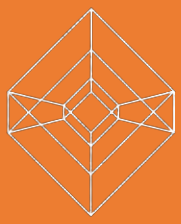


HARVEST4D

HARVESTING DYNAMIC 3D WORLDS FROM COMMODITY SENSOR CLOUDS

Publications for Task 4.4

Deliverable 4.41

Date:	30.6.2016
Grant Agreement number:	EU 323567
Project acronym:	HARVEST4D
Project title:	Harvesting Dynamic 3D Worlds from Commodity Sensor Clouds

Document Information

Deliverable number	D4.41
Deliverable name	Publications for Task 4.4
Version	1.0
Date	2016-06-30
WP Number	4
Lead Beneficiary	TUD
Nature	R
Dissemination level	PU
Status	Final
Author(s)	TUD

Revision History

Rev.	Date	Author	Org.	Description
0.1	2016-06-10	Tobias Plötz	TUD	First draft
1.0	2016-06-30	Tobias Plötz	TUD	Final version

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This deliverable contains original unpublished work except where clearly indicated otherwise.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 4.4, and how they fit into the work plan of the project.

The objective of Task 4.4 is to deal with challenges in multimodal registration and compression in the presence of large datasets. This involves addressing robustness and scalability with regards to both the quantity of data as well as its variety.

There are so far 3 publications that are mainly attributable to Task 4.4, and these can be found in the appendix of this deliverable. Furthermore, one publication has significant parts that belong to Task 4.4. These parts are also discussed here, but form part of other deliverables, hence they can be found on the Harvest4D webpage.

1.2 PUBLICATIONS

The following publications can be found in the appendix:

- Nicolas Mellado, Matteo Dellepiane and Roberto Scopigno.
  Relative scale estimation and 3D registration of multi-modal geometry using growing least squares.
  In: IEEE Transactions on Visualization and Computer Graphics, 2015
- Bas Dado, Timothy R. Kol, Pablo Bauszat, Jean-Marc Thiery and Elmar Eisemann.
  Geometry and attribute compression for voxel scenes.
  In: Computer Graphics Forum (Proc. Eurographics), Vol. 35, No. 2, 2016

At the time of delivery of this deliverable, the following working paper still contains original unpublished work. Therefore, it can only be accessed through the restricted section of the webpage (for papers under submission, conditionally accepted papers, etc.):

- Tobias Plötz and Stefan Roth
  Automatic registration of images to untextured geometry using average shading gradients.
  submitted to International Journal of Computer Vision (special issue ICCV'15)

The following publications can be found on the webpage only:

- Tim Golla and Reinhard Klein.
  Real-time point cloud compression.
  In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015

2 DESCRIPTION OF PUBLICATIONS

2.1 OVERVIEW

The publications in this deliverable tackle different aspects of scaling multimodal registration and compression algorithms to big datasets. First, robustness to heterogeneous data sources is of importance. To this end, "*Automatic registration of images to untextured geometry using average shading gradients"* [Plötz and Roth 2016] shows that our previous work on photo-to-geometry registration is also applicable to the more challenging task of painting-to-geometry registration, thus enabling the integration of historical depictions, real photographs and 3D models of a scene.

Second, this task considers scaling algorithms to big quantities of data. In "*Relative scale estimation and 3D registration of multi-modal geometry using growing least squares"* [Mellado et al. 2015], range scans from multiple sensors with varying intrinsic parameters, such as scale or sampling density, are registered into a single coherent model. Importantly, the method relies only on local computations, thus allowing it to scale to millions of points. For transmitting the resulting huge models, compression algorithms have to be applied. Here, "*Real-time point cloud compression"* [Golla and Klein 2015] contributes a method that achieves real-time performance even on big point clouds. Finally, multimodal compression is applied for real-time rendering of large scenes in "*Geometry and attribute compression for voxel scenes"* [Dado et al., 2016]. The paper proposes a novel compression scheme to store arbitrary per-voxel attributes, such as color, alongside the geometry. This allows to visualize high-resolution voxel scenes in real-time since they can be stored entirely on the GPU.

2.2 AUTOMATIC REGISTRATION OF IMAGES TO UNTEXTURED GEOMETRY USING AVERAGE SHADING GRADIENTS

In our previous work [Plötz and Roth 2015], we introduced a robust algorithm for registering photographs of a 3D scene to an untextured 3D model of the scene, thus allowing to transfer information between photograph and 3D model. In this paper [Plötz and Roth 2016] we demonstrate that our approach is robust enough to also register paintings and other non-photorealistic depictions to the 3D geometry. This opens up the possibility to integrate rich cultural heritage data with the 3D geometry, thus catering to the goals of Harvest4D. For example, paintings showing the same cultural site but spanning different epochs can be registered and then further analyzed in a common reference frame (see Figure 1).

Registering paintings is challenging since they usually abstract from realism deliberately. As can be seen in Figure 1, the depiction does not rigorously follow a perspective projection and the geometry may be drawn only approximately. The method described in [Plötz and Roth 2016] employs a two-stage registration approach. In the first stage, coarse registration hypotheses are formed based on local matching of image features between the painting and renderings of the 3D model. In the second stage, these coarse hypotheses are iteratively refined and afterwards

verified. Only few correct correspondences from the first stage are necessary to find a good registration, thus making the proposed method robust enough to register paintings.

The paper also evaluates the different parts of the registration algorithm in more depth. Regarding the choice of gradient representation for renderings, the analysis shows that the proposed average shading gradients consistently outperforms gradients from a single shaded image.

Figure 1:The method of [Plötz and Roth, 2016] allows registering paintings to untextured 3D models.

2.3 RELATIVE SCALE ESTIMATION AND 3D REGISTRATION OF MULTI-MODAL GEOMETRY USING GROWING LEAST SQUARES

The registration of 3D multimodal geometric data produced with different methods (laser scans, depth cams, multi-view stereo reconstruction, 3D modeling) is a challenging task due to the large heterogeneity of geometric properties of the input, like density, amount of data, scale, noise, deformation, connectivity and overlap. Therefore, [Mellado et al. 2015] presents a method for the registration of multi-modal geometric data, based on the Growing Least Squares (GLS) descriptor, that allows the measurement of the similarity between the geometry surrounding two points and also the estimation of their relative scale. The algorithm is scalable to millions of points since it is based on local computations. Hence, we can analyze and register large point clouds as aimed for in task 4.4.

In more detail, the proposed method uses the GLS descriptor to characterize point-based data and a logarithmic scale-space to get a scale-invariant signature. This description is robust to noise and provides a multi-scale point-wise comparison operator robust to variation of density and detail amount. The new approach computes point-wise multi-scale profiles and estimates the relative scale factor between points. The resulting operator is easy to implement and fast to evaluate, thus allowing the method to scale to very large datasets. This multi-scale comparison operator is used as component of a multi-modal registration procedure using a RANSAC approach to estimate the rigid transformation between the input models.

The proposed algorithm was tested on challenging cases in the registration of 3D models produced by CG modeling, or acquired using laser scanners, LIDAR, multiview stereo and spherical photogrammetry. The obtained registrations are very precise and are computed in a few minutes for input models consisting of millions of points (see **Figure *2***).

Figure 2:. (Left) Two input models with multi-scale profiles of similar points. (Middle) Registration after the multi-scale analysis. (Right) Registration after ICP starting from the position computed in the multi-scale analysis.

2.4 GEOMETRY AND ATTRIBUTE COMPRESSION FOR VOXEL SCENES

Like the previous method, the techniques developed in Harvest4D produce and process very large datasets. To handle these, decreasing the memory footprint is desirable. Due to the sheer magnitude of the data, scalability of the compression algorithms is especially valuable. For this reason, we propose a compression technique for 3D volumetric data that increasingly reduces the required memory per voxel as the voxel resolution increases. The method thus enables scalable real-time rendering of very high-resolution voxel scenes (see **Figure *3***).

Figure 3: Voxel scenes of high resolution can be stored and visualized after being compressed by the method of [Dado et al., 2016].

