (Berlin, Heidelberg, 2006), FSTTCS'06, Springer-Verlag, pp. 21–32. 9

[DT14] DUARTE P., TORRES M. J.: Smoothness of boundaries of regular sets. *Journal of mathematical imaging and vision* (2014), 1–8. 2

[DT15] DUARTE P., TORRES M. J.: r-regularity. *Journal of Mathematical Imaging and Vision 51*, 3 (2015), 451–464. 2

[DW02] DEY T. K., WENGER R.: Fast reconstruction of curves with sharp corners. *Int. J. Comp. Geom. Appl. 12*, 5 (2002), 353 – 400. 2, 8

[EKS83] EDELSBRUNNER H., KIRKPATRICK D. G., SEIDEL R.: On the shape of a set of points in the plane. *IEEE Trans. Inf. Theor. IT-29*, 4 (1983), 551–559. 2

[Fed59] FEDERER H.: Curvature measures. *Transactions of the American Mathematical Society 93*, 3 (1959), pp. 418–491. 3, 4

[FMG94] FIGUEIREDO L. H. D., MIRANDAS GOMES J. D.: Computational morphology of curves. *Vis. Comp. 11*, 2 (1994), 105–112. 2

[FR01] FUNKE S., RAMOS E. A.: Reconstructing a collection of curves with corners and endpoints. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2001), SODA '01, Society for Industrial and Applied Math., pp. 344–353. 2

[Gol99] GOLD C.: Crust and anti-crust: a one-step boundary and skeleton extraction algorithm. In *Proc. of the 15th ann. Symp. on Computational geometry* (New York, NY, USA, 1999), SCG '99, ACM, pp. 189–196. 2

[KR85] KIRKPATRICK D. G., RADKE J. D.: A framework for computational morphology. *Computational Geometry* (1985), 217–248. 2

[Len06] LENZ T.: How to sample and reconstruct curves with unusual features. In *Proceedings of the 22nd European Workshop on Computational Geometry (EWCG)* (Delphi, Greece, March 2006). 2, 7

[LKvK*14] LÖFFLER M., KAISER M., VAN KAPEL T., KLAPPE G., VAN KREVELD M., STAALS F.: The Connect-The-Dots family of puzzles: design and automatic generation. *ACM Transactions on Graphics 33*, 4 (July 2014), 72:1–72:10. 2

[NZ08] NGUYEN T. A., ZENG Y.: Vicur: A human-vision-based algorithm for curve reconstruction. *Robotics and Computer-Integrated Manufacturing 24*, 6 (2008), 824 – 834. FAIM 2007, 17th International Conference on Flexible Automation and Intelligent Manufacturing. 2

[OM13] OHRHALLINGER S., MUDUR S.: An efficient algorithm for determining an aesthetic shape connecting unorganized 2d points. In *Comp. Graph. Forum* (2013), vol. 32, Wiley Online Library, pp. 72–88. 2, 4, 9

[OMW13] OHRHALLINGER S., MUDUR S., WIMMER M.: Minimizing edge length to connect sparsely sampled unstructured point sets. *Computers & Graphics* (2013). 9

[Rup93] RUPPERT J.: A new and simple algorithm for quality 2-dimensional mesh generation. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms* (Philadelphia, PA, USA, 1993), SODA '93, Soc. for Industr. and Appl. Math., pp. 83–92. 3

[ST09] STELLDINGER P., TCHERNIAVSKI L.: Provably correct reconstruction of surfaces from sparse noisy samples. *Pattern Recognition 42*, 8 (2009), 1650–1659. 2

[Ste08] STELLDINGER P.: Topologically correct surface reconstruction using alpha shapes and relations to ball-pivoting. In *Pattern Recognition, 2008. ICPR 2008. 19th Int'l Conference on* (2008), IEEE, pp. 1–4. 2

[ZNYL08] ZENG Y., NGUYEN T. A., YAN B., LI S.: A distance-based parameter free algorithm for curve reconstruction. *Comput. Aided Des. 40*, 2 (2008), 210–222. 2

Multi-Material Adaptive Volume Remesher

Noura Faraj[a], Jean-Marc Thiery[a], Tamy Boubekeur[a]

[a]*LTCI, CNRS, Télécom-ParisTech, Université Paris-Saclay*

Abstract

We propose a practical iterative remeshing algorithm for multi-material tetrahedral meshes which is solely based on simple local topological operations, such as edge collapse, flip, split and vertex smoothing. To do so, we exploit an intermediate implicit feature complex which reconstructs piecewise smooth multi-material boundaries made of surface patches, feature edges and corner vertices. Futhermore, we design specific feature-aware local remeshing rules which, combined with a moving least square projection, result in high quality isotropic meshes representing the input mesh at a user defined resolution while preserving important features. Our algorithm uses only topology-aware local operations, which allows to process difficult input meshes such as self-intersecting ones. We evaluate our approach on a collection of examples and experimentally show that it is fast and scales well.

Keywords: Multi-material tetrahedral mesh ; volume remeshing ; feature preservation

1. Introduction

Multi-material volumetric datasets are widely used to study physical phenomena, model physically-plausible shapes or fabricate/print real objects from digital ones. For instance, a broad range of medical simulations steam from the increasing number of available 3D anatomical images. Those datasets are typically composed of voxel grids acquired using Magnetic Resonance Imaging (MRI) or scanners and accurately labeled by professionals - i.e., each voxel is assigned with a label representing a single material (e.g., organ). The union of these components forms the simulation *domain*, where each material is represented by a single *subdomain*. However, in practice, simulations are often designed to run on a mesh of the input domain, using Finite Elements Methods (FEM). Employing large polyhedra - typically tetrahedra - for constant material regions reduces drastically the computation costs while preserving a good approximations. Indeed, the result of a simulation depends on the domain representation accuracy and the quality on the input mesh since its stability usually depends on the size and shape of its tetrahedra [1]. For instance, tetrahedra with small dihedral angles cause negative volumes under small perturbations, while large angles strongly increase the simulation errors.

As the material is supposed constant within a subdomain, the critical features to preserve during the meshing process reside at the interfaces between labels since they indicate the shape boundaries and the junctions between subdomains. These features can take three forms [2]: (i) the surfaces patches between two labels (*2-junctions*), (ii) the edges between three or more labels (*1-junctions*) and (iii) the corner vertices between four or more labels (*0-junctions*). Generating tetrahedral meshes at the suitable resolution for simulation while accurately capturing such features is a tedious task. Ideally, one could generate high-quality meshes at several resolutions, exploiting each time the previously finer meshes to generate the coarser and trading feature preservation for regularization. Indeed, a regularization process is unavoidable, as smooth boundaries are mandatory for visualization and stability purpose. As the seminal discretization (e.g., from the input image) can fail at meeting these constraints, a remeshing step (i. e. optimization and/or simplification) is often necessary.

Contributions. We propose a simple and practical iterative remeshing algorithm for 3D triangulations, which provides high-quality meshes at a chosen resolution while preserving features such as multi-material boundaries. The user can efficiently reach the desired resolution by adjusting the target edge length: in particular, the mesh is processed iteratively until the size constraint is met without starting over from the input mesh, which is essential during fine-tuning remeshing sessions. Our algorithm allows generating uniform as well as adaptive meshes, for which the spatially-varying resolution is driven by a sizing (scalar) field that can be user-defined, e. g. to generate a dense mesh in regions of interest, or based on a distance field. In particular, such a field may be generated from the subdomain boundaries, yelding elements with increasing size when located away from the boundaries, therefore resulting in an isotropic adaptive mesh. This is especially effective to minimize the number of tetrahedra while preserving accurate boundaries.

In order to provide our algorithm with a structured decomposition of the multi-material domain, we use a *feature complex*, similar to the one proposed by Dey et al. [3], but equipped with a *Moving Least Square* (MLS) geometric definition derived from *Hermite Point Set Surfaces* [4]. Doing so, we decorellate the structured geometry of the domain from the mesh and can perform feature-dependent topological operations coupled with a hierarchical MLS smoothing. We can preserve additional features which are either provided by the user, as a set of polylines, or detected on the domain boundary (e.g., surface sharp features). On the contrary to Delaunay-based methods, ours al-

Figure 1: Our method remeshes complex multi-material tetrahedral meshes, with high geometric quality and exact topology preservation.

lows to efficiently generate meshes at any resolution by only changing the target edge length since the MLS representation of the feature complex enables us to mesh the domain geometry directly, even at low meshing resolution.

By using only topology-aware local operations, our ~~technique~~ can process difficult input meshes such as self-intersecting ones, whereas Delaunay-based techniques will necessarily glue intersecting parts. Last, our remeshing method can be used as a complement to any existing meshing method in order to generate a mesh suited to the users needs. Therefore any structured mesh can be processed with our algorithm. For instance, we apply our method on trivial high-resolution tetrahedral mesh generated from segmented voxel grids.

2. Background

The generation, simplification and refinement of high-quality tetrahedral meshes are very active research fields. Here, we give a non exhaustive overview of the existing methods and focus on the ones that are able to handle multi-material inputs.

Meshing. We can group the meshing methods in three main categories: Delaunay-based, lattice-based and variational.

Delaunay-based methods start by distributing a set of points over the input domain. Then a Delaunay refinement process [5, 6, 7] is used to add *Steiner* points to the triangulation until the input approximation, elements shape and quality criteria are met. This process has first been proposed for domains bounded by a smooth surface [8, 9] and further extended for piecewise smooth boundaries [10]. In order to handle the input domain sharp features, the authors propose to build a *Piecewise Smooth Complex* (PSC), which is composed of surface patches, curves (intersections of surface patches) and points (intersections of curves). Protecting balls of varying size are defined around the edges to preserve the features of this complex during the refinement process [11, 12]. Building upon such a complex, Tournois et al. [13] propose a method coupling refinement and optimization strategies to guide the insertion of Steiner points and directly obtain high-quality meshes. These principles are extended to handle multi-material domains [14] and preserve their 1- and 0-junctions using similar protecting balls [2]. More recently, Dey et al. [3] propose a PSC composed of multi-material junctions, and we use a similar complex to identify the features to preserve. While those methods allow generating high-quality meshes, the implementation of the protecting ball paradigm remains highly non trivial in a multi-material setting.

Lattice-based meshing approaches are inspired by the Marching Cubes algorithm [15] applied to multi-material boundaries [16]. An initial high resolution regular mesh is generated from the volume data and the elements are split according to precomputed boundary configurations until the input domain is well represented. Those methods tend to produce dense meshes and ill-shaped tetrahedra near the boundaries. Furthermore, their resolution depends on the input grid one. The *isosurface stuffing* method [17] uses a similar paradigm to represent single material domains but guarantees theoretical bounds on the tetrahedra' dihedral angles. Inspired by this strategy, Bronson et al. [18] offer similar guarantees for labeled volume data.

Variational approaches [19, 20, 21, 22] insert particles or an initial mesh in the input domain and use a non-linear energy optimization to tailor the feature-aware point distribution. These methods provide high-quality results but are strongly dependent of the initial setting and are computationally expensive.

Poorly-shaped elements. Fig. 2 presents a common classification of tetrahedral degenerencies. In 2D, the quality of the triangle shape is defined as the ratio of the shortest and longest edge because it relates to the minimum angle, but this property is no longer true in 3D. A poorly-shaped tetrahedron can have edges with similar length, e. g. for *slivers*, which are almost flat tetrahedra. A tetrahedron's quality is thus better described by its minimal dihedral angle and its radius-ratio (the ratio between the inscribed and circumscribed spheres' radii). Note that a regular tetrahedron has dihedral angles equal to 70.5 degrees.

Remeshing and quality improvement. Since feature preservation and quality constraints are tedious to combine, ill-shaped tetrahedra are likely to be generated during the meshing process. For instance, Delaunay-based refinement processes do not prevent the apparition of slivers - tetrahedra meeting the Delaunay constraints but with poor quality - and induce an unavoidable post-processing step. A first solution, called sliver *exudation* [23], turns the triangulation into a weighted Delaunay triangulation to address this problem. An alternative approach is to

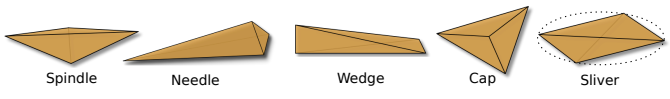


Figure 2: **Ill-shaped tetrahedra:** *needles* and *wedges* (large longest to shortest edge ratio), *caps* (small radius-ratio and three large dihedral angle) and *slivers* (almost flat but with edges of similar length).

perturb slivers through vertex relocation and Delaunay connectivity update [24]. In a more general setting, the quality can be improved using optimization based-smoothing and local topological operations such as edge flip or removal [25]. This approach was further improved by Klingner et al. [26] through additional operations such as vertex insertion, multi-face removal and a roll back mechanism.

To cover a large range of resolutions and define more densely meshed regions of interest, both simplification and refinement schemes - generating high-quality meshes - are necessary when remeshing. For visualization purposes, the simplification of tetrahedral meshes is widely used through edge contraction [27, 28], similar to surface simplification techniques [29, 30] or point sampling [31, 32]. Mesh adaptation is widely used to increase simulation accuracy by improving the mesh quality to better capture the studied physical phenomena. Local operations are used in order to modify the mesh and satisfy a given mesh metric field [33, 34, 35]. Unfortunately, only a limited number of methods are generalized to handle multi-material meshes and 1- and 0-junctions. Cutler et al. [36] propose a method to generate high-quality segmented meshes at different resolutions by performing local topological operations to meet the quality and edge length criteria. While this method allows preserving boundary surfaces using a volume based error metric, 1- and 0-junctions are not taken into account. To the best of our knowledge, only ~~the following ones~~ account for 1- and 0-junctions [37, 38]. The authors propose *link conditions* defining if an edge can be collapsed without changing the topology of the features of different degrees.

Similar issues are tackled for the remeshing of triangular *surface* meshes. In particular, following [39, 40], Botsch and Kobbelt [41] propose an iterative remeshing method to generate isotropic high-quality triangular meshes using simple local operations performed in a predefined order.

As discussed in this section, no existing method allows generating high-quality meshes at various resolutions efficiently while preserving multi-material features. Most existing meshing processes allow generating a mesh within several minutes or hours, and in some cases to refine this mesh (Delaunay triangulation), but not to simplify it. Simplification methods can preserve these features but do not meet the quality requirements.

Beyond the contributions listed earlier, our volume remeshing method builds upon several previous ideas. First, the core of our approach is inspired from the *surface* remeshing method of Botsch and Kobbelt [41], using local connectivity modifications only [39, 40] which have proved to be highly efficient in the surface case. Second, we avoid self-intersections robustly by adding imaginary tetrahedra to the triangulation [42]. Third, to unify the local remeshing rules sustaining our algorithm, we perform all operations on an extended complex by linking the outer facets of the triangulation to a dummy vertex.

3. Our algorithm

3.1. Overview

Given a target edge length l, our remeshing method for multi-domain tetrahedral meshes can be summarized as follows:

Preprocess detect the boundaries and features to preserve, add imaginary tetrahedra to prevent self-intersections.

1 **split** any edge longer than $e_{\max}$,

2 **collapse** any edge shorter than $e_{\min}$,

3 **flip** edges to minimize the average valence and to optimize locally the dihedral angle distribution,

4 **filter** to relocate vertices, taking features into account,

5 go to **1** unless the target resolution is reached,

Postprocess remove slivers and improve mesh quality.