Voxel-based approaches are today's standard to encode volumetric data. Recently, directed acyclic graphs (DAGs) were successfully used for compressing sparse voxel scenes, but they are restricted to a single bit of (geometry) information per voxel. We present a method to compress arbitrary data such as colors, normals, or reflectance information alongside the geometry. By decoupling geometry and voxel data via a novel mapping scheme, we are able to apply the DAG

principle to encode the topology, while using a palette-based compression for the voxel attributes, leading to a drastic memory reduction.

Our method outperforms existing state-of-the-art techniques and is well-suited for GPU architectures. We achieve real-time performance on commodity hardware for colored scenes with up to 17 hierarchical levels (a $128K^3$ voxel resolution), which are stored fully in core due to reducing the memory footprint by up to two orders of magnitude. Furthermore, the approach enables several advanced rendering techniques such as color bleeding or reflections for voxel data, since normal and reflection information can be stored and compressed as well.

2.5 REAL-TIME POINT CLOUD COMPRESSION

While the previous paper handles big volumetric data, [Golla & Klein, 2015] consider compression of large multimodal point cloud data. Especially, it is beneficial to compress it in real-time directly as the points are acquired from the scanner. To this end, the paper proposes a compression approach that is able to compress colored 3D points at the scanner rate of 1.5 million points per second with compression ratios that are at least comparable to, but mostly better than previous compression approaches. See **Figure *4*** for a visual example of how the uncompressed point cloud compares to the original point cloud.

Our algorithm supports incrementally acquired data that is required, e.g., in online robotics applications. Decompression also works at real-time rates and is local, i.e. only the necessary parts of the data can be extracted. It inherently supports subsampling, i.e. a sub-sampled version of the point cloud can be reconstructed from the compressed representation. Where possible, we used standard techniques in order to make our algorithm easy to implement. Furthermore, our approach is able to trade compression ratios for speed. To the best of our knowledge, our algorithm is the first to exhibit all these properties.

Figure 4, Left: Screenshot of the original Frankenforst dataset, consisting of 43.6 million points (1.5GB). Right: Reconstruction from a compressed version with a file size of 3.76MB, an $RMSE$ of 0.013m and an $RMSE_{RGB}$ of 18.3 obtained by using the approach in [Golla & Klein, 2015].

3 REFERENCES

- Tim Golla and Reinhard Klein.
  Real-time point cloud compression.
  In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015
- Nicolas Mellado, Matteo Dellepiane and Roberto Scopigno.
  Relative scale estimation and 3D registration of multi-modal geometry using growing least squares.
  In: IEEE Transactions on Visualization and Computer Graphics, 2015
- Bas Dado, Timothy R. Kol, Pablo Bauszat, Jean-Marc Thiery and Elmar Eisemann.
  Geometry and attribute compression for voxel scenes.
  In: Computer Graphics Forum (Proc. Eurographics), Vol. 35, No. 2, 2016
- Tobias Plötz and Stefan Roth
  Automatic registration of images to untextured geometry using average shading gradients.
  Submitted to: International Journal of Computer Vision (special issue ICCV'15)
- Tobias Plötz and Stefan Roth
  Registering images to untextured geometry using average shading gradients.
  In: Proceedings of the IEEE International Conference of Computer Vision (ICCV), 2015

4 APPENDIX

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2505287, IEEE Transactions on Visualization and Computer Graphics

Relative scale estimation and 3D registration of multi-modal geometry using Growing Least Squares

Nicolas Mellado, Matteo Dellepiane, Roberto Scopigno

Abstract—The advent of low cost scanning devices and the improvement of multi-view stereo techniques have made the acquisition of 3D geometry ubiquitous. Data gathered from different devices, however, result in large variations in detail, scale, and coverage. Registration of such data is essential before visualizing, comparing and archiving them.
However, state-of-the-art methods for geometry registration cannot be directly applied due to intrinsic differences between the models, e.g. sampling, scale, noise. In this paper we present a method for the automatic registration of multi-modal geometric data, i.e. acquired by devices with different properties (e.g. resolution, noise, data scaling). The method uses a descriptor based on Growing Least Squares, and is robust to noise, variation in sampling density, details, and enables scale-invariant matching. It allows not only the measurement of the similarity between the geometry surrounding two points, but also the estimation of their relative scale. As it is computed locally, it can be used to analyze large point clouds composed of millions of points.
We implemented our approach in two registration procedures (assisted and automatic) and applied them successfully on a number of synthetic and real cases. We show that using our method, multi-modal models can be automatically registered, regardless of their differences in noise, detail, scale, and unknown relative coverage.

Index Terms—Multi-modal data, 3D Registration, Multi-scale descriptors

1 Introduction

In this paper, we focus on the registration of 3D multi-modal data describing surfaces, i.e. on point clouds or meshes generated by different acquisition devices (e.g. laser scans, depth cams, multi-view stereo reconstruction) or modeling tools. Indeed, acquisition devices are today ubiquitous and affordable, making the surface acquisition more and more popular. This is a consequence of the evolution of manual and assisted image modeling tools, of the development of automatic and robust methods for multi-view stereo reconstruction, and of the availability of low cost depth cameras like Kinect. As a consequence, multi-modal registration is today a common issue when working with 3D objects: a typical scenario is the registration of a low-resolution point-cloud to a high resolution mesh generated from 3D scanning.

The main difficulty when working with multi-modal data is to be robust to a large heterogeneity of geometric properties, e.g. noise, sampling, and scaling. Many approaches have been already proposed to register 3D surfaces, however they are mainly devoted to range maps alignment or non rigid registration, and usually assume almost-uniform geometric properties within the data. In practice they do not prove to be effective for multi-modal data, due to differences in terms of:

- *Density and detail amount*: surface details may not be present on some types of data, and sampling density can vary a lot.
- *Scale*: data coming from manual or assisted modeling, or structure from motion are in an arbitrary scale. Hence, a scale factor has to be estimated in addition to roto-translation.
- *Noise and deformation*: non-uniform noise and deformations are common in data which do not come from 3D scanning.
- *Connectivity*: multi-view stereo methods may not be able to provide a surface, but only a point cloud, hence the connectivity information may not be available.
- *Overlap*: the surface covered by the data is usually not known in advance, so no information about the amount of overlap is available.

A similar problem is faced by the Medical Imaging community, which have to analyze, register and process data acquired by different devices [2], [3], [4]. The main motivation is here to acquire different properties of the matter composing a subject for a given application purpose, and combine them in a meaningful manner. To do so, methods are designed to process low resolution 3D images, where the information is stored by voxel: algorithms take benefits of the regular structure (3D grid) as a support for computations. In our case, the lack of a regular spatial structure prevents to apply such methods on detailed meshes, and even less on unstructured data like point-clouds.

In this paper, we present a method for the registration of multi-modal geometric data, based on the Growing Least Squares descriptor (GLS) [5] and illustrated in Figure 1. This

- *Nicolas Mellado is a researcher at Université de Toulouse, UPS, IRIT. Part of this work has been done at University College London, Dept. of Computer Science, London.*
- *Matteo Dellepiane and Roberto Scopigno are Visual Computing Lab, ISTI-CNR, Pisa.*

1077-2626 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2505287, IEEE Transactions on Visualization and Computer Graphics

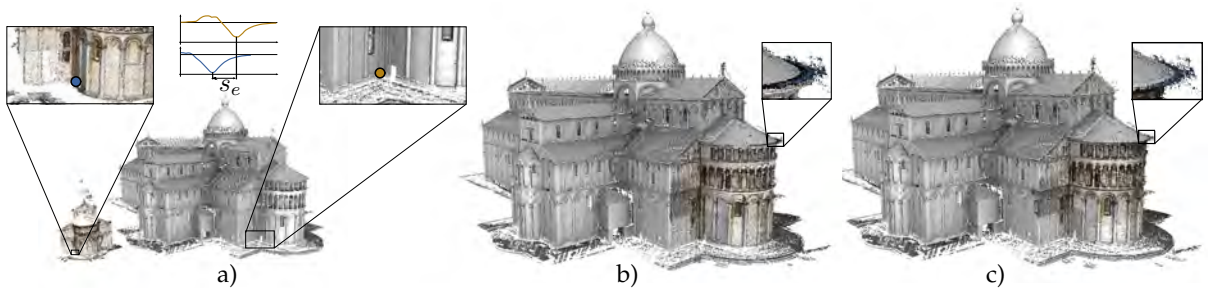


Fig. 1: We present a method for the automatic registration of 3D *multi-modal* discrete surfaces, ie. generated by different acquisition devices. a) Starting from a reference model (acquired by LIDAR) defining the scale of the scene, our approach automatically estimates a local relative scale s_e of a dense mesh (from multi-view stereo) by matching and comparing a descriptor based on Growing Least Square. b) Point-wise relative scales are used to register the two models using a RANSAC variant. c) Alignment and scale are refined using [1].

approach defines a meaningful scale-space representation for point-clouds, and provides both local and global descriptions of the geometry. The need to work with a meaningful multi-scale description is illustrated in Figure 2: a point $\mathbf{p}$ is within shapes of three different sizes: a local bump, the woman shoulder and the complete torso. The associated scales, respectively s_1, s_2, and s_3, can be detected using GLS and used for further processing, e.g. registration. The scale is here defined as the size of an Euclidean neighborhood around $\mathbf{p}$.

Based on this analysis framework, we present a new approach to estimate the relative scale factor between two multi-modal models. In opposite to previous work, our approach is robust to intra and inter-model variations of sampling and noise, and doesn't require an intermediate surface reconstruction step. As a consequence, our approach can be applied directly on large and raw point clouds, just after registration and without any pre-process. Our contributions can be summarized as follow:

- We present a new robust point-wise *scale estimation* and *multi-scale comparison operator*, which is invariant to relative scale factors, fast to evaluate and robust to typical acquisition artifacts.
- We demonstrate the efficiency and robustness of this operator through two practical registration frameworks (assisted or automatic), easy to implement and successfully applied on multi-modal models composed of millions of points at arbitrary scales (see example in Figure 1).

2 Related Work

Three-dimensional geometry registration transforms multiple 3D datasets in a common reference system. The alignment can be obtained using a rigid or non-rigid transformation. Please refer to [6], [7], [8] for surveys covering the main issues and methods related to this task.

In this paper, we focus on the registration of data describing rigid objects. The alignment of geometry data from rigid objects is usually split in two steps: a *rough* alignment, that estimates an initial registration, and a *fine* alignment, which refines the registration starting from the result of the previous step. The *fine* alignment step can be performed

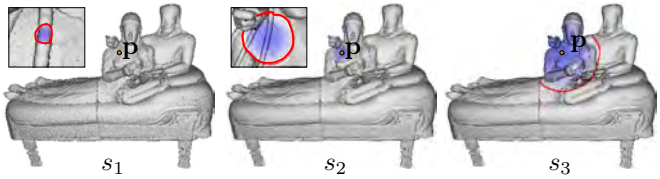


Fig. 2: A point $\mathbf{q}$ on the model $\text{SPOUSE}_{\text{SCAN}}$, which belongs to structures defined at three different scales (s_1, s_2, s_3, shown as red circles): a local bump, the shoulder and the entire torso. Using a multi-scale signature to represent this point and its neighborhood is necessary to get a meaningful description of both local and global shape properties.

in a fully automatic way, and the proposed solutions, eg. instantaneous kinematic [9] and Iterative Closest Point (ICP) with its variants [1], [10], [11], [12], proved to be robust and reliable. In the opposite, automatic *rough* alignment is a more challenging task, especially if no assumptions on the input data are done. From a practical point of view, current software for 3D scanning achieves the alignment of range maps using markers or user intervention.

The goal of our approach is to automatically find a *rough* alignment for multi-modal data, with the possibility to *refine* it in a further step. For this reason, this section will focus on a broad overview of the existing alignement approaches, and stressing their applicability to multi-modal data.

2.1 Point clouds registration

The problem of registering acquired point clouds has been thoroughly studied in the last few years. Local shape descriptors (see Heider et al. [13] for a survey) can be used to match points with similar signatures, and obtain an initial registration, which can be refined using ICP. Some examples of local descriptors include Spin Images [14], feature lines [15], feature histograms [16], and methods based on SIFT and their variants [17], [18]. All of these methods are limited by their locality, so that data with different density or level of detail cannot be treated properly. An interesting alternative would be to use Integral Invariants as multiscale signatures and use them for matching [19]. However such approaches require to build a volumetric representation of the data, which would be challenging because of the noise

1077-2626 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2505287, IEEE Transactions on Visualization and Computer Graphics

and holes potentially present in our data.

Other approaches are based on the correlation of Extended Gaussian Images (EGI) in the Fourier domain [20], or perform the optimization directly on the affine space [21]. Krishnan et al. [22] and Bonarrigo and Signoroni [23] proposed a framework to perform the optimization on the manifold of rotations through an iterative scheme based on Gauss-Newton optimization method. None of the above methods is very robust against noise or outliers.

A very interesting alternative is the 4-Point Congruent Set (4PCS) [24] which explores the transformation space between two clouds by matching almost-planar 4-point quadrilaterals. The proposed scheme can run without shape descriptor, is robust to noise and outliers, and runs with a lower complexity than RANSAC, respectively performing in quadratic versus cubic time according to the number of points. 4PCS can theoretically be extended to estimate a relative scale factor, but in that case the size of the explored congruent set makes the procedure impracticable. The recent Super4PCS variant [25], tailored for rigid transformation estimation, uses efficient indexing techniques to run in linear time in number of point.

2.2 Multi-modal geometry registration

The registration of multi-modal geometric data is usually performed in a semi-automatic fashion. A user provides an initial alignment by manually solving the problem of the initial roto-translation and scaling of the data, or by specifying point-wise correspondences between objects. The analysis of data coming from different devices is applied in methods which aim at automatically align groups of images in a geometry. For instance in the work by Pintus et al. [29], a point cloud generated by structure from motion is used to align a set of images to a different point cloud. Nevertheless, even in this case the initial alignment is estimated using user input. Alternatively, if images were used for geometry generation, they can be analysed to find common features and calculate the 3D registration [30].

The factors described in Section 1 (e.g. scale difference, data density and noise *in primis*) prevent from using most of the aforementioned approaches. An evaluation by Kim and Hilton [31] shows that statistical local descriptors perform poorly in the registration of multi-modal data. An increasing amount of techniques have been proposed recently to address this issue. Lee et al. [32] proposed a method to align 3D scans by analyzing and matching surfaces in the 2D parametric domain. While able to cope with scale differences, the solution is not tested on multi-modal data, and requires parametrized surfaces. Quan and Tang [33] proposed a scale invariant local descriptor applied for 3D part-in-whole matching for mechanical pieces. Unfortunately, the method may be applied only on meshes, and the robustness to noise and to lack of features (i.e. the strong edges of mechanical pieces) is unclear. Rodola' et al. [26] use a game-theoretic framework with scale-independent descriptors in the context of object recognition. While independent of scaling, the method relies on similar geometries, and it can be applied only on continuous surfaces or extremely dense point clouds. A similar approach, that combines local descriptors at different scale to extract and compare the so-called *keyscale* of a model has been recently proposed by Lin et al. [28]. The goal of this method is to estimate the relative scale only, so that ICP can be applied to find the alignment. The proposed descriptor proved to be robust to noise. Nevertheless, it seems to be effective only if a major overlap between the models is present.

In the context of an automatic image registration system, Corsini et al. [27] proposed an automatic method for the alignment of a point-cloud acquired by Structure-from-Motion on another 3D model. The method is based on an extension of 4PCS, and it makes use of the Variational Shape Approximation algorithm [34] to reduce the complexity of the 4PCS when scale has to be estimated and no previous information about overlapping is known. Moreover, this approach requires to reconstruct or at least approximate the input object surface, which can be tedious in some cases. In the context of feature independent methods, a parameters-free framework to fit models to contaminated data was applied also on an example of scale-independent 3D similarity transformation [35].