For a target edge length, we use constant values $e_{\max} = 4l/3$ and $e_{\min} = 4l/5$. These thresholds avoid *looping*, and splitting an edge verifying $|e_{max} - l| > |\frac{1}{2}e_{max} - l|$ and collapsing an edge verifying $|e_{min} - l| > |\frac{3}{2}e_{min} - l|$ reduces the deviation from the target length, see [43] for more details. Optionally, we can tailor l in a spatially-varying fashion w.r.t. a sizing field capturing either the distance to the boundary or user-defined regions of interest. In the particular case of an input mesh having already the aimed resolution and processed solely for regularization and quality improvement purposes, the target length is set to a value which is slightly lower than the current average edge length, to introduce perturbations in the optimization [41].

3.2. Representation

We note the set of labels associated with either an input multi-material 3D triangulation as $\mathcal{L} = \{l_n\}_{n \in I_\mathcal{L}} \subset \mathbb{Z}$. By convention, null values represent the background, i.e., the parts of the data that do not belong to the represented domain.

3.2.1. Labeled tetrahedral mesh

The mesh is noted $M = \{V, E, T\}$ with $V = \{v_i\}_{i \in I_V} \subset \mathbb{R}^3$ its vertices, $E = \{e_{ij}\}$ its edges connecting adjacent vertices v_i and v_j and $T = \{t_k\}_{k \in I_T}$, its tetrahedra indexed over V. We call the triangular faces of the tetrahedra *facets*. We note $T_1(v_i)$ (resp. $F_1(v_i)$) the set of tetrahedra (resp. facets) incident to a vertex v_i and $T_1(e_{ij})$ (resp. $F_1(e_{ij})$) the set of tetrahedra (resp. facets) around an edge e_{ij}.

The input mesh is typically composed of n subdomains, with $L(t_k) = l_i$ denoting the label associated with a given tetrahedron t_k. We add a special imaginary subdomain to ensure that the remeshing process will not introduce self-intersections of the represented domain [42]. The additional *imaginary* tetrahedra have a null label $L(t_k) = 0$ (i.e., background) and will be processed like any other subdomain. As illustrated in Fig. 3,

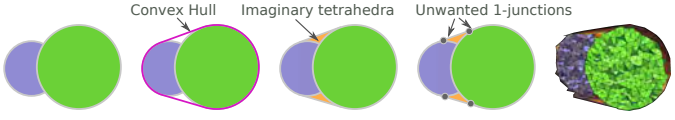


Figure 3: Directly filling the space delimited by the convex hull and the mesh boundaries with imaginary tetrahedra can create unwanted features.

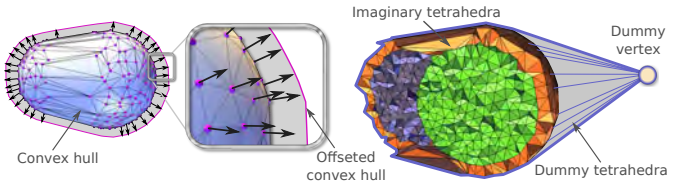


Figure 4: **Pre-processing.** Addition of a layer of imaginary tetrahedra and extension of the complex by connecting the outer facets to the dummy vertex.

naively filling the convex hull of the input vertices generates unwanted features. To tackle this issue, the input mesh is embedded into an inflated convex hull to ensure that the input domain is surrounded by at least one layer of tetrahedra (see Fig. 4). To do so, we duplicate the elements of V laying on the convex hull and displace them in their outer normal direction before triangulating them, using a Restricted Delaunay Triangulation preserving the original mesh outer boundary. The displacement factor is typically set to 4% of the domain's bounding box. To process the outer boundary like any other, i. e. unifying the representation of inter-domain boundaries and 3D surfaces, we connect the outer boundary facets (i.e., of the offseted convex hull) to a dummy vertex creating *dummy* tetrahedra marked with a negative label.

3.3. Elements notations

Here, we define the simplices notations used in the remainder of the document (see Fig. 5). Facets shared by two tetrahedra that belong to different subdomains are *boundary facets*. The vertices (resp. edges) of those facets are *boundary vertices* (resp. *boundary edges*). For each vertex v_i, we note $S(v_i) \subset L$ the set of labels incident to v_i:

$$S(v_i) = \{L(t_k)_{t_k \in T_1(v_i)}\}.$$

Similarly, we define $S(e_{ij})$ for an edge. *Volume* (resp. *boundary*) vertices verify $|S(v_i)| = 1$ (resp. $|S(v_i)| > 1$), except for the dummy vertex. As shown in Fig. 5, we identify four types of edges:

- *volume edges* connect two volume vertices (green, red),
- *mixed edges* connect a volume and a boundary vertex (gray),
- *boundary* edges connect two boundary vertices and have incident boundary facets (pink and blue),
- all other edges are *critical* edges (yellow).

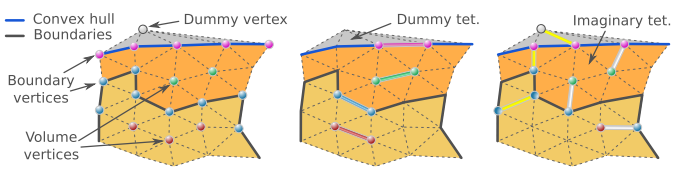


Figure 5: **Classification.** (Left) Vertex types, (Center) Boundary and volume edges and (Right) Mixed and critical edges.

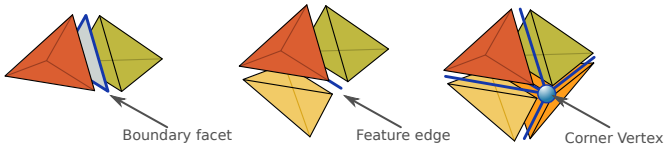


Figure 6: **Feature elements** of various dimension detected using solely incident subdomain indices.

3.4. Feature detection

In order to conform to the input boundaries and perform feature-aware operations, we identify feature elements of different dimensions that together form a *feature complex* similar to a point-sampled cell complex [44] or a PSC [3].

Feature elements. The *boundary facets*, which are facets between tetrahedra of different labels, are the complex elements of dimension 2. The *feature edges*, which are boundary edges at the intersection of three or more labels ($|S(e_{ij})| > 2$) are the elements of dimension 1. Vertices with three or more incident labels ($|S(v_i)| > 3$) and at least three feature edges in their one-ring are *corner vertices* (see Fig. 6) and are 0-dimensional elements.

The input domain's n-junctions - with n the dimension in the feature complex - are represented in the mesh (see Fig. 7) as follows:

- *2-junctions* are sets of connected boundary facets located at the interface between two subdomains. Each 2-junction forms a triangle surface patch, where its orientation is deduced from one of its incident subdomains and delimited by 1-junctions (if present in the data).
- *1-junctions* are sets of connected feature edges sharing the same set of incident subdomains. Each 1-junction forms a polyline at the intersection of 2-junctions with different subdomains pairs.
- *0-junctions* are corner vertices and are located at the intersection of 1-junctions.

We refer to boundary vertices that lie on 2-junctions but which do not belong to a 1- or 0-junction as *surface vertices*. The vertices that lie on 1-junctions and that are not corner vertices are referred to as *feature vertices*. Additionally, boundary edges linking two surface vertices are called *surface edges*. We do not need to build the feature complex since the elements of the feature complex are already present in the input mesh.

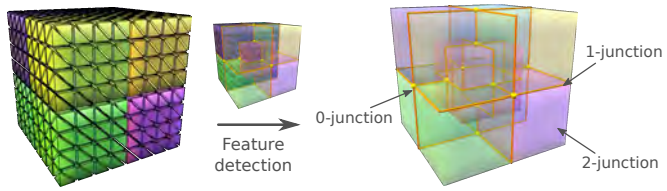


Figure 7: **Feature detection.** The surface patches represent 2-junctions, the polylines 1-junctions and the spheres 0-junctions.

3.5. Smooth Modeling

We model piecewise smooth boundaries by fitting 2-junctions with MLS surfaces. These implicit, meshless surfaces are defined through a local operator allowing to project a 3D point on a local surface reconstructed from unorganized point samples. We use a classical Point Set Surface (PSS) definition, proposed by Alexa et al. [45], to represent aliased boundaries and an Hermite Point Set Surfaces (HPSS) definition, proposed by Alexa and Adamson which avoids shrinking of the input model [4], otherwise. Any other definition can be used without any change to our methodology. For instance, in the case of input meshes with limited noise and relevant features, one might prefer a representation handling sharp features [46]. We refer to [47] for an excellent survey on MLS surfaces. We define the MLS surface associated with each 2-junction using a dense, area-based, consistently-oriented point sampling of it.

The MLS representation of the 2-junctions allows us to filter noise and acts as a regularizer at each step of our remeshing algorithm. Storing it also allows us to recover fine geometric details, when navigating from a low-resolution to a high-resolution version of the mesh during the remeshing session.

4. Operations

Special care needs to be taken when processing boundaries since the used local operations, described in this section, do not result in the preservation of the feature's topology. Therefore, following [36] and [37, 38], we define feature-aware rules and conditions depending on the feature complex hierarchy. Indeed, our rules assign a priority when processing the mesh elements based on their type which defines a so-called hierarchy.

4.1. Classification

The main idea is that only interior vertices should be allowed to be relocated freely in the volume, while n-junction vertices should be moved along their n-junction, in order to preserve the latter (the same idea was used in 2D for the tangential Laplacian [43], resulting in the preservation of the surface features). To fit these constraints, we define three different conditions between the pairs of connected vertices leading to different rules: *similarity*, *inclusion* and *exclusion* (see table 1).

- The *similarity* condition is met for v_i and v_j, that are not corner vertices, if $|S(v_i)| = |S(v_j)|$ and $|S(e_{ij})| = |S(v_i)|$ - i.e., for volume vertices, surface vertices linked by a surface edge, feature vertices linked by a surface edge and feature vertices linked by a feature edge.

- The *inclusion* condition if $|S(v_i)| > |S(v_j)|$ and $|S(e_{ij})| = |S(v_j)|$ and v_j is not a corner vertex - i.e., for mixed edges, edges linking a surface vertex and a feature or corner vertex and feature edges linking a feature and a corner vertex.

- Otherwise, the *exclusion* condition is met - i.e., critical edges, edges between feature or corner vertices not belonging to the same 1-junction or linking two 0-junctions.

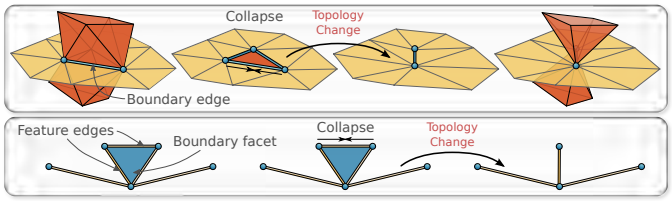


Figure 8: **Topology exceptions.** Additional tests to preserve the topology of the 1- and 2-junctions in small tubular regions.

The pair of vertices meeting the similarity condition affect each other. For the ones meeting the inclusion condition, only the vertex with the highest dimensionality in the feature complex, or the one not belonging to it, will be affected. And finally, for the exclusion condition, none of the vertices will be affected since it would create undesired merging of vertices that belong to different 2- or 1-junctions and fail to preserve small local features of the subdomains. Table 1 summarizes the conditions and the derived rules. In practice, except for corner vertices, a vertex v_i will only be affected by the vertices v_j of its one-ring if $|S(v_i)| \geq |S(v_j)|$ and $|S(e_{ij})| = |S(v_j)|$.

Unfortunately, these conditions do not prevent a change of topology in the same tubular regions as illustrated in Fig. 8. The three edges around the red facet meet the similarity condition but a collapse operation would change the topology of the subdomains, creating a pinched boundary. Similarly, the collapse of any of the three feature edges around the blue facet would change the topology of the 1-junction. Therefore, we perform two additional *topology tests* for boundary and feature edges to detect these configurations and prevent the collapse in these cases. A boundary edge is not collapsed if one of its incident facets is not a boundary one but is composed of three boundary edges. Similarly, a feature edge is not collapsed if one of its incident facets is composed of three feature edges. Now that we have defined the feature-aware rules, we describe our remeshing process in the following.

4.2. Split edges

In the first step of our iterative procedure, all the edges with a length superior to $e_{\max}$ are split, i.e., a vertex v_k is added at their mid-point, dividing every tetrahedron around the edge into two. Note that the two new tetrahedra are assigned with the same label as the one they are subdividing. The set of labels of the added vertex v_k is the set of labels of the tetrahedra around the current edge, i.e., $S(v_k) = S(e_{ij})$. As only insertions are performed at this step, no feature preservation rule is required.

4.3. Collapse edges

Now that the long edges have been split, we remove short edges in order to get a uniform edge length close to the target one. To do so, we collect all the edges to collapse with a length smaller than $e_{\min}$, and proceed to perform the possible collapses. At this point, our thorough classification becomes instrumental, as not all edges can be collapsed without modifying the topology of the subdomains, inverting tetrahedra or even resulting in an invalid data structure [29, 48].

Condition	Type of v_i	Type of v_j	Type of e_{ij}	$\lvert S(v_i)\rvert$	$\lvert S(v_j)\rvert$	$\lvert S(e_{ij})\rvert$	# f_i	# f_j	Update v_i	Update v_j
Similarity	volume	volume	volume	1	1	1	n/a	n/a	yes	yes
Similarity	surface	surface	surface	2	2	2	-	-	yes	yes
Similarity	feature	feature	feature	$n_i > 2$	$= n_i$	$= n_i$	< 3	< 3	yes	yes
Inclusion	volume	boundary	mixed	1	> 1	1	-	-	yes	no
Inclusion	surface	feature or corner	surface	2	> 2	2	-	-	yes	no
Inclusion	feature	corner	feature	$n_i > 2$	$n_j > n_i$	n_i	< 3	-	yes	no
Exclusion	boundary	boundary	critical	n_i	n_j	$\neq min(n_i, n_j)$	-	-	no	no
Exclusion	feature	feature or corner	surface	-	-	-	> 2	-	no	no
Exclusion	corner	corner	-	-	-	-	-	> 2	no	no

Table 1: **Conditions and rules ensuring the preservation of the subdomains' topology.** All operations depend on the type of the edge and its vertices. We can easily detect if the vertices will be affected during the current operation depending on their number of incident subdomains and of incident feature edges noted $\#f_i$, i.e., on their dimension in the feature complex.

Feature preserving collapse. We start by evaluating which condition the current edge meets in order to define which type of collapse to perform, according to table 1:

- *similarity*: mid-point collapse (both vertices are affected),
- *inclusion*: toward the vertex with the highest number of subdomains, i.e., with the lowest dimension in the feature complex,
- *exclusion*: no collapse, since a contraction would change the topology of the subdomains.

The tetrahedra incident to the affected vertex, or vertices, are set to be updated, except the ones incident to the current edge, which are set to be removed. In order to ensure the validity of the operation, we perform the following tests:

1. all the updated tetrahedra have a positive volume,
2. all the new edges are shorter than e_{max},
3. no incident non boundary facet has three boundary edges,
4. no incident boundary facet has three feature edges.

The third (resp. fourth) topology test is performed for boundary (resp. feature) edges. The operation is performed if these four constraints are met. Some of the new edges, incident to the remaining vertex, may be shorter than e_{min}. Therefore, we evaluate their length and set the ones that do not respect the length criteria to be collapsed. This process is repeated until all edges are either smaller than e_{min} or impossible to collapse.

Following Botsch and Kobbelt [41], once the overall edge length is close enough to the target one, the local connectivity is changed in order to minimize the average valence and optimize locally the dihedral angle distribution, using an edge flip operation described in the following.

4.4. Flip edges

Flipping an edge in a tetrahedral mesh induces far more changes than for a triangular mesh. Indeed, the operation changes the number of tetrahedra adjacent to the edge, except when there are exactly two or four adjacent tetrahedra. It removes an edge and replaces it with facets. The flip operation, also called *edge removal*, has been first proposed in [49] and further studied in [25, 26], as a mesh improvement strategy. For each edge e_{ij}, we explore the space of possible flip operations and perform the one that offers the best quality for $T_1(e_{ij})$. Specifically, it should maximize the worst dihedral angle for volume edges and average the boundary vertices valences, i. e. minimize the average boundary vertices valence's deviation from 6 for surface vertices and 4 for feature vertices. In the mean time, the operation generating inverted tetrahedra or edges that already exist are discarded. Recalling that boundary edges have exactly two incident boundary facets, we preserve the 2-junctions by flipping the edges towards one of the two boundary vertices of these facets, enforcing a boundary edge in the new configuration. Last, feature edges are not flipped since processing them would fail to preserve 1-junctions.

4.5. Filtering

Once the connectivity has been updated to locally optimize the dihedral angle distribution, we relocate the vertices to improve their distribution (see Fig. 9). Each vertex is relocated by averaging the subset of its one-ring vertices that meet the similarity or inclusion condition (according to table 1) and, for boundary vertices, that verify the topology tests. To prevent the inversion of tetrahedra, first we smooth the feature vertices, then the surface ones and finally the volume vertices.

Feature vertices. The feature vertices lie at the intersection of three or more 2-junctions. For each adjacent 2-junction, we perform a tangential Laplacian smoothing using the positions

Figure 9: **Feature preserving smoothing.** Using the feature complex, a feature preserving hierarchical smoothing is performed.

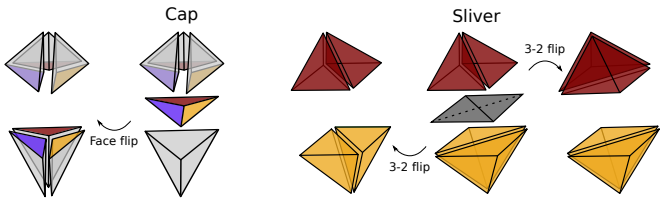


Figure 10: **Sliver removal.** Optimization step to remove the remaining few poorly-shaped tetrahedra: (Left) face flip, for cap tetrahedra, or (Right) edge removal for other kinds of degeneracy.

of its neighbors on the related 1-junction and the vertex normal corresponding to the current 2-junction, then project the result on its MLS surface. The smoothed position is then obtained by averaging the projected points.

Surface vertices. We perform a tangential Laplacian smoothing by considering (i) the surface, updated feature and corner vertices of its one ring, that verify the *similarity* or *inclusion* condition and the topology tests, and (ii) the corresponding vertex normals, which are computed using its adjacent boundary facets. The smoothed position is then projected onto the MLS surface modeling the related 2-junction.

Volume vertices. Finally, the vertices that do not belong to the feature complex are smoothed by performing a Laplacian smoothing using all its adjacent vertices, since they all verify the *similarity* and *inclusion* conditions.

4.6. Quality improvement

After a few iterations (typically 5 to 10), the target resolution is usually reached and the overall mesh quality is improved. However, we observed that performing a few cycles of smooth and flip operations drastically improves the final quality.

A final optimization process might be necessary to remove slivers or poorly-shaped tetrahedra. For each tetrahedron not meeting the quality criteria, we identify its type of degeneracy (see Fig. 2) and we perform the operation that improves the local quality while respecting the previously prescribed rules.

First, *needles* or *wedges*, which have a large longest to shortest edge ratio in addition to a small radius-ratio, are removed by collapsing their first collapsable shorter edge improving the local quality. Note that, since our method equalizes the edge length, these types of elements are unlikely to appear. Then the *caps*, which have a small radius-ratio and three large dihedral angles, are removed by flipping the face opposite to the three largest dihedral angles. Finally, for *slivers*, which have two large and two small dihedral angles, we perform the flip of one of the two edges, shared by the pairs of facets forming the larger dihedral angles, that best improves the local quality (see Fig. 10). Usually, a small number of tetrahedra do not meet the quality criteria, and only a few iterations are required.

4.7. Adaptive sizing field

Our algorithm supports spatially-varying edge length targets. We present in this section results based on sizing fields

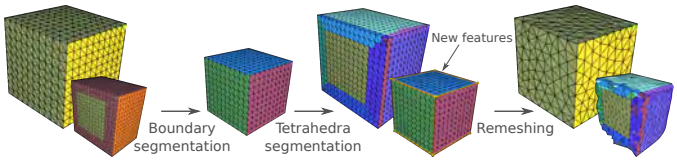


Figure 11: Closed surface features are detected on the outer boundary of the represented domain, and are added to the feature complex by segmenting the imaginary tetrahedra.

that we derive automatically from the input geometry. In particular, in order to get graded meshes of reduced size, we propose to compute the sizing field based on a distance field from the boundaries. This distance field is computed by first discretizing the space inside the triangulation's offseted bounding box, resulting in a voxel grid of a user-defined resolution. In a second step, each voxel is assigned the smallest distance from its center to the set of boundary facets of the input model. These distances are efficiently computed using an acceleration structure detailed in section 5. Finally, the grid values are normalized. Given a target length for the boundary edges l_B and one for the volume edges l_V, the target length l_{ij} for the current edge e_{ij} is given by:

$$l_{ij} = (l_V - l_B)D(m_{ij}) + l_B$$

with m_{ij} the edge mid-point and $D : \mathbb{R}^3 \rightarrow \mathbb{R}^+$ the normalized distance field from the boundaries. Note that any user-defined sizing field can be used to tailor the edge length and produce dense meshes in regions of interests.

4.8. Additional feature preservation

The preserved 1- and 0- features stem from the multi-material junctions. Nevertheless, the input domain's sharp features are not taken into account. In the following, we explain how to preserve additional features by either segmenting the imaginary tetrahedra, hence creating multi-material junctions, or by explicitly tagging features edges and corner vertices to be preserved during the remeshing process.

Closed features. We propose to detect closed features on the outer boundary of the represented domain, and to add them to the feature complex by segmenting the imaginary tetrahedra. To do so, the outer boundary facets - with one incident imaginary tetrahedron - are clustered in surface patches with similar normals delimited by closed feature lines using the Variational Shape Approximation (VSA) method [50].

Detected feature lines represent sharp creases of the input domain. In order to preserve them during the remeshing process, we add them to the feature complex by propagating the facet's segmentation to the imaginary tetrahedra. The imaginary tetrahedra are therefore labeled implicitly, creating 1-junctions where the detected close features lie (see Fig. 11 for an illustraton). In concave regions, facets belonging to two different regions might be connected by critical edges (see Fig. 12). In that case, performing a straightforward flood-filling, propagating the facet segmentation to the imaginary tetrahedra, induces unconnected components, conflict regions and sparse 1- and 0-junctions created on the boundaries (see Fig. 12). To overcome this problem, we split imaginary critical edges as a pre-process.

Figure 12: To overcome conflicts that can occur in concave regions, the imaginary critical edges are split as a pre-process.

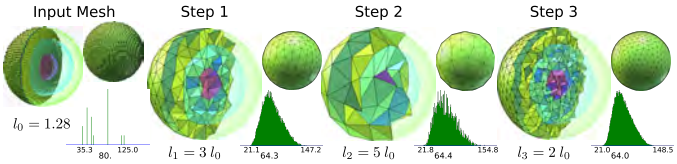


Figure 13: **Parameter space exploration.** High-quality meshes generated for each target length, in 5 iterations plus 2-3 flip and smooth cycles, with dihedral angles between [21.1, 147.2], [21.8, 154.8] and [21.0, 148.5] for step 1, 2 and 3 respectively.

Marked features. The described feature detection and preservation method is limited to closed junctions. However, not all relevant feature lines (e.g., surface curvature-based ones) are closed. Hence, we propose to preserve user-prescribed polylines by adding (i) each polyline as a distinct 1-junction to the feature complex and (ii) their end points and intersections as corner vertices. The features are explicitly tagged and not implicitly defined by the multi-material junctions. Hence when splitting a feature edge, the two new edges and the inserted vertex are tagged as features. Consequently, our conditions definitions (see Table 1) still hold. Table 2 summarizes the specific rules for this case: if the user wants to preserve the exact input feature vertices, we set the feature polylines being smoothed by the remeshing process as corner vertices.

Condition	# f_i	# f_j	Is e_{ij} feat.	Update v_i	Update v_j
Similarity	2	2	yes	yes	yes
Inclusion	0	>= 1	no	yes	no
Inclusion	2	1 or > 2	yes	yes	no
Exclusion	> 0	> 0	no	no	no
Exclusion	1 or > 2	1 or > 2	-	no	no

Table 2: Feature-aware rules for tagged features.

5. Results and comparisons

Performances were measured on an Intel Core2 Duo (single thread) at 2.4 GHz with 8GB of main memory. We used the Computational Geometry Algorithms Library (CGAL) 3D triangulation code as an underlying mesh structure and its Axis-Aligned Bounding Box (AABB) tree implementation to efficiently compute the adaptive sizing field. This tree is built using the input mesh's boundary facets and is used as an acceleration structure to compute the distance field. Since the conditions are evaluated using local adjacency information, we do not need to build the feature complex explicitly in contrast to [3]. We only store the list of labels incident to each vertex and construct the MLS representation of the 2-junctions. Note that we tagged the elements of the user-provided features. Our implementation is robust to poorly-shaped tetrahedra with zero or negative volume. Furthermore, any type of triangulation can be remeshed by our method, allowing us to process a wide range of input. We demonstrate the validity of our approach on both synthetic and acquired segmented voxel grids. We apply our method on naive meshes generated from a segmented 3D voxel grid defined through a five-cells decomposition of all the voxels contained in the bounding box of the domain while unifying internal and external boundaries. The resulting high-resolution mesh reproduces the grid topology with aliased boundaries. As explained in the previous section, our rules depend of the element's dimension in the feature complex. For instance, we start by flipping the boundary edges that improve the average valence of the boundary vertices and then flip the volume edge that locally maximizes the minimal dihedral angle. The MLS representation of the surface patches allows smoothing noisy boundaries and features. Furthermore, it allows refining as well as simplifying the input mesh while preserving the boundaries' shape, since the representation is independent from the current mesh state. To evaluate the quality of the resulting meshes, we report the dihedral angle distributions using green histograms and the distributions of radius-ratio (multiplied by 3 for normalization) using orange ones. Fig. 13 illustrates an interaction session where the user navigates between different resolutions. High-quality meshes are generated, within seconds, for three different target lengths, in 5 iterations plus 2-3 flip and smooth cycles. Even though no sliver removal step was performed, the resulting meshes all have dihedral angles above 21 degrees.

Our approach is targeted but not limited to multi-material input domains, and we evaluate our approach on single material input meshes as well (see Fig. 1, 14, 15 and 16). We can observe the good angle distribution and assess visually the overall mesh quality. We compare our method with state-of-the-art single-material meshing techniques. In Fig. 15, the meshes of the first row are generated from the same Sphere surface mesh using the same size and quality parameters. The first mesh is generated by a Delaunay Refinement process and the second by using DelPSC [11]. The meshes of the second row are generated using the Refinement mesh as an input. We performed our remeshing process, using 5 iterations and 3 flip-smooth cycles, in 20 seconds and performed an ODT [51, 52] optimization with the same time limit. We can see that our method does not produce slivers since the smallest dihedral angle is 23.2 degrees, whereas other methods need an additional optimization

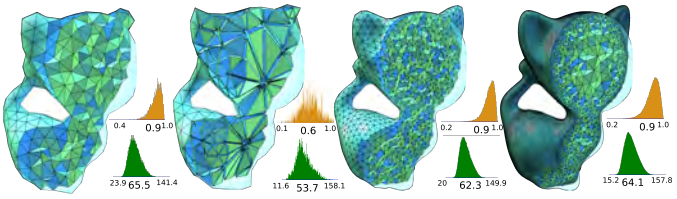


Figure 14: **Kitten.** High-quality isotropic and adaptive remeshing.

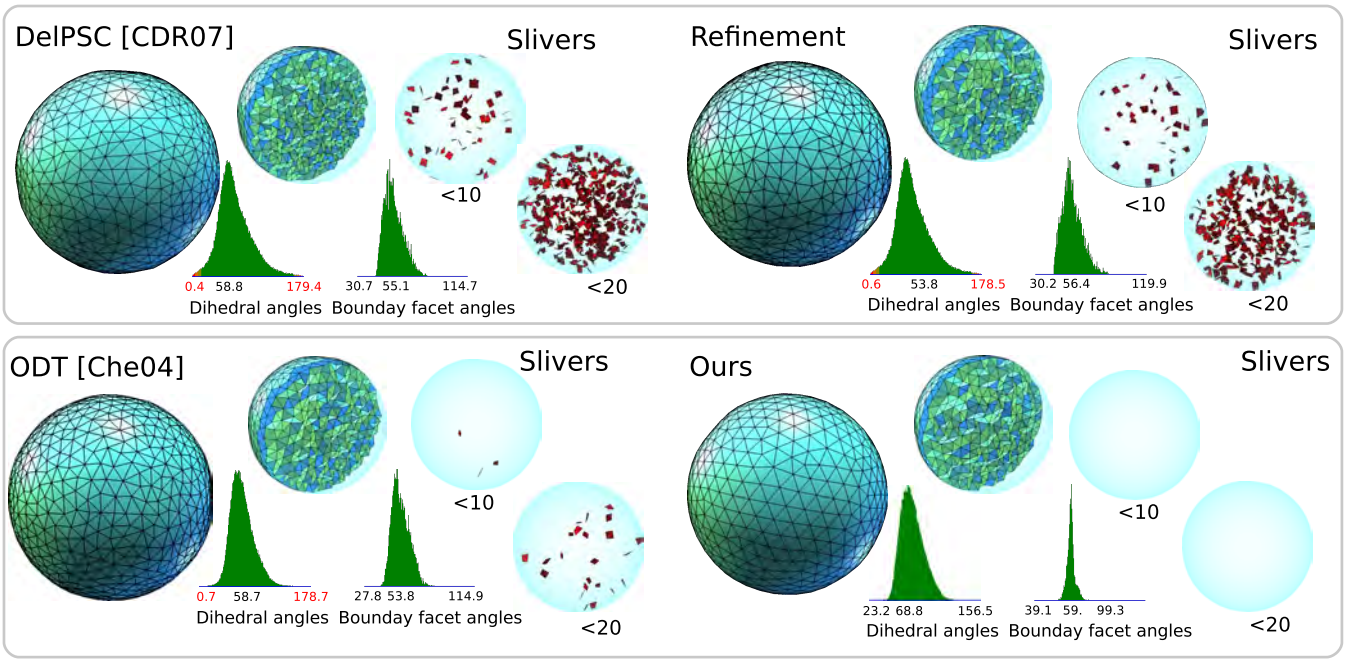


Figure 15: **Comparison.** First row: Mesh generated from the same Sphere surface mesh using the same size and quality parameters. The first mesh is generated by using DelPSC [11] and the second by a Delaunay Refinement process. Second row: Meshes generated using the Refinement mesh as an input.