2.3 Multi-scale geometry analysis

The analysis of 3D objects at multiple scales has been widely studied recently. It has been mainly focused on shape retrieval in 3D databases [36] and matching of deformable objects [37]. Most of the techniques recently presented focus on the intrinsic geometry properties such as the eigenfunctions of the Laplace-Beltrami operator [38], [39], and the Heat Kernel Signatures [40], [41]. Originally limited to meshes, recent approaches have been proposed to extend diffusion and geodesic distances to point clouds [42], [43]. These approaches do not fit our requirement since they cannot automatically detect the scale associated to detected features [44] and thus do not help to estimate the scale between two models. Moreover, they require to solve the diffusion equation globally on the entire object, which makes their use impractical with acquired 3D objects defined by millions of points. An alternative is to evaluate the diffusion on a sub-sampled geometry to speed up computation. However, in this case the details removed from the original model cannot be caught by the multi-scale signatures.

The goal of *scale-space* techniques is to analyze a signal at different scales to discover its geometric structure [45]. A well known use of this theory is the feature detection stage of the Scale-Invariant Feature Transform (SIFT) [46]. These approaches rely on the existence of a parametrization, which is used to compute the spatial derivatives of a signal and extract its relevant structures at multiple scales. Methods have been proposed to adapt SIFT-point detection to 3D by extending image-based techniques either to voxel grids [47], [48], or locally on surfaces using mesh connectivity [49], [50], [51]. In our case we need to work with point-clouds, which makes such techniques unusable without prior remeshing. We refer the reader to the recent work of [44] for a practical and up-to-date comparison of mesh-based *scale-space* techniques. Recently, Mellado et al. have proposed a technique called *Growing Least Squares* (GLS) [5], which aims at extending the *scale-space* formalism to point-set surfaces using implicit kernels evaluated at growing scales. This approach is computed locally at any location on the object, does not require any parametrization, supports arbitrary scale

1077-2626 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2505287, IEEE Transactions on Visualization and Computer Graphics

Differences in ...	BBox fitting	Game-Theoretic framework [26]	SIFT [17], [18]	Corsini et al [27]	Keyscale [28]	Our method
Density and detail amount	V	V?	-	V	V	V
Noise and deformation	V	-	V?	V	V	V
Scalability	V	V	V	V	-	V
Point-clouds	V	-	-	-	V	V
Low overlap	-	V	V	V	V?	V

TABLE 1: Comparison of existing registration techniques regarding their robustness to multi-modal data specificities. In addition to the variations of properties that may occur between the models (e.g. in term of sampling or noise), registration techniques need to handle potentially large point-clouds with unknown overlap.

sampling and can be used to represent and match features using a robust multi-scale geometry descriptor.

2.4 Discussion

The registration of multi-modal data shares most of the main steps with the usual registration of 3D data. During the *rough* registration step, a – potentially large – geometric transformation is computed in order to transform one model and match the other. In case the scale is unknown, as when considering multi-modal data, one need to estimate both the rigid transformation and the scale factor to match the models. For very specific case, eg. partial shape matching with very important overlap, one can estimate the transformation by fitting the two models bounding boxes. In other cases, it becomes necessary to explore the transformation space, preferably by finding correspondences between the models. In real world scenarios, finding these correspondences is a critical step, unlocking at the same time the estimation of the relative scale factor and the rigid transformation. For these reasons we designed our method as a point-wise comparison operator, allowing to estimate a relative scale factor between two points. We argue that when a method is able to find the scale between two models, and can also be used to compute the rigid transformation to align them.

State-of-the-art registration methods may require *ad-hoc* modifications to estimate a relative scale factor and handling multi-modal data specificities, as described in Section 1. We compare in Table 1 the robustness of techniques estimating both scaling and alignment regarding these properties. We added a row on methods *scalability*, in order to emphasis that modern acquisition methods can generate massive amounts of data, which should be taken into account during registration. The "V?" symbol indicates that the method may have the possibility to deal with an issue, but hasn't been tested. From our experiments and original papers conclusions, none of the state-of-the-art method is able to cope with all the issues related to multi-modal data. Our method on the other hand, is able register models from real-world multi-modal datasets, as shown in Section 5.

3 Overview

In this paper we propose a new practical registration technique for 3D multi-modal models, robust to the characteristics described in Section 1. It is designed as follows:

- We use the GLS descriptor to characterize *point-based* data, and use logarithmic scale-space to get scale-invariant signature. This description is robust to *noise*, a realistic amount of *outliers*, and provides a multi-scale point-wise comparison operator robust to *variation of density* and *detail amount*.
- We propose a new approach to compare *point-wise* multi-scale profiles and estimate the *relative scale factor* between points (see Figure 1-a). The resulting operator is easy to implement and fast to evaluate. The amount of relative scaling that can be handled by our approach depends only on the scale-space sampling domain.
- We demonstrate the robustness of our scale estimation to match and estimate scales between points of two input models, and guide global matching techniques (eg. manually assisted and fully automatic RANSAC) to estimate a rigid transformation between them (see RANSAC results in Figure 1-b).

The paper is structured as follow: we first focus on the *point-wise* similarity and scale estimation in Section 4, and then present how we use it to register multimodal data in Section 5. Both sections present related contributions and associated results and evaluations.

4 Point-wise relative scale estimation

This section is organized as follow: we first recall basics concepts related to the Growing Least Squares descriptor, then we present how we extend the GLS comparison operator to estimate a relative scale factor between two points. Then, we evaluate the robustness of our point-wise similarity and scale estimation under variable amount of noise, details, outliers and variations of scales and density.

4.1 Background

The key idea of the GLS approach is to perform a scale-space analysis of point-set surfaces by means of continuous algebraic fits. More specifically, a scale-space is built through least-square fits of an algebraic sphere onto neighborhoods of continuously increasing sizes. The use of an algebraic surface ensures robust fits even at large scales [52] and yields a rich geometric descriptor with only a few parameters, called the *GLS descriptor* [5]. The continuity of the fitting process through scales provides for a stable and elegant analysis of geometric variations, called the *GLS analysis*.

The GLS descriptor is composed of three geometric parameters (illustrated in Figure 4) describing the geometry

1077-2626 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2505287, IEEE Transactions on Visualization and Computer Graphics

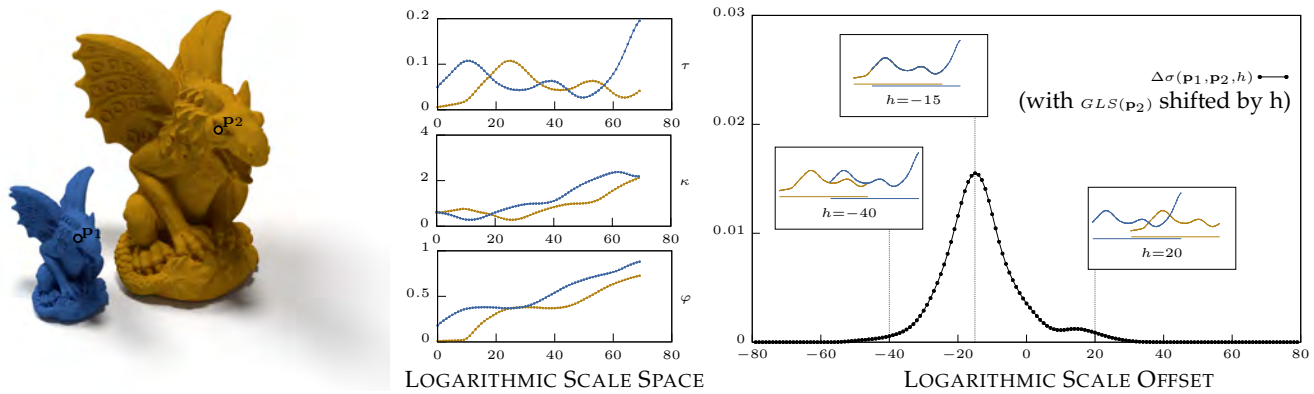


Fig. 3: Overview of our scale estimation. (a) Models $\text{GARG}_{\text{SCALED}}$ (blue) and GARGOYLE (gold). Their relative scale of 2 leads to a translation of the scale-invariant GLS profiles (τ, κ, φ) in logarithmic scale-space (log base$=1.05$), here shown for $\mathbf{p}_1$ and $\mathbf{p}_2$ (b) (colors are consistent). We convolve the two profiles, measure their similarity Δ (see Eq. 2), and extract the relative offset h giving the best score (c), here -15, giving an estimated scale $\frac{1}{1.05^{-15}} = 2.08$ (see Eq. 3).