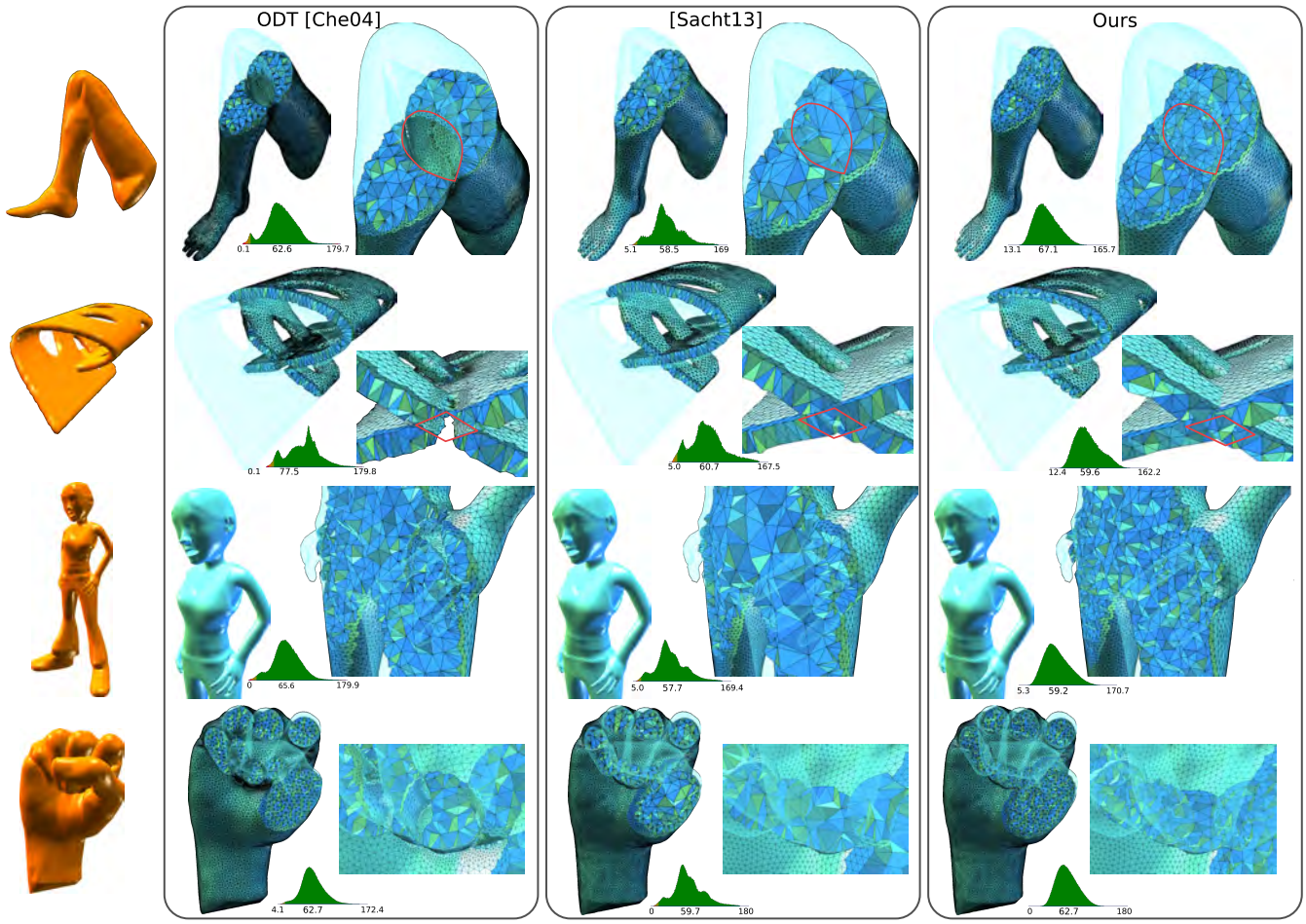


Figure 16: **Self intersecting mesh processing.** Note that Delaunay-based techniques, such as ODT can not process self-intersecting surfaces. The green histograms show the dihedral angle distributions.

Model	Input. #tet	$s_B - s_V$	Output #tet	In complex #tet.	Timing
Spheres	3 999	1	20 062	6 165	8s
		1-5	23 588	2 000	10s
		0.6-5	44 784	10 669	31s
4 labels sphere	750 000	2	85 853	30 834	1m24s
		6	3 192	1 153	6s
		4-5	5 470	2 074	20s
Hepatic Vessel	4 757 904	1.1-8	679 580	187 616	20m
		2.5-8	61 170	16 546	3m20s
Hand	4 158 720	2.4 - 8	3 717 750	88 167	8m38s
		1.1 - 8	617 920	219 739	7m30s
Kitten	371 183	7.6	9 439	3 393	57s
		7.6-8	6 291	856	2s
		3.6	299 669	31 322	53s
		1.1-13	784 717	244 424	14m08s

Table 3: **Performances.** The results are obtained by setting two scalars values s_B and s_V for adaptive meshes, or a single scalar $s_B = s_V$ for isotropic meshes, defining the target edge lengths as $l_B = s_B * l_0$ for the boundaries and $l_V = s_v * l_B$ for the volume, l_0 denoting the average input boundary edge length. Output #tet is the number of tetrahedra in the triangulation and in complex and #tet displays the number of tetrahedra that belong to the input domain, i. e. that are not imaginary. Note that further increased performance is to be expected, if one desires to ignore imaginary tetrahedra in the mesh.

step. Note that on a similar example, NODT [13] performs better than ODT but also produces slivers. Our method prevents the apparition of degenerated tetrahedra with a large longest to shortest edge ratio, such as needles and wedges. Since the edge lengths are equalized, most of the slivers and spindles are removed on account of the angle-based flip and the remaining few are taken out as a final process, along with the cap tetrahedra (see Fig. 2). Note that this final step is often unnecessary.

Since only local topological operators are used, our method allows processing self-intersecting volumes meshes. In Fig. 16, we show (i) on the left results of a Delaunay-based meshing method of a self-intersecting input domain creating holes in the output mesh, (ii) in the middle the tetrahedral meshes generated by the method proposed by Sacht et al. [53] and (iii) on the right our remeshing resulting using the later meshes as an input. Note that the output of Sacht et al. [53] (Fig. 16) have been optimized in *a different pose*: they smooth the input surface until intersections are resolved, triangulate the resulting smoothed surface and only then put the triangulation in a geometric state that is compatible with the input geometry of the surface. On the contrary, we optimize the mesh in its input pose directly.

Last, we present additional synthetic results (see Fig. 17) and apply our method on segmented medical images (see Fig. 1 and 17). Note that the extremely low dihedral angles (bottom left mesh in Fig. 17) correspond to isolated tetrahedra or small features w.r.t. the target edge length in the input. Our method optimizes the geometry under the hard constraint of exact feature preservation. Table 3 shows the performances for the remeshing of the presented various input data sets. Note that the performances strongly depend on the input and the edge aimed length. Timings are obtained using the mesh resulting from the previous remeshing session as an input, in order to emphasize the dependance to the *current state* instead of the *input state*. We made sure to present sessions where the resolution was changed smoothly and others where the resolution was changed arbitrarily, to balance illustration and fairness of comparison with existing previous work.

6. Limitations & Future Work

Our feature-aware operations strictly preserve the input topology, and this property may cause problems for noisy datasets. Indeed, poorly-shaped tetrahedra are generated when the target edge length is significantly larger than the size of the feature to preserve. To tackle this issue, part of these feature-preserving conditions could be relaxed or the target edge length could be changed during a post-processing stage. Additionally, when starting from a segmented voxel grid, a pre-processing step can be performed to remove isolated voxels and small features as well as merge corner vertices being too close [3]. More generally, groups of poorly-shaped tetrahedra could be removed using a feature preserving version of the vertex insertion strategy proposed in a single material setting [26]. These processes can be combined with a roll back mechanism in order to cancel the operations that did not improve the quality. Since we use local topological operations only, our remeshing method can be applied on a portion of the mesh using only local neighborhood information. Therefore, based on the streaming algorithm for compressing tetrahedral volume meshes proposed [28], an extension of our method to model data that do not fit in main memory seems foreseeable.

7. Conclusion

We have proposed a new efficient multi-domain adaptive remesher for tetrahedral meshes. Our approach is based on local operations which simultaneously refine or simplifying the tetrahedra while improving the quality through local topology changes and point relocations. Additionally, our framework allows to preserve features detected on the outer boundary of the domain as well as user-defined features. By decorrelating the piecewise smooth boundary model from the mesh resolution using an MLS approach, our method provides high-quality meshes at different resolutions using well known simple local topological operators and is general enough to be applied to any structured mesh. As a result, our high-quality adaptive remesher can compete with state-of-the-art methods [3] but is free from the computation of a Delaunay triangulation/Voronoi diagram of multi-material domains, requires only to build a PSC instead, has lower memory cost, can process self-intersecting meshes, and is significantly easier to implement.

[1] Shewchuk JR. What is a good linear element? - interpolation, conditioning, and quality measures. In: 11th International Meshing Roundtable. 2002, p. 115–26.

[2] Boltcheva D, Yvinec M, Boissonnat JD. Feature preserving Delaunay mesh generation from 3d multi-material images. In: Symposium on Geometry Processing. 2009, p. 1455–64.

[3] Dey TK, Janoos F, Levine JA. Meshing interfaces of multi-label data with Delaunay refinement. Engineering with Computers 2012;28:71–82.

[4] Alexa M, Adamson A. Interpolatory point set surfaces-convexity and hermite data. ACM Transactions on Graphics (TOG) 2009;28:20:1–20:10.

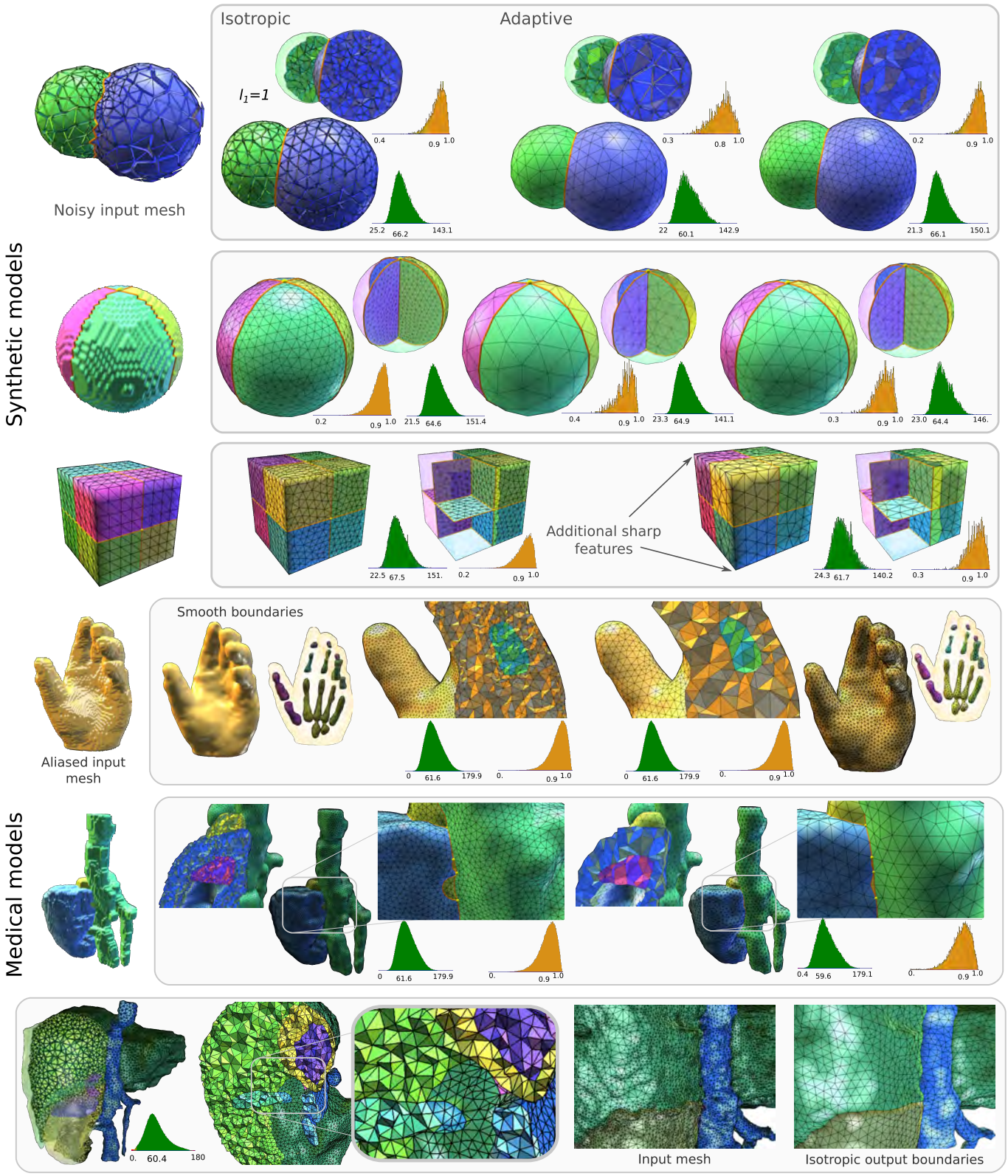


Figure 17: **Remeshing synthetic and medical data at different resolutions.** The last row of the synthetic data illustrates the addition of closed features to the feature complex which are detected on the mesh outer boundary. The green histograms show the dihedral angle distributions and the orange ones display the radius-ratio distributions.

[5] Chew LP. Guaranteed-quality mesh generation for curved surfaces. In: Symp. on Computational geometry. 1993, p. 274–80.

[6] Ruppert J. A delaunay refinement algorithm for quality 2-dimensional mesh generation. J Algorithms 1995;18:548–85.

[7] Shewchuk JR. Tetrahedral mesh generation by Delaunay refinement. In: Fourteenth annual symposium on Computational geometry. 1998, p. 86–95.

[8] Boissonnat JD, Oudot S. Provably good sampling and meshing of surfaces. Graphical Models 2005;67:405 –51.

[9] Cheng SW, Dey TK. Quality meshing with weighted Delaunay refinement. SIAM Journal on Computing 2003;33:69–93.

[10] Cheng SW, Dey TK, Levine A. A practical Delaunay meshing algorithm for a large class of domains. In: Proceedings of the 16th International Meshing Roundtable. 2007, p. 477–94.

[11] Cheng Sw, Dey TK, Ramos EA. Delaunay refinement for piecewise smooth complexes. In: 18th Annual ACM-SIAM Symposium Discrete Algorithms. 2007, p. 1096–105.

[12] Dey T, Levine J. Delaunay meshing of piecewise smooth complexes without expensive predicates. Algorithms 2009;2:1327–49.

[13] Tournois J, Wormser C, Alliez P, Desbrun M. Interleaving delaunay refinement and optimization for practical isotropic tetrahedron mesh generation. ACM Transactions on Graphics (TOG) 2009;28(3):Art–No.

[14] Pons JP, Ségonne F, Boissonnat JD, Rineau L, Yvinec M, Keriven R. High-quality consistent meshing of multi-label datasets. In: 20th international conference on Information processing in medical imaging. 2007, p. 198–210.

[15] Lorensen WE, Cline HE. Marching cubes: A high resolution 3d surface construction algorithm. ACM Siggraph Computer Graphics 1987;21(4).

[16] Wu Z, Sullivan JM. Multiple material marching cubes algorithm. International Journal for Numerical Methods in Engineering 2003;58:189–207.

[17] Labelle F, Shewchuk JR. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. ACM Transactions on Graphics (TOG) 2007;26.

[18] Bronson J, Levine J, Whitaker R. Lattice cleaving: Conforming tetrahedral meshes of multimaterial domains with bounded quality. In: Jiao X, Weill JC, editors. 21st International Meshing Roundtable. 2013, p. 191–209.