surrounding an arbitrary location $\mathbf{p}$: the algebraic distance τ_s between $\mathbf{p}$ and the fitted primitive, the unit normal vector $\boldsymbol{\eta}$, and the mean curvature κ_s, where s is the evaluation scale defining the size of the neighborhood used to compute these values. We combine these three parameters with the scale invariant fitness value $\varphi \in [0\ ;\ 1]$, computed as the fitting residuals [52]. Parameters τ_s and κ_s depend on the size of the object and are not reliable to compare multi-modal data. According to [5], one may use their scale-invariant counterparts $\tau = \frac{\tau_s}{s}$, and $\kappa = \kappa_s * s$. Examples of normalized profiles are shown in Figure 3-b). In the following we note $GLS(\mathbf{p}, s) = \left[\tau\ \boldsymbol{\eta}^T\ \kappa\ \varphi\right]^T$ to refer to the scale invariant Growing Least Squares descriptor of point $\mathbf{p}$ at scale s. We refer to the original paper for a more in depth presentation and evaluation of the GLS framework.

According to the original paper, two descriptors computed at different locations $\mathbf{p}, \mathbf{p}'$ and scales s, s' can be compared using the dissimilarity function δ, defined as

$$\delta(\mathbf{p}, s, \mathbf{p}', s') = w_\tau \left(\tau(s) - \tau'(s')\right)^2 + w_\kappa \left(\kappa(s) - \kappa'(s')\right)^2 + w_\varphi \left(\varphi(s) - \varphi'(s')\right)^2 \quad (1)$$

where $\delta = 0$ means a perfect match. GLS descriptors are by definition translation and scale invariants. The rotation invariance is achieved by ignoring the unit normals $\boldsymbol{\eta}$ and $\boldsymbol{\eta}'$. In this paper we used $w_\tau = w_\kappa = w_\varphi = 1$.

This measure can be integrated over a scale domain $[a, b]$ to compare multi-scale profiles, leading to the dissimilarity function

$$\Delta(\mathbf{p}, \mathbf{p}') = \frac{1}{b-a} \int_a^b \delta(\mathbf{p}, s, \mathbf{p}', s)\ ds. \quad (2)$$

Note that the integral is normalized in order to be invariant to the domain size.

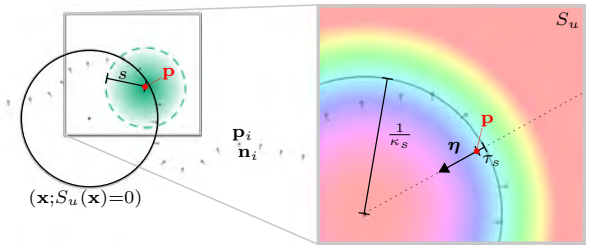


Fig. 4: Geometric meaning of the three geometric components of the GLS descriptor, illustrated in the 2D case: τ_s is the algebraic distance between $\mathbf{p}$ and the fitted primitive, $\boldsymbol{\eta}$ is the unit normal vector, and κ_s the mean curvature, computed at scale s. Figure courtesy Mellado et al. [5].

4.2 Point-wise relative scale estimation and similarity

In the original GLS paper, authors propose to compare discrete multi-scale signatures by summing their dissimilarity Δ over a discrete scale intervals. This approach is valid only when the scale sampling is consistent between the two descriptors, and thus cannot be used directly in our case. As defined in Section 1, multi-modal data can be obtained at arbitrary scales, making point-wise multi-scale signature comparisons challenging. Anyway, we propose in this paper to use the GLS to estimate a relative scale factor between two models.

Scale estimation

According to Bronstein and Kokkinos [41], a scale-space can be constructed to allow scale-invariant multi-scale signature comparison, eg. Scale-Invariant Heat Kernel Signatures (SI-HKS). The key idea is to build a "*logarithmically sampled scale-space in which shape scaling corresponds, up to a multiplicative constant, to a translation*". The goal of SI-HKS is to provide a scale-invariant signature comparison, hence the translation between the two profiles is undone using Fourier analysis and the corrected profiles compared.

One option could be to use the translation between SI-HKS profiles to estimate a relative scale factor between two points. However, in this approach the notion of scale is defined as a diffusion time, and as far as we know, there is no way to convert a difference of diffusion time to an actual distance in Euclidean space.

We propose to solve this issue by computing GLS descriptors in logarithmic scale-space instead of SI-HKS signatures, and use profile translations to estimate a relative

1077-2626 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2505287, IEEE Transactions on Visualization and Computer Graphics

scale factor between two points. Thanks to its scale-invariant formulation and as shown in Figure 3, GLS profiles exhibit the same behavior than SI-HKS signatures when computed on similar objects with different scales: profiles are shifted along the scale dimension. Assuming that the two profiles are computed using a logarithmic scale sampling with basis m, the relative scale s_e between the two models can be retrieved as

$$s_e = \frac{1}{m^h}, \tag{3}$$

where h the translation between two profiles in logarithmic space. An example of shifted descriptors is shown in Figure 3-b).

Scale-invariant similarity

The problem we now need to solve is how to match shifted profiles and compute the associated offset in logarithmic space. When profiles are shifted, one solution is to use the Fourier transform to undo the translation in scale-space and then compute profile similarity, like it is done with SI-HKS. However in our case, the GLS descriptor is a multi-dimensional vector, and our goal is to take into account each of its component to estimate the offset and compute the similarity.

Our approach is illustrated in Figure 3, and allows us to estimate the scale offset between two descriptors and compute their dissimilarity at the same time. To do so, we propose to *convolve* the two sets of profiles. For each convolution step, we compute a similarity measure σ (see Eq. 4), and select the best score configuration. The associated logarithmic shift h can then be used in Equation 3 to estimate the relative scale between the two points. We define the similarity measure σ returning 0 for incompatible descriptors and 1 for a perfect matches as

$$\sigma(\mathbf{p}, s, \mathbf{p}', s') = 1 - \tanh\left(\alpha * \delta(\mathbf{p}, s, \mathbf{p}', s')\right), \tag{4}$$

where tanh is the hyperbolic tangent. The hyperbolic tangent is used to map the values computed by δ from $[0, +\infty]$ to $[0, 1]$, in order to avoid large dissimilarity values and facilitate further processing [5]. The input range of the hyperbolic tangent function is adjusted using α, we used $\alpha = 4$ in all our experiments ($\sigma \approx 0$ when $\delta > 0.5$)

Given this normalized measure, we can now define our discrete scale-invariant similarity and shift estimation operator, as

$$\Delta_\sigma(\mathbf{p}, \mathbf{p}', h) = \frac{1}{2I} \sum_{i=-I}^{I} \sigma(\mathbf{p}, h - i, \mathbf{p}', h), \tag{5}$$

where $2I$ is the length of the overlapping interval between two descriptor for a logarithmic offset h between the two descriptors. Note that we normalize the sum result by the length of the associated interval. This normalization has two purposes: firstly it ensures to produce exactly the same results as the original dissimilarity Δ (see Eq. 2) when $h = 0$. Secondly, it avoids to have a dependency to the interval $2I$, which would naturally favoring solutions with large overlap, i.e. when $h \to 0$. The relative scale between descriptors is then estimated using Equation 3, with h computed as

$$\arg\max_h \ \Delta_\sigma(\mathbf{p}, \mathbf{p}', h). \tag{6}$$

We illustrate in Figure 3 how we compute the best offset by convolving the two descriptors in logarithmic scale-space.

It is important to remark that convolving two GLS descriptors and picking the higher similarity value does not guarantee to estimate the real scale factor between two points, as illustrated in Figure 5-a). However, we observed in practice that this kind of situation can usually be avoided, as shown in next section. In addition, it is usually safer to ignore values of Δ_σ generated from a very small overlap intervals to avoid instabilities and non-representative results. More in general, four parameters are related to the GLS profiles: the logarithmic base, the number of scale samples to estimate, the minimum and the maximum scale to estimate. Setting three of them automatically sets the fourth. Setting the parameters may influence the performance of the descriptor (i.e. the minimum scale can be too big to account for small details), but their relation with the shape of the object is straightforward. A systematic evaluation of the robustness to parameters changes is presented in the next section. Moreover, we observed during our experiments that a set of parameters is enough to cope with very different cases (see Section 5.2), and can also be adjusted manually in an intuitive way (see Section 5.1).

4.3 Evaluation

In this section we evaluate the robustness of the point-wise relative scale estimation technique described in the previous section, against different geometric configurations and artifacts that can be found in multi-modal data. We evaluate our approach by varying the scale-space sampling and measuring its impact on the estimation. We also highlight geometric configurations that could break the scale estimation, and show how they can be detected and thus avoided. We used generic models in this section to make further comparison easier, and we refer the reader to Section 5 for more results on acquired objects.

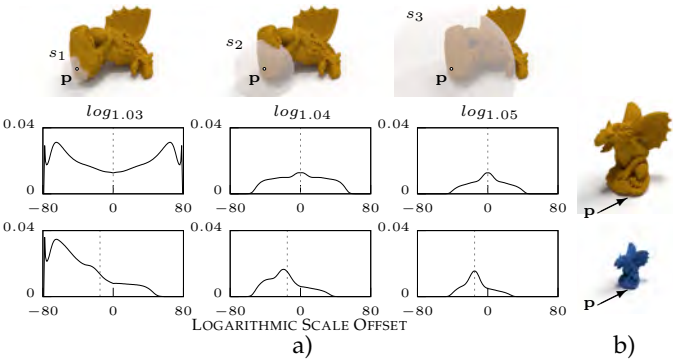


Fig. 5: Point-wise relative scale estimation between scaled versions of the same cloud, for the point $\mathbf{p}$. Top row is without scaling, bottom row with a 0.5 scaling factor. Three different scale samplings are studied, starting from the same minimum scale, but with different log bases, leading to three different scale ranges $[1, s_1]$, $[1, s_2]$, and $[1, s_3]$. The expected positions of the Δ_σ maximum is shown as vertical dashed lines.

1077-2626 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2505287, IEEE Transactions on Visualization and Computer Graphics

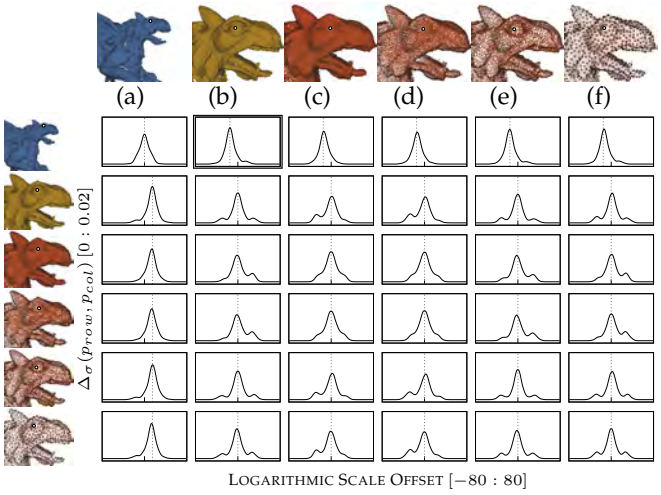


Fig. 6: GLS convolution between models $\text{GARG}_{\text{SCALED}}$ (a), GARGOYLE (b), $\text{GARG}_{\text{B500K}}$ (c), $\text{GARG}_{\text{B120K}}$ (d), $\text{GARG}_{\text{P66K}}$ (e) and $\text{GARG}_{\text{P10K}}$ (f). The selected point is shown as white dot, and selected manually in the eye of the gargoyle. For the sake of clarity, we show the 3 last point-clouds on top of a reference mesh, but only the visible points are used for computation. Expected maxima position are shown as vertical dashed lines, see Table 1 in additional materials for numerical values, maximum error reported: 2 units. See zoom of $\Delta_\sigma(a, b)$ in Figure 3-c).

Evaluation point and scale range

The efficiency of the scale estimation is directly related to the properties of the compared GLS descriptors. The fact that a descriptor could be properly used in Eq. 6 is strongly influenced by the quantity of *significant* geometric configurations surrounding the evaluation point within the studied scale range.

Figure 5 illustrates these conditions. The point $\mathbf{p}$ stands in a part of the object which is locally planar. Hence, computing Eq. 6 fails essentially when the scale range used for computation is too small. When the scale range increases, additional geometric details are added, improving the scale estimation. This is because local, regional and global configurations are combined during the estimation. This also shows that the position of the evaluation points is important. When a manual choice is needed, points exhibiting geometric details at different scales could be preferable. If a candidate point has to be chosen automatically, a proper analysis of its GLS profiles could provide feedback about its quality. Additional comments on this can be found in Section 5.

Noise and spatial sampling

We evaluated the stability of our scale estimation by comparing models with almost the same geometry but with different samplings: GARGOYLE and $\text{GARG}_{\text{SCALED}}$ (see Figure 3), $\text{GARG}_{\text{B500K}}$ and $\text{GARG}_{\text{B120K}}$ (from Berger et al. [53][1]), $\text{GARG}_{\text{P66K}}$ and $\text{GARG}_{\text{P10K}}$ (subsampled versions of GARGOYLE from Berger et al. [54]). The last four models are smoother than first two, while $\text{GARG}_{\text{B120K}}$ contains noise due to simulated acquisition. The evaluation points are also roughly selected by hand on the geometrically meaningful position illustrated in Figure 3-a): the center of the eye. The models do not share the same samples, so the evaluation position may change a bit, as well as the composition of the neighborhood for descriptor computation. Scale-space sampling and range are the same as in Figure 3, with log basis $= 1.05$ and scale range $[1, s_3]$.

Output convolution profiles are shown in Figure 6, and numerical values are available in additional materials. Among all the configurations, error of the estimation are in a range of $[-2, 2]$ in logarithmic scale-space, while the range of possibilities was $[-80, 80]$. The expected positions of the maximum are shown as vertical dashed line in the graphs. Note that $\text{GARG}_{\text{SCALED}}$ is two times smaller than all the other objects, and this factor is well detected by the estimations. Finally, the strength of the maxima visible in all the profiles is a proof of the robustness of our approach, which is not altered by the variation of sampling and lack of information, even for $\text{GARG}_{\text{P10K}}$ which does not contain any feature at fine and medium scales. Regarding the robustness to noise, the procedure used to fit this descriptor from Guennebaud and Gross [52] has been already proven to be robust to noise, and used with success to reconstruct surfaces from noisy point clouds in Berger et al. [53].

Robustness to limited overlapping and outliers

Point-clouds and meshes acquired with different devices are perturbed by different types of noise (eg. frequency, amplitude, pattern), and possibly by outliers. The robustness of our approach is directly related to the robustness of the GLS descriptor. However, this approach is theoretically designed to handle a limited amount of outliers, which can strongly influence the least-square minimization and lead to non-representative GLS descriptors. In practice, we observed that a realistic amount of outliers can be present in the data without perturbing the descriptor computation (see results in Figure 9). Another issue is related to the overlap between two models: if one of the two describes only a portion of the surface, the profiles associated to two corresponding points could present partially different profiles. In this case, only the compatible parts of the descriptor will influence the convolution output.
Given the amount of overlap, the size of the profiles to be compared should be carefully chosen. For example, in an automatic scenario, the profile parameters could be refined after an initial relative scale estimation step.