[19] Crossno P, Angel E. Isosurface extraction using particle systems. In: IEEE Visualization. 1997, p. 495–8.

[20] Meyer M, Whitaker R, Kirby R, Ledergerber C, Pfister H. Particle-based sampling and meshing of surfaces in multimaterial volumes. Visualization and Computer Graphics, IEEE Transactions on 2008;14:1539 –46.

[21] Dardenne J, Valette S, Siauve N, Burais N, Prost R. Variational tetrahedral mesh generation from discrete volume data. The Visual Computer 2009;25:401–10.

[22] Goksel O, Salcudean S. Image-based variational meshing. Medical Imaging, IEEE Transactions on 2011;30:11 –21.

[23] Cheng SW, Dey TK, Edelsbrunner H, Facello MA, Teng SH. Silver exudation. J ACM 2000;47:883–904.

[24] Tournois J, Srinivasan R, Alliez P. Perturbing Slivers in 3D Delaunay Meshes. In: 18th International Meshing Roundtable. 2009, p. 157–73.

[25] Freitag LA, Ollivier-gooch C. Tetrahedral mesh improvement using swapping and smoothing. International Journal for Numerical Mthods In Engineering 1997;40:3979–4002.

[26] Klingner BM, Shewchuk JR. Agressive tetrahedral mesh improvement. In: Proceedings of the 16th International Meshing Roundtable. 2007, p. 3–23.

[27] Garland M, Zhou Y. Quadric-based simplification in any dimension. ACM Transactions on Graphics (TOG) 2005;24:209–39.

[28] Isenburg M, Lindstrom P, Gumhold S, Shewchuk J. Streaming compression of tetrahedral volume meshes. In: Proceedings of Graphics Interface 2006. Canadian Information Processing Society; 2006, p. 115–21.

[29] Hoppe H. Progressive meshes. In: 23rd annual conference on Computer graphics and interactive techniques. 1996, p. 99–108.

[30] Garland M, Heckbert PS. Surface simplification using quadric error metrics. In: 24th annual conference on Computer graphics and interactive techniques. 1997, p. 209–16.

[31] Uesu D, Bavoil L, Fleishman S, Shepherd J, Silva CT. Simplification of unstructured tetrahedral meshes by point sampling. 2005.

[32] Farias R, Mitchell J, Silva C, Wylie B. Time-critical rendering of irregular grids. In: Computer Graphics and Image Processing. 2000, p. 243 –50.

[33] Alauzet F, Li X, Seol ES, Shephard MS. Parallel anisotropic 3d mesh adaptation by mesh modification. Engineering with Computers 2006;21(3):247–58.

[34] Loseille A, Löhner R. Robust boundary layer mesh generation. In: Proceedings of the 21st International Meshing Roundtable. Springer; 2013, p. 493–511.

[35] Dapogny C, Dobrzynski C, Frey P. Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems. Journal of Computational Physics 2014;262:358–78.

[36] Cutler B, Dorsey J, McMillan L. Simplification and improvement of tetrahedral models for simulation. In: Proc. Symposium on Geometry Processing. 2004, p. 93–102.

[37] Thomas DM, Natarajan V, Bonneau GP. Link Conditions for Simplifying Meshes with Embedded Structures. Transactions on Visualization and Computer Graphics 2010;:1007–19.

[38] Vivodtzev F, Bonneau GP, Hahmann S, Hagen H. Substructure topology preserving simplification of tetrahedral meshes. In: Topological Methods in Data Analysis and Visualization. 2011, p. 55–66.

[39] Kobbelt L, Bareuther T, peter Seidel H. Multiresolution shape deformations for meshes with dynamic vertex connectivity. 2000.

[40] Vorsatz J, Rössl C, peter Seidel H. Dynamic remeshing and applications. In: Department, Stony Brook University. Her. 2003, p. 167–75.

[41] Botsch M, Kobbelt L. A remeshing approach to multiresolution modeling. In: Symposium on Geometry Processing. 2004, p. 185–92.

[42] Kraus M, Ertl T. Simplification of nonconvex tetrahedral meshes. In: Hierarchical and Geometrical Methods in Scientific Visualization. 2002, p. 185–96.

[43] Botsch M. High quality surface generation and efficient multiresolution editing based on triangle meshes. Ph.D. thesis; 2005.

[44] Adamson A, Alexa M. Point-sampled cell complexes. In: ACM Transactions on Graphics (TOG); vol. 25. 2006, p. 671–80.

[45] Adamson A, Alexa M. Approximating bounded, non-orientable surfaces from points. In: Shape Modeling International. 2004, p. 243–52.

[46] Oztireli C, Guennebaud G, Gross M. Feature Preserving Point Set Surfaces based on Non-Linear Kernel Regression. Computer Graphics Forum 2009;28:493–501.

[47] Cheng ZQ, Wang YZ, Li B, Xu K, Dang G, Jin SY. A survey of methods for moving least squares surfaces. In: Point-Based Graphics. 2008, p. 9–23.

[48] Dey T, Edelsbrunner H, Guha S, Nekhayev D. Topology preserving edge contraction. Publ Inst Math(Beograd)(NS) 1999;66:23–45.

[49] de l'Isle EB, George. PL. Optimization of tetrahedral meshes. In: Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations, IMA Volumes in Mathematics and its Applications. 1995, p. 97–128.

[50] Cohen-Steiner D, Alliez P, Desbrun M. Variational shape approximation. ACM Transactions on Graphics (TOG) 2004;23:905–14.

[51] Chen L. Mesh smoothing schemes based on optimal Delaunay triangulations. In: 13th International Meshing Roundtable. 2004, p. 109–20.

[52] Alliez P, Cohen-Steiner D, Yvinec M, Desbrun M. Variational tetrahedral meshing. ACM Transactions on Graphics (TOG) 2005;:617–25.

[53] Sacht L, Jacobson A, Panozzo D, Schüller C, Sorkine-Hornung O. Consistent volumetric discretizations inside self-intersecting surfaces. Computer Graphics Forum (proceedings of EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing) 2013;32(5):147–56.

Eurographics Symposium on Rendering 2016
E. Eisemann and E. Fiume
(Guest Editors)

Volume 35 (2016), Number 4

Forward Light Cuts: A Scalable Approach to Real-Time Global Illumination

Gilles LAURENT[+,*] Cyril DELALANDRE[+] Grégoire de LA RIVIÈRE[+] Tamy BOUBEKEUR[*]

[+] Dassault Systèmes [*] LTCI, CNRS, Telecom ParisTech, Université Paris-Saclay

Figure 1: Real-time rendering with the first bounce of indirect lighting using Forward Light Cuts. The scene is composed of 7M triangles and is rendered in 16 ms at 1024×512 pixels resolution, without any precomputation and accounting for occluded yet contributing surfaces and long range light bounces.

Abstract

We present Forward Light Cuts, a novel approach to real-time global illumination using forward rendering techniques. We focus on unshadowed diffuse interactions for the first indirect light bounce in the context of large models such as the complex scenes usually encountered in CAD application scenarios. Our approach efficiently generates and uses a multiscale radiance cache by exploiting the geometry-specific stages of the graphics pipeline, namely the tessellator unit and the geometry shader. To do so, we assimilate virtual point lights to the scene's triangles and design a stochastic decimation process chained with a partitioning strategy that accounts for both close-by strong light reflections, and distant regions from which numerous virtual point lights collectively contribute strongly to the end pixel. Our probabilistic solution is supported by a mathematical analysis and a number of experiments covering a wide range of application scenarios. As a result, our algorithm requires no precomputation of any kind, is compatible with dynamic view points, lighting condition, geometry and materials, and scales to tens of millions of polygons on current graphics hardware.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Radiosity

1. Introduction

Light transport simulation is an important component of realistic image synthesis. Beyond direct lighting, *global illumination*, despite its well known physics laws, remains a challenging problem due to its high computational cost, with even more critical consequences for fully dynamic real-time scenarios involving large objects. Hidden behind the recursive nature of the rendering equation [Kaj86], global illumination simulation has been addressed in a number of approaches with applications ranging from visual special effects to scientific visualization, through animated pictures and video games. Currently, we can distinguish two lines of research: *offline rendering*, which targets a solution as close as possible to physics and *interactive rendering*, which aims at quickly providing a visually convincing approximation of global illumination. While significant progress have been recently made for the former using the Monte Carlo rendering framework, we focus on the latter and the set of constraints induced by real-time scenarios.

Due to the low-pass filtering nature of diffuse material reflection [RH01, BJ03], most real-time global illumination methods exploit the reasonable assumption that indirect radiance can be described by a low frequency function, especially when the emitter is far from the receiver (Sec. 2). In particular, numerous *radiance caching* methods [WFA*05, REG*09] compute a hierarchical spatial structure over the geometry of the scene and use it to model a multiscale radiance function, later queried to illuminate the pixels of the final image. The leaves of this tree structure are typically

formed by so-called *virtual point lights* (or VPLs), which are sampled on the scene surfaces that are visibile from the primary light emitters at caching time. Every internal node of this tree is then set with a representative response that approximates the radiance of its related subtree. At shading time, a set of nodes, called *light cut*, is adaptively gathered from the tree to evaluate the incoming radiance at a given point. Although it may contain leaves (i.e., original VPLs) for close-by elements, it is typically mostly formed of internal nodes that act as economic substitutes to represent the incoming radiance from distant locations, saving both time and memory.

For this category of methods, a fully dynamic scenario, involving large objects, induces at least two limitations. First, the caching structure needs to be recomputed at each frame, as detecting changes under fully dynamic conditions ends up being just as costly as recomputing the whole cache, in particular regarding the limitation of the fine-grained parallel nature of modern graphics hardware. Second, the initial set of VPLs may be too large to cope with real time constraints, in particular when their generation cannot be amortized over time.

In this paper, we adopt a forward strategy to address these problems (Sec. 3). First, we observe that the scene polygons themselves can trigger the VPL generation process and propose a tessellation/decimation GPU pipeline that uses both the geometry shader and the tesselator unit to generate an initial set of VPLs (Sec. 4), refining the geometry of the scene wherever it is too coarse to accurately capture the radiance, and simplifying it where the polygon distribution is too dense. Second, we propose a stochastic clustering scheme that associates subsets of the resulting VPLs to bounded regions of influence for which they act as radiance representatives to shade points. This yields a multiscale representation of indirect lighting free from any explicit tree structure used to efficiently shade receivers (Sec. 5). Moreover, we designed both the caching and shading stages to efficiently map on modern graphics architectures (Sec. 6). As a result, our entire algorithm runs from scratch at every frame and preserves real-time performance even for large scenes, capturing diffuse unshadowed indirect illumination under dynamic conditions, while naturally accounting for long range indirect illumination and hidden geometry (Sec. 7).

2. Previous Work

A full survey of real-time global illumination methods is beyond the scope of this paper and we refer the reader to the recent manuscripts by Ritschel et al. [RDGK12] for an up-to-date overview of real-time global illumination methods and Dachsbacher et al. [DKH*14] for a complete overview of the many-lights framework. As we target diffuse GI for large dynamic scenes, we focus on the most relevant prior art in the following, namely *screen-space* and *object-space* diffuse global illumination solutions.

Mittring et al. [Mit07] introduced an ambient occlusion approximation method using the depth-buffer as an economic, random-accessible substitute to the actual (potentially large) geometry of the scene, and parameterizing the light cache in screen-space. Later, Ritschel et al. [RGS09] extended this approach to simulate diffuse color bleeding in a similar setting. A large variety of other methods [RDGK12] exploit screen-space approximations to lower the computational complexity of some lighting effects. However, despite their real-time and dynamic performances, such approaches rely on depth peeling and multiple views rendering to account for the full geometry of the scene i.e., beyond the first depth layer and outside the view frustum, which quickly impacts negatively their native speed. Recently, Mara et al. [MMNL14] proposed to take advantage of temporal coherency to build a multi-layered sampling strategy and remove most of hidden surface issues – *e.g.* view dependent ghosting artifacts. This approach is effective in a number of cases, but still suffers from grazing angle geometry undersampling issues.

In contrast to screen-space approaches, solving for indirect illumination in object-space avoids such view-dependent artifacts, at the cost of less GPU-friendly light caches. For instance, Instant Radiosity (IR) methods [Kel97] work with VPLs, a set of secondary point light sources, directly generated on the geometry illuminated by the primary sources. Thus, a VPL set acts as a discrete representation of the scene's indirect lighting and allows to reduce computations drastically when approximating light bounces. *Reflective Shadow Maps* (RSM) [DS05, DS06] provide an efficient VPL generation mechanism by sampling the scene in light-view space. This method has been improved by adding a clustering strategy over the RSM pixels which allows to reduce the number of VPLs by keeping the relevant ones only [PKD12]. However, scaling up to massive data requires huge amounts of VPLs. This is challenging as both generation and shading costs of so many VPLs is prohibitive in dynamic scenes. This problem is typically addressed with hierarchical methods such as *Lightcuts* [WFA*05] or *Point-Based Global Illumination* [Chr08, REG*09], which aim at managing massive sets of VPLs using multiple level-of-details of the point-sampled light field. In this context, *ManyLODs* [HREB11] reached interactive rendering time by computing in parallel many coherent cuts in the VPL tree. To reach real-time performances on complex scenes with up to millions of lights, Olsson et al. [OBA12] chose to address fill-rate issues by clustering the GBuffer pixels using an extension of tiled shading [OA11]. However, these techniques are based on a tree structure which requires amortizing its construction over time, preventing full dynamism in the scene.

A challenging issue while simulating indirect illumination with IR is to compute visibility between VPLs and pixels. Numerous approaches have been developed to this end, such as the *Imperfect Shadow Maps* (or ISMs) [RGK*08] which are generated quickly and in droves from on a point sampling of the scene, and queried during shading to approximate indirect visibility. Virtual Area Lights (or VALs) [DGR*09] allow to reduce the number of visibility queries by clustering light emitters into small surfaces, with the visibility being approximated by computing soft shadows from VALs shadow maps. In order to amortize VPLs shadow computation, Laine et al. [LSK*07] proposed to select at each frame a subset of VPLs for which shadow maps are computed or updated. To improve temporal coherency, Barák et al. [BBH13] used a RSM to sample preferably the scene in region with high radiance. Recently, Hedman et al. [HKL16] developed a technique which generates a temporally coherent VPL sampling of large scenes by memorizing their position from frame to frame and only invalidating them when they no longer influence framebuffer pixels. Indirect visibility is then implicitly solved by using ray tracing to sample VPLs.

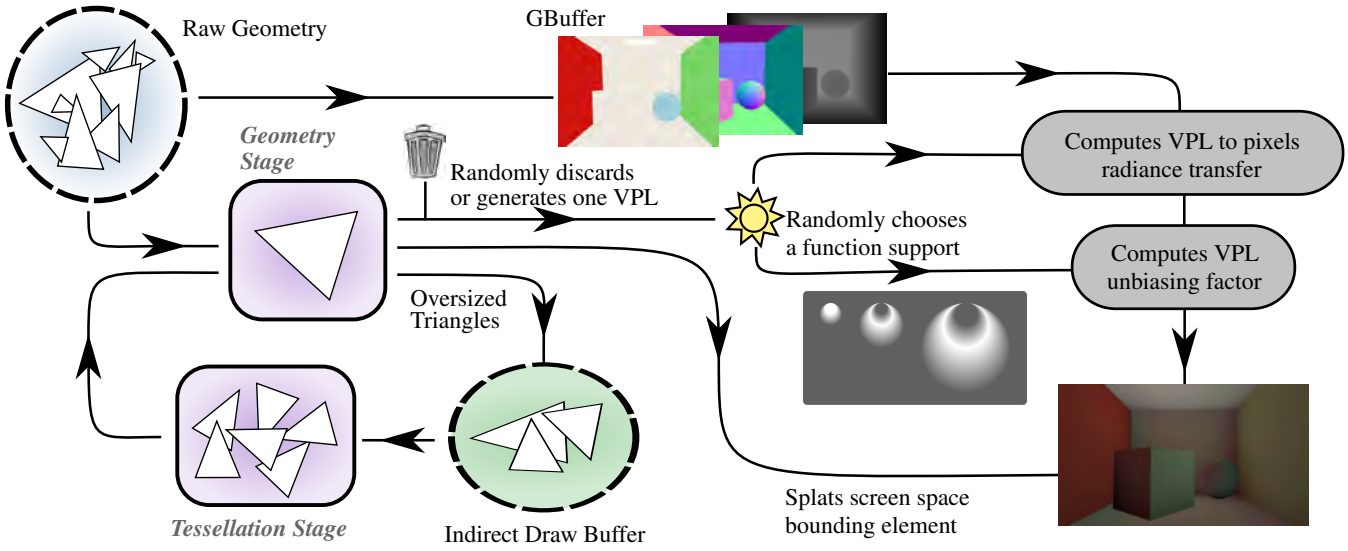


Figure 2: **Forward Light Cuts**. The raw geometry is sent to the geometry shader where it is split into regular and divergent triangles. Divergent triangles – *i.e.* with an area greater than a certain threshold – are used to fill an indirect draw buffer and then subdivided such that every new triangle may be considered as regular. Regular and subdivided triangles are then randomly distributed over $N+2$ subsets including one which is discarded. The surviving triangles are classified among subsets according to a probability distribution depending on their size. Finally, for each of these triangles, a single VPL is created whose power depends on the subset it belongs to and its support function is computed accordingly.

Note that our proposed algorithm does not address indirect visibility, as it is an orthogonal problem to the one we target: efficiently balancing VPL distribution to focus computations where needed.

Finally, the Deep Screen Space (DSS) approach [NRS14] proposes to exploit the advantages of both screen-space and object-space radiance caching. The same way as object-space strategies, this method generates on-surface VPLs, even on occluded geometry that may still impact the image; and similarly to screen-space approaches, it benefits from a native GPU support, with the tessellator unit – instead of the rasterizer – being used as a surface sampler to generate the VPLs. Still, although DSS can successfully be used for rendering small to medium size scenes, it cannot cope with larger ones, where the real-time constraint imposes decimating geometry rather than refining it. Our technique makes a step forward in this direction, by proposing a diffuse GI pipeline which can both refine and simplify the set of geometry-driven VPLs in a two-pass strategy. Exploiting both the tessellator unit and the geometry shader to adjust the resolution of an object-space radiance cache in the context of scenes with a massive number of triangles. In particular, we also reach real-time performance by using a multi-scale representation of the light field but, contrary to the aforementioned techniques, our method is fully dynamic and does not resort to any tree structure, nor imposes to maintain any data structure among frames.

3. Algorithm Overview

Our algorithm, illustrated in Fig. 2, has the structure of typical VPL based pipelines, composed of three main stages: VPL generation, indirect light caching and lighting with VPLs. Our contributions mainly focus on VPL generation and their usage during lighting, and can be summarized as follows:

- at loading time, we associate a single random integer to each vertex; this number will be used at rendering time to generate per-triangle random numbers consistently, even under dynamic geometry transformations (see Sec. 6);
- the full set of triangles $\mathcal{L}$ is then randomly partitioned according to a probability distribution (Sec. 4, Fig. 3); for each triangle, a VPL is stochastically generated and its outgoing radiance is computed based solely on its related partition ($\mathcal{L}_0 \dots \mathcal{L}_N$); in order to balance computations, we distribute many samples with small influence distance in $\mathcal{L}_0$ and decrease the number of samples while increasing the influence distance in $\mathcal{L}_k$ when k grows; this leads to a functional equation on the VPLs radiance that we derived in Sec. 5;
- in addition, to significantly reduce the number of triangles to manage while optimizing the amount of sampling information, a special set – mainly composed of small triangles – is completely discarded;
- furthermore, to properly cope with hardware restrictions when it comes to dynamic data amplification, we create exactly one VPL per triangle; as such, large triangles – that we call "divergent" (Sec. 4) – are not well sampled by the aforementioned strategy and may introduce important lighting artifacts in the final image; consequently, we reroute them through the tessellation unit, where they are subdivided to reach the proper resolution (see Sec. 6);
- last, we splat indirect illumination in a typical deferred shading process, with the splatted function supports depending on the VPL partition, reserving powerful VPLs to carry on distant lighting using crescent-shaped support (Sec. 5, Fig. 3).

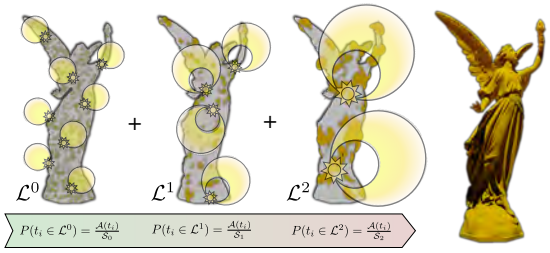


Figure 3: *Multi-scale radiance cache. In our approach, VPLs are distributed among different subsets, each of which having a bounded region of influence (in yellow). The final illumination computed at a given point uses samples from the different clusters to reconstruct the incoming radiance at shading time.*

4. VPLs Generation

Our VPL generation method is based on the standard hardware rasterization pipeline and is designed to exploit the GPU fine-grained parallelism by generating each VPL independently from the others. Moreover, our algorithm only implies a single draw pass of the geometry, enabling its use with scenes featuring a high polygon count. To do so, we distinguish regular triangles from divergent ones, *i.e.* the set of triangles $\{t_i\}$ with surface area $\mathcal{A}(t_i)$ greater than a certain threshold $\mathcal{S}_0$. We set

$$\mathcal{S}_0 = 4\pi \frac{D_{near}^2}{N_{avg}}, \tag{1}$$

which is a heuristic aiming at lighting pixels with approximately N_{avg} VPLs at least D_{near} far from them. Let R_{scene} be the scene radius, we typically set $D_{near} = 0.2 \times R_{scene}$ and N_{avg} between 64 and 1024 depending on the desired quality/speed tradeoff. We discuss how we handle divergent triangles in Sec. 6 and assume triangles to be regular in the remaining of this section.

Triangle decimation Small triangles contribute weakly to the final rendering for diffuse indirect lighting [DKH*14] and we tend to favor their removal in our pipeline. However, we cannot just discard every triangle smaller than a given threshold, since *groups* of small triangles may collectively have an important impact in the light transport reaching a distant point. We address this problem by adopting a stochastic decimation approach: we retain the contribution of heavily tesselated geometry by computing, for each triangle, a uniform random value u_{t_i} lying between 0 and 1. We then keep this triangle if $\mathcal{A}(t_i) > u_{t_i}\mathcal{S}_0$. Because every triangle is assumed to be regular, the probability for a triangle to be kept boils down to:

$$\forall t_i \in \mathcal{L},\ \ P(t_i \in \mathcal{L}^*) = \frac{\mathcal{A}(t_i)}{\mathcal{S}_0}, \tag{2}$$

where $\mathcal{L}$ denotes the set of all triangles and $\mathcal{L}^*$ the set of kept triangles. This means that the smaller a triangle is, the greater its chance to be discarded becomes. At the same time, this partitioning translates into a uniform distribution of samples over the entire scene surface, such that the expectation of the surviving triangle count is $\mathbb{E}[N_{sample}] = \frac{\mathcal{A}_{Scene}}{\mathcal{S}_0}$, with $\mathcal{A}_{Scene}$ the total scene area.

Triangle multiscale partitioning Once small triangles have been discarded, we randomly dispatch the remaining ones ($\mathcal{L}^*$) in a partition $(\mathcal{L}^0, \ldots, \mathcal{L}^N)$ such that $\mathcal{L}^N$ contains a few triangles and $\mathcal{L}^k$ is more and more populated when k gets closer to 0. To do so, we introduce the sequence of $N+1$ increasing values $\{\mathcal{S}_0 < \cdots < \mathcal{S}_N\}$ representing the desired VPL partitioning. By further defining in Alg. 1:

$$\forall k \in [0 \ldots N],\quad \tilde{\mathcal{S}}_k = \frac{1}{\sum_{j=0}^{k} \frac{1}{\mathcal{S}_j}},$$

a multiscale partitioning of representative scene triangles emerges from our decimation strategy (Fig. 3), with the probability for a triangle to lie in the subset $\mathcal{L}^k$ being:

$$\forall k \in [0 \ldots N],\quad P(t_i \in \mathcal{L}^k) = \frac{\mathcal{A}(t_i)}{\mathcal{S}_k}. \tag{3}$$

Note that, with this definition, a triangle is considered as divergent if its area is greater than $\tilde{\mathcal{S}}_N$.

In such a way, the key property of our approach at this stage is that we **do not** generate, maintain or manage any kind of explicit hierarchy – because we affect a triangle to a certain subset independently from the choice made for any other – while still being able to later gather an adaptive multiscale light cut. This clearly favors parallel execution, however, this also means that a given triangle, located in a given subset, does not capture any coarse-grained information carried by finer triangles. This issue is discussed and partially addressed in Sec. 5.

Algorithm 1 Multiscale Partition

```
procedure ComputeLevel(u_{t_i}, t_i)
    for k ← 0...N do
        if u_{t_i} < A(t_i)/S̃_k then
            return k
        end if
    end for
    DiscardTriangle( )
end procedure
```

5. Lighting with VPLs

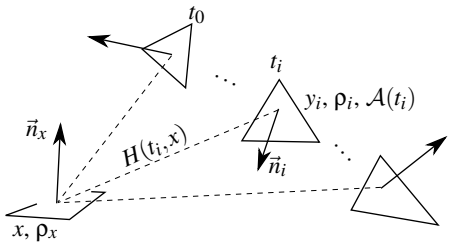


Figure 4: *VPL illumination at point x*

In the many-lights framework, the indirect outgoing radiance $L(x, \vec{n}_x)$ of a point x with normal $\vec{n}_x$ is approximated by $L^{ML}(x, \vec{n}_x)$ which is defined as the discrete sum of the radiance coming from a set of VPLs:

$$L^{ML}(x, \vec{n}_x) = \sum_{t_i \in \mathcal{L}} H(t_i, x, \vec{n}_x)\mathcal{A}(t_i), \tag{4}$$

where $\mathcal{L}$ represents the set of all triangles in the scene and $H(t_i,x,\vec{n}_x)$ stands for the incoming radiance transfer function starting from t_i toward the receiver x oriented by $\vec{n}_x$ (Fig. 4). For a diffuse receiver with albedo ρ_x, this function is defined as:

$$H(t_i,x,\vec{n}_x) = L(t_i,y_i\bar{x})\frac{\rho_x}{\pi}\frac{\langle\vec{n}_x,x\bar{y}_i\rangle^+\langle\vec{n}_i,y_i\bar{x}\rangle^+}{d_i^2}, \tag{5}$$

where $\bar{u}=\frac{\vec{u}}{\|\vec{u}\|}$, $\langle\vec{u},\vec{v}\rangle^+=\max(0,\langle\vec{u},\vec{v}\rangle)$, $L(t_i,y_i\bar{x})$ is the radiance leaving the VPL centered at $y_i\in t_i$ toward the direction $y_i\bar{x}$ and $d_i=\max(\varepsilon,\|x\vec{y}_i\|)$ is the distance between y_i and x clamped to a user parameter ε to avoid singularities. We model the first diffuse bounce of light with the following VPL outgoing radiance expression:

$$L(t_i,y_i\bar{x}) = \rho_i E(t_i)\frac{3}{2\pi}\langle\vec{n}_i,y_i\bar{x}\rangle^+, \tag{6}$$

where $E(t_i)$ is the direct irradiance falling to the triangle t_i. Note that because of the term $\langle\vec{n}_i,y_i\bar{x}\rangle^+$, these reflectors cannot be considered as perfectly lambertian anymore. Although light is therefore preferably reflected in the direction of the geometric normal, our experiments show that no important changes appear, while this greatly alleviates upcoming computations. The term $\frac{3}{2\pi}$ comes to ensure energy conservation, with the radiosity $B(t_i)$ and the irradiance $E(t_i)$ being related by:

$$B(t_i) = \int_\Omega L(t_i,\omega)\langle\vec{n}_i,\omega\rangle^+ d\omega = \rho_i E(t_i)$$

Approximating VPL lighting We propose to approximate the computation of $L^{ML}(x,\vec{n}_x)$ by summing the contribution of a subset of VPLs (Fig. 3), *i.e.* the ones lying in $(\mathcal{L}^0,\dots,\mathcal{L}^N)$. Thus, we define $K(x,\vec{n}_x)$ an estimator of $L^{ML}(x,\vec{n}_x)$ as follows:

$$K(x,\vec{n}_x) = \sum_{k=0}^{N}\sum_{t_i\in\mathcal{L}^k} H(t_i,x,\vec{n}_x)F^k(t_i,x), \tag{7}$$

$F^k(t_i,x)$ is an unknown function of the position x, the emitter t_i and the index k. Its equation is derived below.