Spatial smoothness

An important aspect of our approach is that it allows to retrieve the right scale between two points which are not exactly in the same position on the geometry, i.e. the scale estimations vary smoothly on the surface. This spatial smoothness is illustrated in Figure 7, where we show the estimated scales between a point $\mathbf{p}$, and all the points on three scaled version of the BUNNY.

There are two interesting behaviors we would like to emphasis in this example. First, the right scale is estimated on a subpart of the ear of the BUNNY, which mean that we do not need to pick exactly the same positions on the

1. Respectively the reference model and the sample 9

1077-2626 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2505287, IEEE Transactions on Visualization and Computer Graphics

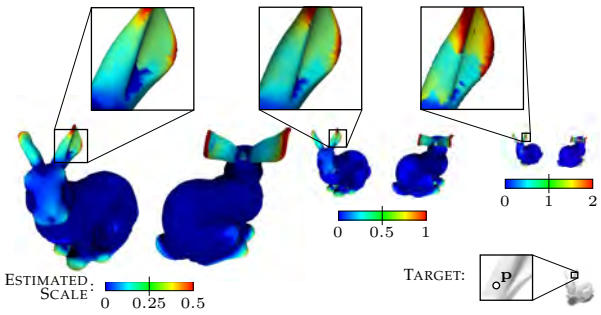


Fig. 7: Color-coded visualization of the estimated scale between all points in 3 different scaled versions of BUNNY, and $\mathbf{p}$. The target scale is always shown in green, under-estimations in blue, and over-estimations in red. Scale-space is sampled with a basis of 1.05, starting from 1 unit, with 100 samples (max. scale $\approx \frac{1}{2}$ bounding box size of the target model).

geometry to get a correct estimation. On the other hand, this stable area is not too big, and wrong scales are estimated on the head or the ear extremity. Second, this behavior is stable over relative scale variations without any change of the scale-space range or sampling. Our semi-automatic matching approach (see Section 5.1) takes advantage of this to allow the user not to be too accurate in selecting correspondences between models, and this is used also by the automatic approach to select starting *seeds* (see Section 5.2). Note that we get similar scale estimations for both ears of the BUNNY, because of the strong symmetry of the model. Ears can however been disambiguated if the scale interval is big enough, using details at large scales. Indeed, the magnitude of the maximum extracted in Eq. 6 encodes the similarity between points, and a bigger value means a better match.

5 MULTI-MODAL REGISTRATION

Registering 3D objects is a common step in the acquisition and digital modeling of physical objects. When dealing with complex real scene or objects, user intervention is most of the time necessary to guide processes. A typical scenario is to let the user setting some correspondences between two models and then align them. Automatic procedures can also be envisaged, especially when the data exhibit comparable size, amount of noise and sampling, and sufficient overlapping. As demonstrated in previous sections, multi-modal data usually break these conditions, and for that reason cannot be registered by such techniques. Another important issue is the increasing number of acquired data today accessible, which requires more and more efficient registration techniques, designed either as simple and fast supervised systems or automatic procedures.

We have implemented our point-wise scale estimation and comparison estimator in two concrete and efficient registration systems, designed to handle point-clouds composed by millions of points and corrupted by acquisition artefacts. To prove the versatility and robustness of our approach, we have used them to register both synthetic and acquired objects, and compared our results to existing techniques on our data. Our systems are designed as follow:

- **A semi-automatic approach** to interactively register models with a simple interaction, and without any pre-processing or subsampling.
- **An automatic approach** based on a standard RANSAC scheme and using our scale estimation to filter invalid configurations.

Both approaches are designed to be available independently, but one could use them processing pipeline where most of the data are processed automatically, and problematic cases fixed by user intervention.

5.1 Semi-automatic registration

In this approach we propose to use minimal user intervention to retrieve the relative scale between two acquired models, at interactive rate, without pre-computation or subsampling. The overall pipeline of our approach is illustrated in Figure 8, and requires two inputs from the user: a scale range, and a pair of roughly corresponding points between the objects. The key idea is to let the user set the parameters that have a strong influence on the process, but are easy to set for a human being. Then, the relative transformation between the two models (scale, rotation and translation) is estimated automatically in interactive time. We would like to emphasize that this approach has been designed as a proof of concept, and requires small implementation efforts. It uses simple heuristics and neither complex optimization nor advanced hardware. Despite this simple design, it has been proven more efficient than existing tools, and applied successfully on real datasets.

Interactive scale estimation and matching

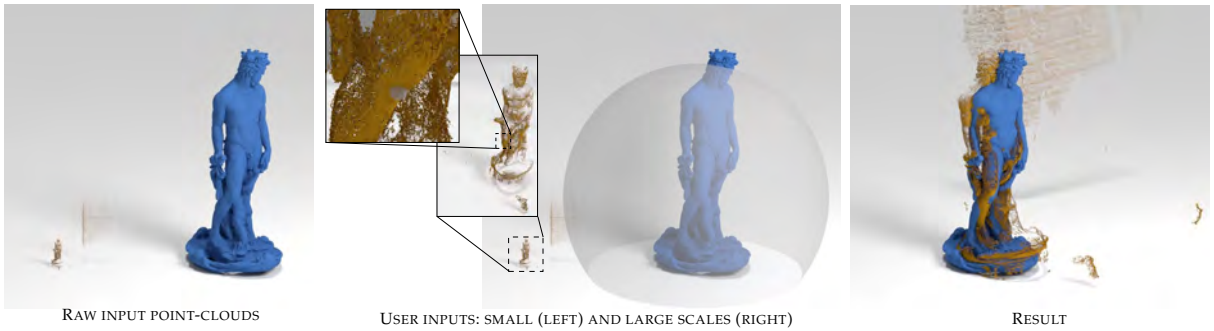
According to Section 4.3, the scale range and sampling parameters could have a strong influence on the accuracy of the scale estimation. We let the user specify them, simply by picking points on the models and adjusting the profiles range by changing the global minimum scale, the logarithmic base and the number of scale samples (the maximum scale is automatically computed from these values). The changes of the parameters is visually shown by indicating the volume (or better to say, the sphere of influence) that will be taken into account for the profile calculation. The user can also take into account the amount of overlap between the two models and adjust the profile to better fit the common information between them.

Then, the user can select a pair of corresponding points and the system interactively finds the relative scale, and align the two models. The selected points can be edited and the result updated. Note that the user does not need to be very accurate in the point selection, thanks to the spatial smoothness of the scale estimation.

According to Section 4.2, we estimate the relative scale between the two selected points as a maximum of Δ_σ, while the translation between both models is found by simply aligning the two corresponding points. The rotation between the two models is retrieved by estimating a local frames for each selected point, and aligning them. The local frames are computed at a given scale as $\left(\boldsymbol{\eta},\quad \mathbf{d_{k_1}},\quad \boldsymbol{\eta} \times \mathbf{d_{k_1}}\right)$ where $\times$ is the cross product between two vectors, and $\mathbf{d_{k_1}}$ the principal curvature direction computed by spatially differentiating the GLS descriptor [55].

1077-2626 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

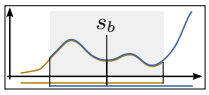
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2505287, IEEE Transactions on Visualization and Computer Graphics



RAW INPUT POINT-CLOUDS USER INPUTS: SMALL (LEFT) AND LARGE SCALES (RIGHT) RESULT

Fig. 8: Overview of our interactive registration tool, which uses a single pair of points specified by the user to estimate both the relative scale and the local frame between two acquired models, here $\text{BIANCONE}_{\text{SCAN}}$ (blue, 2M points) and $\text{BIANCONE}_{\text{PMVS}}$ (gold, 1M points). Neighbourhood collection, descriptor computation, and alignment takes less than 20 seconds (CPU using one single core) without pre-processing.