By computing the expectation of $K(x,\vec{n}_x)$ over the set of every possible partition $(\mathcal{L}^0,\dots,\mathcal{L}^N)$, we get:

$$\begin{aligned}\mathbb{E}[K(x,\vec{n}_x)] &= \mathbb{E}\left[\sum_{k=0}^{N}\sum_{t_i\in\mathcal{L}^k} H(t_i^k,x,\vec{n}_x)F^k(t_i^k,x)\right]\\ &= \sum_{t_i\in\mathcal{L}} H(t_i,x,\vec{n}_x)\mathbb{E}\left[\sum_{k=0}^{N} F^k(t_i,x)\mathbb{1}_{[t_i\in\mathcal{L}^k]}\right]\\ &= \sum_{t_i\in\mathcal{L}} H(t_i,x,\vec{n}_x)\sum_{k=0}^{N} F^k(t_i,x)P(t_i\in\mathcal{L}^k),\end{aligned} \tag{8}$$

where $\mathbb{1}_{[t_i\in\mathcal{L}^k]}$ is the indicator function, that equals to 1 if $t_i\in\mathcal{L}^k$ and 0 otherwise. If we want $K(x,\vec{n}_x)$ to represent an unbiased estimator of the incoming radiance $L^{ML}(x,\vec{n}_x)$, we have to verify the following functional equation on F^k:

$$\forall x,\quad \sum_k F^k(t_i,x)P(t_i\in\mathcal{L}^k) = \mathcal{A}(t_i). \tag{9}$$

According to our VPL partitioning strategy (see Eqn. 3), we define

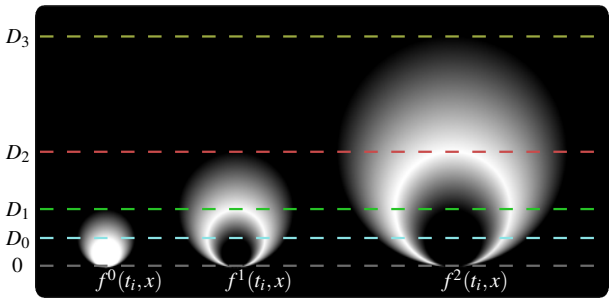


Figure 5: *Visualization of our support functions $f^k(t_i,x)$ on a planar section, for values ranging from 0 (black) to 1 (white). $\mathcal{B}_h(t_i)$ are the isolevels of these functions.*

F^k as:

$$F^k(t_i,x) = \mathcal{S}_k f^k(t_i,x), \forall[0\dots N], \tag{10}$$

which translates the unbiased condition (Eqn. 9) to a partition of unity problem, seeking for a set of functions $(f^k)_k$ such that:

$$\forall x,\quad \sum_k f^k(t_i,x) = 1. \tag{11}$$

Choice of partition of unity Inspired from PBGI tree cuts strategies [Chr08], we introduce a family of nested balls $\mathcal{B}_h(t_i)$ characterized by $h>0$. For a given h, $\mathcal{B}_h(t_i)$ represents the set of points for which the contribution of the VPL is significant. This means that for each point x outside of $\mathcal{B}_h(t_i)$, the function $H(t_i,x,\vec{n}_x)$ has a smaller value than h, whatever the orientation of the receiver $\vec{n}_x$:

$$\forall h\in\mathbb{R}^*,\quad \mathcal{B}_h(t_i) = \left\{x\in\mathbb{R}^3 \text{ s.t. } \max_{\vec{n}} H(t_i,x,\vec{n})\geq h\right\}. \tag{12}$$

Furthermore, $H(t_i,x,\vec{n})$ (Eqn. 5) is maximal when the receiver is front facing the emitter, *i.e.* $\vec{n}=x\bar{y}_i$. Thus, with our VPL radiance distribution model, we can write:

$$\mathcal{B}_h(t_i) = \left\{x\in\mathbb{R}^3 \text{ s.t. } \frac{\|x-y_i\|}{\langle\vec{n}_i,x\bar{y}_i\rangle^+}\leq D(h)\right\}, \tag{13}$$

where

$$D(h) = \frac{1}{\pi}\sqrt{\frac{3\rho_x\rho_i E(t_i)}{2h}}.$$

Hence, as $D(h)$ does not depend on x, $(\mathcal{B}_h(t_i))_{h\in\mathbb{R}^*}$ is a family of nested balls whose frontier owns y_i and center lies on the line (y_i,n_i) (Fig. 5). We impose that our 3D unit partition $(f^k)_k$ is constant over the spheres being the frontier of a $\mathcal{B}_h(t_i)$. Then, by defining the following mapping from $\mathbb{R}^3$ to $\mathbb{R}$:

$$\forall x\in\mathbb{R}^3,\quad d(t_i,x) = \frac{\|x-y_i\|}{\langle\vec{n}_i,x\bar{y}_i\rangle^+}, \tag{14}$$

our problem boils down to a 1D partition of unity $(\tilde{f}^k)_k$:

$$\forall x\in\mathbb{R}^3,\quad f^k(t_i,x) = \tilde{f}^k(d(t_i,x)). \tag{15}$$

Since $(f^k)_k$ will be used as splatting functions during rendering, we aim at making them as smooth as possible while keeping them

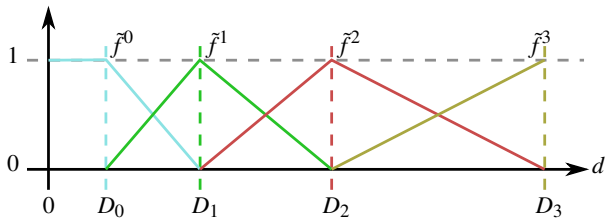


Figure 6: *One dimensional partition of unity*

easy to compute and define them as the following set of continuous piecewise affine functions with compact support:

$$\forall d \in \mathbb{R},\quad \tilde{f}^k(d) = \begin{cases} 1 & \text{if } k = 0 \text{ and } d \in]0, D_1] \\ \frac{d - D_{k-1}}{D_k - D_{k-1}} & \text{if } k > 0 \text{ and } d \in]D_{k-1}, D_k] \\ \frac{D_{k+1} - d}{D_{k+1} - D_k} & \text{if } k > 0 \text{ and } d \in]D_k, D_{k+1}] \\ 0 & \text{otherwise} \end{cases}. \quad (16)$$

Where $\{D_k\}$ allow to specify the influence distance of each level (Fig. 5, 6).

Parameters setting In order to mimic the traditional hierarchical representations used with light fields, we generate our partition with subsets size expectation that decreases geometrically. Furthermore, while $\mathcal{S}_k$ may be understood as the average surface of triangles lying in the level $\mathcal{L}^k$, we define them by:

$$\mathcal{S}_k = \mathcal{S}_0 \mu^k, \quad (17)$$

where $\mu > 1$ is a user-defined real number. Depending on the scene and the number of levels, we typically set μ between 1.4 and 5. In addition, still by mimicking the hierarchical approaches, we propose to define the distance of VPLs influence such that each point in space is reached by a controlled number of VPLs, which may be translated into:

$$\begin{cases} D_k = \sqrt{\mathcal{S}_0 \mu^k}, \quad \forall k \in [0 \dots N] \\ D_{N+1} = D_N \end{cases}, \quad (18)$$

6. Implementation details

We implemented our method with the OpenGL 4.4 API.

Pipeline description As depicted in Fig. 2, our pipeline only contains two geometry draw passes of the entire scene: one to generate the GBuffer and one to generate and splat the VPLs. This moderate use of the raw geometry is an important metric for our application scenarios because we aim at managing scenes with a large number of polygons. In fact, a third geometry draw pass occurs, but involves only a fraction of the scene: the divergent triangles. As the divergence criterion is set such that the number of divergent triangles remains small, this last pass is not computationally prohibitive. Our algorithm exploits intensively the geometry shader stage to perform computations on a per-triangle basis rather than a per-fragment or a per-vertex one. Fortunately, for recent GPU architecture, the formally prohibitive overhead of the geometry shader stage has been greatly reduced, enabling polygon-wise computations for large streams.

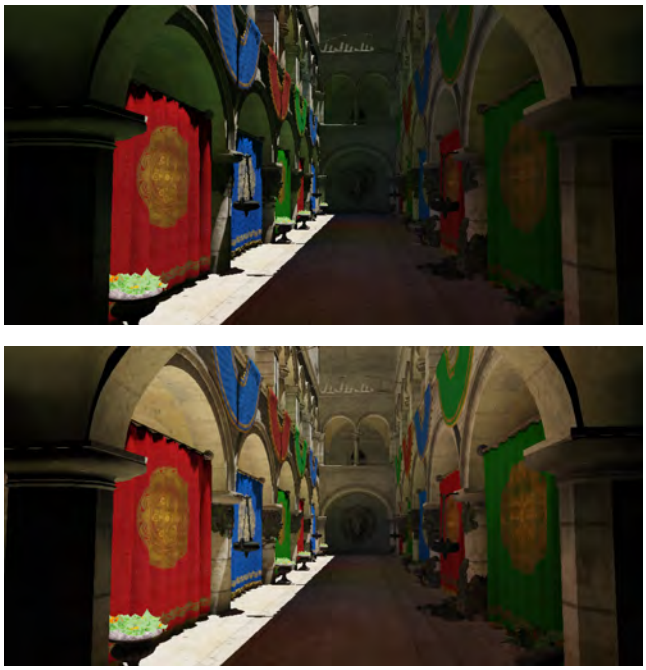


Figure 7: *FLC on the Crytek Sponza. Top: without the divergent pipeline, only the heavy tessellated geometry (green flowers and arc hessian) cast indirect lighting. Bottom: With the full pipeline, the ground (4 triangles) light bounce reveals much of the scene.*

Divergent triangle management Current hardware tessellation units are not designed to manage massive triangle sets as input, inducing a noticeable overhead while processing a triangle even if this triangle does not require any subdivision. At the same time, the geometry stage of these architectures are extremely efficient at discarding or letting polygons pass trough, *i.e.* when no geometry amplification is mandatory. This motivates us to design our indirect lighting pipeline with the two following passes. In the first pass, the entire scene geometry is processed but the tesselation stage is disabled. Divergent triangles are detected at the geometry stage and stored in a separate buffer, while regular ones are stochastically sampled. In order to manage scenes with numerous materials and textures, we also store a per-triangle material index, used in the following pass to fetch information from a material or texture atlas.

Using the OpenGL "glDrawArrayIndirect" feature, the divergent buffer is subsequently directly used as input geometry for the second pass without any CPU synchronization. The tessellation stage is solely activated for this particular pass and used to subdivide the triangles such that their area becomes small enough to be processed by our regular pipeline. In general, the number of large triangles is relatively small compared to the total triangle count. Consequently, the overhead induced by the divergent buffer filling and vertex processing is negligible compared to the visual effect improvement (see Fig. 7 for instance).

Per-triangle random number generation To evaluate Alg. 1 for each triangle, we use a pseudo-random value for the variable u_{t_i}. The generation of this value only requires the mesh to own an additional per-vertex uint32 attribute – v_rand. This attribute is initial-

Algorithm 2 Per-triangle random value computation

```
1:  uniform uint32 u_rand
2:  uniform texture2D noise
3:  function REGULARRAND(ivec3 v_rand)
4:      return u_rand xor v_rand.x xor v_rand.y xor v_rand.z
5:  end function
6:  function DIVERGENTRAND(ivec3 v_rand, vec2
    tess_coord[3])
7:      ivec3 v_tess_rand
8:      v_rand_tess.x = TEXTURE(noise, tess_coord[0])
9:      v_rand_tess.y = TEXTURE(noise, tess_coord[1])
10:     v_rand_tess.z = TEXTURE(noise, tess_coord[2])
11:     return REGULARRAND(v_rand xor v_tess_rand)
12: end function
```

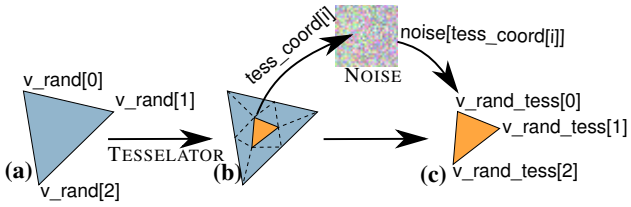


Figure 8: *Generation of per-triangle uniform random values in the divergent pipeline. Each divergent triangle (a) is first subdivided. For each sub-vertex (b), its barycentric coordinates* tess_coord *are used to fetch a random value in the noise texture. Finally the fetched value is used to initialize the subtriangle (c) random attribute.*

ized when loading the mesh with random values uniformly chosen between 0 and $2^{32}-1$. In the regular pipeline, at the geometry shading stage, u_{t_i} is computed as a xor between the random attributes of the three vertices of t_i. Note that the choice of the xor operator avoids correlation among two or three triangles. Moreover, new random values can be generated at any time by using u_rand, a global uniform random value that may be updated at most once per frame. Being xor'd with the original per-vertex values, it allows to get whole new random distributions over the mesh (Alg. 2).

For the divergent pipeline, the construction of u_{t_i} is quite more subtle. Indeed, this value remains unaltered, even under camera motion or mesh deformation, *i.e.* as long as the mesh topology remains the same. To preserve these properties, we perform the tessellation in the model space and exploit the fact that the tessellation pattern only depends on the input triangle shape. Indeed, in our use cases, the tessellation parameters are only determined by the original triangle area. Therefore, during the tessellation evaluation stage, we use the barycentric coordinates of the generated vertices to fetch a uint32 value, named v_rand_tess, from a precomputed noise texture. Thus, to generate u_{t_i} for a triangle sprung from the subdivision, we compute a xor between the three v_rand_tess, the three base triangle v_rand and finally the global uniform random value u_rand (Fig. 8, Alg. 2).

Progressive Rendering Since the VPL support functions defined in Eqn. 16 are smooth, our algorithm produces at each frame a visually plausible rendering without high frequency artifacts. Nevertheless, with the aforementioned global uniform variable u_rand, it is straightforward to generate many independent renderings of the same scene with our approach. In addition, for each triangle, we can easily derive two new independent random values from u_rand. These values are used to jitter the VPL center y_i defined in Eqn. 5 over the triangle t_i. Then, by averaging all these renderings with jittered VPLs, Eqn. 8 provides a result that is close to the true solution of our problem, *i.e.* the computation of the first bounce outgoing radiance on surface composed of diffuse reflectors and ignoring indirect visibility. While this means that indirect lighting is computed by integrating over every scene triangle, this progressive version of our technique is able to manage any kind of disturbed geometry (*e.g.* normal mapping, alpha tested) and could also be extended to manage emissive textured geometry (Fig. 11).

	Cornell Box Tiling ×64	Power Plant Tiling ×64
GBuffer + SM	**0.3**	**3.6**
Indirect	**3.5**	**8.5**
Regular Pipeline	0.9	6.5
Divergent Pipeline	1.0	0.8
Upscaling + Blur	1.2	1.2
Full frame	**3.8**	**12.1**

Table 1: *Rendering time break drown (ms) at* 1280×720 *resolution, for the Cornell Box (1K tri.) and the PowerPlant (12M tri.)*

Splatting indirect illumination The way the VPL contributions are summed to simulate the indirect lighting is orthogonal to the previous discussion. Hence, to keep a simple pipeline, we use a splatting strategy similar to deferred lighting [ST90]. In particular, this allows to manage geometry decimation, VPL generation and lighting in a single shader program. Indeed, besides determining whether a triangle is divergent or not and which is the level of generated VPLs, we use the geometry shader to transform input triangles in *sized* point primitives which encompasses the underlying VPL screen space function support. Although the major drawback of such a single-pass lighting pipeline is the fillrate consumption, our algorithm aims at simulating a low frequency phenomenon. Therefore, undersampling the resulting signal appears as a reasonnable optimization. To do so, we partition the viewport in $4^{N_{\text{tiling\_level}}}$ tiles and assign to each pixel in the viewport a unique tile pixel at the same relative location – this technique is often referred as *interleaved sampling* [KH01, LSK*07]. Next, at splatting time, a tile is randomly chosen for each VPL thus dividing the number of touched pixels by $4^{N_{\text{tiling\_level}}}$. We finally recompose the image by untiling the buffer and blurring it to remove the generated noise.

7. Results

All our experiments are performed on a standard PC equipped with a Quadro M6000 GPU. Tab. 1 gathers timings of our FLC algorithm on two scenes: one very simple and one containing a much more complex geometry. We observe that the overall performance of our algorithm mainly depends on two factors: the framebuffer resolution and the number of triangles composing the scene (Fig. 13). Note also that the divergent pipeline costs roughly the same in both cases.

When geometry is not the bottleneck, the framerate heavily depends on the number of lower resolution subtiles used when splat-

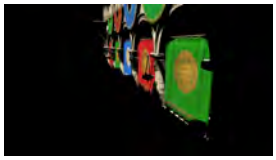
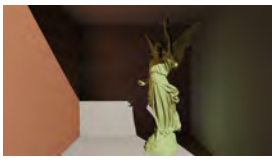
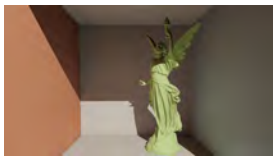
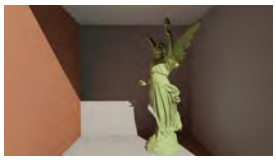
Model	Direct lighting only	With SSDO	With DSS	With FLC (ours)
Sponza 270k tri.	 4ms	 7ms	 19ms	 8ms
Oil Platform 700k tri.	 1ms	 5ms	 22ms	 14ms
Car 3M tri.	 2ms	 6ms	 28ms	 20ms
San Miguel 12M tri.	 9ms	 13ms	 35ms	 22ms
Lucy 28M tri.	 18ms	 20ms	 80ms	 35ms

Table 2: Comparisons performed at a 1280x720 resolution. All high resolution images are provided as supplemental material.

ting indirect illumination. However, even if no information is lost from the viewport sampling, reducing subtile resolution also comes with drawbacks such as removing geometry details in shadowed area (Fig. 9), due to the indirect lighting interpolation. The number of required VPLs depends on the scene (Fig. 10) and induces fillrate, bandwidth and shading costs which obviously influences the FLC framerate. However, although the number of used VPLs is proportional to rendering time, the overall visual quality does not map linearly to this value. In fact, we observe that the degradation brought with our downscale strategy is well compensated by increasing the number of VPLs. This allows trading VPLs for subtiles to adjust a visually pleasant real-time framerate.

Comparison We evaluate our FLC approach against two algorithms that we reimplemented in our framework: the popular SSDO method [RGS09] and the more recent DSS approach [NRS14]. These two algorithms share the same properties as our FLC method: (i) they run in real time (ii) with no preprocessing (i.e., on fully dynamic scenes) and they reproduce one-bounce, unshadowed diffuse indirect lighting. We perform our comparison using 5 scenes that exhibit different structures, polygon counts and lighting conditions. Table 2 provides resulting images obtained by the three different approaches and reports the complete frame generation time in each case. First, we can observe that our method succeeds at producing similar or better result than DSS, while being significantly faster. This behavior is emphasized when the polygon count grows and can be explained by our rerouting strategy, which reduces significantly the tessellator work load for large models. Second, when comparing to SSDO, we can observe that this method often fails at revealing the scene regions which are not directly lit, but should receive a signifcant amount of first bounce indirect lighting. Of course, being fully screen-space, the SSDO framerate remains high even with large scenes, but the visual improvement appears to be small compared to direct lighting only, while our approach, just such as DSS, reproduces better and more consistent long range indirect lighting. In addition, Fig. 11 illustrates the execution of our approach when enabling progressive rendering. We compute the average of many renderings for which we change the random seed at each frame. Each rendering is done with a high number of subtiles and a few number of VPLs which allows to improve the rendering time until convergence. We can visually assess

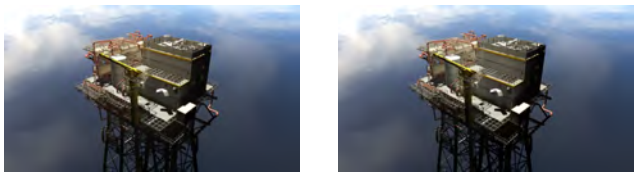


Figure 11: (Left) Progressive rendering (250ms) by averaging 50 renderings. (Right) Real time rendering (14 ms).

that, at the cost of a longer rendering time, this strategy allows to completely remove any kind of artifact (like spikes).

Limitations and future work Our algorithm does not produce indirect shadow casting (Fig. 12) nor multiple light bounces since splatting indirect illumination in screen space does not provide any direct mean to simulate such phenomena. A future approach would be to incorporate object space strategies [OBA12] to address these issues, while still preserving the scalability and full dynamic-compatibility of our method. The second limitation of our approach is its current restriction to diffuse indirect lighting: incorporating glossy reflectors in our mathematical framework is an important research direction for managing scenes which are complex both from a geometric and a material point-of-view.

8. Conclusion

We have proposed *Forward Light Cuts*, a novel approach to compute diffuse indirect global illumination in real-time. Our geometric method generates VPLs using a stochastic decimation process of the input triangles within a two-stages pipeline that either simplifies or refines the scene's geometry to reach a suitable radiance caching resolution. Our approach does not imply any complex preprocessing nor requires carrying complex data structures over frames. It is compatible with large fully dynamic scenes, including light, view point and geometry animations. Last, in terms of integration, our approach naturally fits modern graphics pipelines and does not make any strong assumption on the complementary rendering techniques employed by the host application. In addition to the mathematical basis of our method, we evaluated it on a number of scenes with high polygon counts, ranging from CAD models to scans, and reported interactive performances in each case.

References

[BBH13] BARÁK T., BITTNER J., HAVRAN V.: Temporally coherent adaptive sampling for imperfect shadow maps. In *Comp. Graph. Forum* (2013), vol. 32, pp. 87–96. 2

[BJ03] BASRI R., JACOBS D. W.: Lambertian reflectance and linear subspaces. *IEEE Trans. PAMI 25*, 2 (2003), 218–233. 1

[Chr08] CHRISTENSEN P.: Point-based approximate color bleeding. *Pixar Technical Notes 2*, 5 (2008), 6. 2, 5

[DGR*09] DONG Z., GROSCH T., RITSCHEL T., KAUTZ J., SEIDEL H.-P.: Real-time indirect illumination with clustered visibility. In *Proc. VMV* (2009), pp. 187–196. 2

[DKH*14] DACHSBACHER C., KŘIVÁNEK J., HAŠAN M., ARBREE A., WALTER B., NOVÁK J.: Scalable realistic rendering with many-light methods. In *Comp. Graph. Forum* (2014), vol. 33, pp. 88–104. 2, 4

[DS05] DACHSBACHER C., STAMMINGER M.: Reflective shadow maps. In *Proc. I3D* (2005), pp. 203–231. 2

[DS06] DACHSBACHER C., STAMMINGER M.: Splatting indirect illumination. In *Proc. I3D* (2006), pp. 93–100. 2

[HKL16] HEDMAN P., KARRAS T., LEHTINEN J.: Sequential monte carlo instant radiosity. In *Proc. I3D* (2016), pp. 121–128. 2

[HREB11] HOLLANDER M., RITSCHEL T., EISEMANN E., BOUBEKEUR T.: Manylods: Parallel many-view level-of-detail selection for real-time global illumination. In *Comp. Graph. Forum* (2011), vol. 30, pp. 1233–1240. 2

[Kaj86] KAJIYA J. T.: The rendering equation. In *Proc. SIGGRAPH* (1986), vol. 20, pp. 143–150. 1

[Kel97] KELLER A.: Instant radiosity. In *Proc. SIGGRAPH* (1997), pp. 49–56. 2

[KH01] KELLER A., HEIDRICH W.: *Interleaved sampling*. Springer, 2001. 7

[LSK*07] LAINE S., SARANSAARI H., KONTKANEN J., LEHTINEN J., AILA T.: Incremental instant radiosity for real-time indirect illumination. In *Proc. EGSR* (2007), pp. 277–286. 2, 7

[Mit07] MITTRING M.: Finding next gen: Cryengine 2. In *SIGGRAPH 2007 courses* (2007), pp. 97–121. 2

[MMNL14] MARA M., MCGUIRE M., NOWROUZEZAHRAI D., LUEBKE D.: *Fast global illumination approximations on deep G-buffers*. Tech. rep., Tech. Rep. NVR-2014-001, NVIDIA Corporation., 2014. 2

[NRS14] NALBACH O., RITSCHEL T., SEIDEL H.-P.: Deep screen space. In *Proc. I3D* (2014), pp. 79–86. 3, 8

[OA11] OLSSON O., ASSARSSON U.: Tiled shading. *Journal of Graphics, GPU, and Game Tools 15*, 4 (2011), 235–251. 2

[OBA12] OLSSON O., BILLETER M., ASSARSSON U.: Clustered deferred and forward shading. In *Proc. HPG* (2012), pp. 87–96. 2, 9

[PKD12] PRUTKIN R., KAPLANYAN A., DACHSBACHER C.: Reflective shadow map clustering for real-time global illumination. In *Proc. EUROGRAPHICS Short Papers* (2012), pp. 9–12. 2

[RDGK12] RITSCHEL T., DACHSBACHER C., GROSCH T., KAUTZ J.: The state of the art in interactive global illumination. In *Comp. Graph. Forum* (2012), vol. 31, pp. 160–188. 2

[REG*09] RITSCHEL T., ENGELHARDT T., GROSCH T., SEIDEL H.-P., KAUTZ J., DACHSBACHER C.: Micro-rendering for scalable, parallel final gathering. In *ACM Trans. Graph.* (2009), vol. 28, p. 132. 1, 2

[RGK*08] RITSCHEL T., GROSCH T., KIM M. H., SEIDEL H.-P., DACHSBACHER C., KAUTZ J.: Imperfect shadow maps for efficient computation of indirect illumination. In *ACM Trans. Graph.* (2008), vol. 27, p. 129. 2

[RGS09] RITSCHEL T., GROSCH T., SEIDEL H.-P.: Approximating dynamic global illumination in image space. In *Proc. I3D* (2009), pp. 75–82. 2, 8

[RH01] RAMAMOORTHI R., HANRAHAN P.: On the relationship between radiance and irradiance: determining the illumination from images of a convex lambertian object. *JOSA A 18*, 10 (2001), 2448–2459. 1

[ST90] SAITO T., TAKAHASHI T.: Comprehensible rendering of 3-d shapes. In *Proc. SIGGRAPH* (1990), vol. 24, pp. 197–206. 7

[WFA*05] WALTER B., FERNANDEZ S., ARBREE A., BALA K., DONIKIAN M., GREENBERG D. P.: Lightcuts: a scalable approach to illumination. In *ACM Trans. Graph.* (2005), vol. 24, pp. 1098–1107. 1, 2

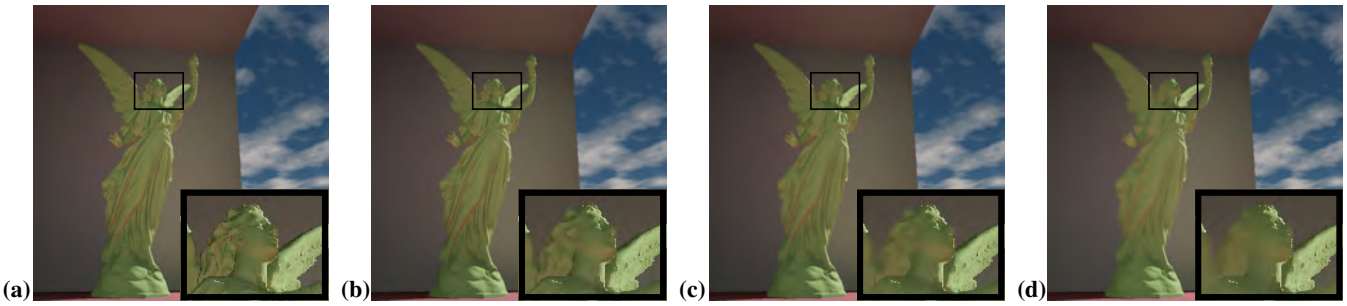


Figure 9: Influence of the tiling resolution parameter on the Lucy statue (28M triangles) in the Cornell Box model. The viewport resolution is 1024×1024 pixels and we vary the tiling level from 0 (a) to 3 (d). Because of indirect light interpolation, Lucy's face details are more and more blurred while reducing resolution. At the same time, the indirect splatting time is reduced from 93 ms (a), to 48 ms (b) and 33 ms (c, d). (c) and (d) rendering times do not vary because our algorithm is no more fillrate bottlenecked at this level.

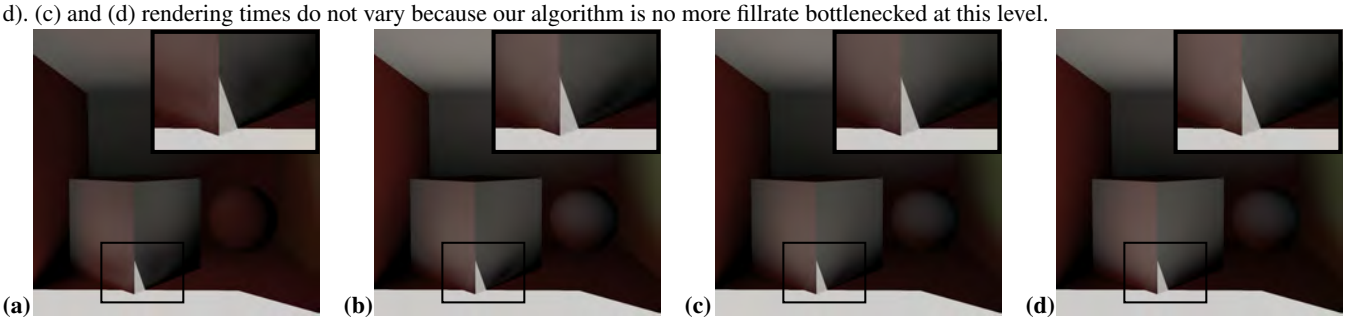


Figure 10: Indirect illumination computed with our algorithm on the Cornell Box model at 1024×1024 resolution. We fixed the number of partitions level to $N = 5$ and we compare the results by varying N_{avg}, *i.e.* the approximate number of VPLs influencing every pixel. Rendering times are directly proportional to this number – (a) $N_{avg} = 64$ is rendered in less than 1ms, (b) $N_{avg} = 128$ in 3ms, (c) $N_{avg} = 256$ in 7ms, (d) $N_{avg} = 512$ in 14ms. The close-up shows the typical artifacts appearing when N is not high enough.

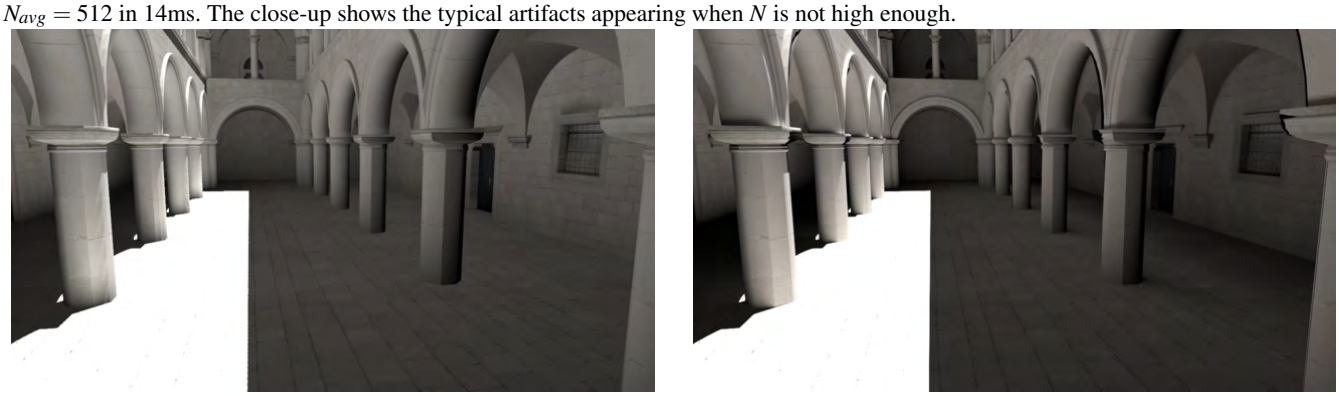


Figure 12: **Limitaton.** Comparison between our technique (left) and a ground truth solution (right) for the first indirect light bounce. The ground truth is rendered with path tracing using 8196 samples per pixels. We can observe the missing contact indirect shadow in our solution, mainly visible at the pillar bases on the right. Underneath the right archs also appears much brighter than the reference in our solution, which is due to the fact that the entire left facade illuminates this region without being occluded by the first floor.

Figure 13: *FLC on the Power Plant model (12M triangles). The scene is rendered at* 2560×1440 *pixels resolution in 8 ms for the direct lighting (a) and 25 ms for the direct and indirect lighting (c) per frame.*