A key aspect here is to choose the right scale to estimate the local basis: small scales are potentially noisy, and at very large scale (eg. when considering the entire object) the principal curvature direction can be poorly descriptive. In practice, we first estimate the scale between two points, and then use the scale s_b in the middle of the overlapping interval to compute the local basis (see inset below). This heuristic worked for our tests, and can be easily tuned or interactively changed by the user.



Comparisons

We used Patate [55] to compute, match and differentiate the GLS descriptors. Neighborhoods are collected using a KdTree built on the full-resolution cloud, and computations were made using a single core of a Xeon CPU (3.00GHz).

We compared our approach with the manual scale estimation and registration tool available in Meshlab [56] (see attached video). In this system, users have to set multiple pairs of corresponding points between two models, and a PCA is applied to retrieve the similarity matrix between them. Especially when scaling is taken into account, at least 4-5 correspondences pair must be chosen, and they should be well distributed in the context of the overlap between models. This operation can be quite time-consuming, especially when models are noisy or incomplete.

Table 2 shows a comparison of the performances on three datasets from Corsini et al. [27]. Both manual and semi-automatic approaches are able to provide accurate registration. However, our approach permits to complete the operation in a shorter time (including correspondence setting) and requires less accuracy and effort by the user. Note that in the three examples the larger scale used to compute the GLS descriptor includes the whole input clouds. Hence, most of the computation time is in practice spend on neighborhood queries, which could be improved using multi-resolution schemes.

Input Data			Semi-automatic (our)			Manual	
Input	Reference	Target Scale	Nb Scales	Time (sec)	Est. scale	Time (sec)	Est. scale
$\text{GARG}_{\text{SCALED}}$ (1.5M vert.)	$\text{GARG}_{\text{B500K}}$ (500k vert.)	2.0	20	1.07 0.04	2.08	90	2.1
$\text{SPOUSE}_{\text{PMVS}}$ (1.4M vert.)	$\text{SPOUSE}_{\text{SCAN}}$ (4M vert.)	4.25	70	7.8 15.5	4.32	285	4.22
$\text{BIANCONE}_{\text{PMVS}}$ (970k vert.)	$\text{BIANCONE}_{\text{SCAN}}$ (2M vert.)	8.20	100	6.87 6.08	7.83	240	8.13

TABLE 2: Timings of our semi-automatic scale estimation and a manual approach implemented in Meshlab. Our approach requires to compute the GLS descriptor around points on the reference and the input model (respectively first and second line in the Time column). The time to set the points was between 15 and 20 seconds in our experiments (the GLS of the first picked point is computed while choosing the second one), by using a simple trackball and a picking system. The manual approach requires to inspect and set accurately 4-5 pairs of points between the two models. In both cases the time required to estimated the scale is negligible (a few milliseconds).

5.2 Automatic registration

Depending on the application context, it might be necessary to automatically register two point clouds, and avoid user intervention. In that case, automatic registration systems need to explore the transformation space (scaling and rigid transformation). In order to reduce the search space, one may use hints from the models geometry. A standard approach is to detect representative points within the clouds, called *seeds* in the following, and find correspondences between them. In our semi-automatic approach, this critical step is done by the user using context-specific knowledge. A wide range of automatic registration techniques have been proposed in the past to explore the transformation space using points correspondences, e.g. RANSAC [57] and evolutionary game theoretic matching [58]. We choose to design our automatic registration method as a simple RANSAC scheme, and demonstrate its efficiency on real-world dataset. More involved approaches can be derived from this for specific application context.

In order to reduce the amount of points and reduce the computational load, we start by sub-sampling the two clouds with a variant of the Constrained Poisson Disk approach [54]. This approach outputs a homogeneous distribution of samples while preserving the details (we used 200k samples in our experiments). We then pre-compute the GLS descriptors on these clouds. The automatic registration procedure is divided in two steps: first the *seeds* extraction and matching, then the transformation space exploration

1077-2626 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2505287, IEEE Transactions on Visualization and Computer Graphics

using RANSAC.

Seeds selection and matching

In Section 4.3, we showed that our scale estimation is robust to variations of sampling density, noise, and more importantly that it evolves smoothly on objects. This property is really important since it allows us to compute a good relative scaling without requiring *accurate* corresponding points. Even if the corresponding points are not taken in in the exact relative position on the two models, the scale estimation can be accurately performed. Nevertheless, according to [24], using points which are too close one to another during the registration increases the chance to estimate a noisy transformation. Special care must then be taken to avoid having all the *seeds* located at the same place on the model and ensure that they cover as much as possible the important features of the object. We ensure a minimal distance m_d between *seeds* by sampling a second time the models using [54], with a sampling density is defined w.r.t m_d (we used between 2k and 3k points for our experiments). The value of m_d can be calculated by taking into account the size of the bounding box of the object.

The resulting set defines the *seeds* used later to define correspondences. Instead of using a totally random approach, we decided to prioritize the *seeds* in order to increase the speed and accuracy of the registration procedure. We prioritize the *seeds* using the geometric variation ν proposed in [5], and defined as

$$\nu(\mathbf{p}, s) = w_\tau \left(\frac{\delta \tau_s}{\delta s}\right)^2 + w_{\boldsymbol{\eta}} \left(s\frac{\delta \boldsymbol{\eta}}{\delta s}\right)^2 + w_\kappa \left(s^2\frac{\delta \kappa_s}{\delta s}\right)^2.$$

As in Equation 1 we used $w_\tau = w_{\boldsymbol{\eta}} = w_\kappa = 1$. The priority of a *seed* is computed as

$$priority(\mathbf{p}) = \frac{1}{\sum_{s_{min}}^{s_{max}} 1} \sum_{s=s_{min}}^{s_{max}} 1 - tanh(\alpha * \nu(\mathbf{p}, s)) \quad (7)$$

with α defined as in Equation 4. The intuition behind this measure is to prefer points exhibiting variations in their GLS descriptor, producing a more distinctive signature for the scale estimation process.

Then, each *seed* $i_k \in I$ in the input model is matched to its three most similar points in the reference model using Eq. 5 with h computed using Equation 6. Both the estimated scale and the points priorities are attached to the generated pair. A set of three *pairs* is created for each *seed*. We note P the priority queue storing the pairs of points of the reference and input models, with the priority of a pair defined as the product of its points priority.

Finding relative scale and registration

In a second stage, we use the priority queue of *pairs* generated by the previous step to find a the correct relative scale and an initial registration to be refined in a subsequent stage. We use the RANSAC scheme described in Algorithm 1. If no solution is found after a chosen number of iterations, another set of *seeds* I is computed on the input model, and the procedure starts again. For the sake of clarity, we described sub-procedures in Appendix.

Data: PriorityQueue<Pairs >P,Q;
Result: TransformationMatrix M

```
while IterationCount <ItMax do
    triplet_ref = ExtractTriplet(P);
    scale_ok = scaleDiff(triplet_ref) <1 ± e_s ;
    if scale_ok then
        M = ComputeRigidTr(triplet_ref);
        if (RegistrationErr(triplet_ref, M ) <e_p
         AND NormalErr(triplet_ref, M ) <e_n) then
            Q=P ;
            while !Q.isempty() do
                q_ref = ExtractFourthPair(Q.pop());
                if IsValid(triplet_ref,q_ref) then
                    return
                     ComputeRigidTr(triplet_ref,q_ref);
                end
            end
        end
    end
end
return IdentityMatrix
```

Algorithm 1: RANSAC scheme used to explore the transformation space using pairs of *seeds* from the two clouds.

Parameters

All the results in the next section were obtained with a logarithmic base of 1.2. The minimum and maximum scale values used to compute the GLS profile were computed respectively as the average distance between the samples and as the diagonal of the bounding box, for each cloud independently. The other parameters were set as $m_d = 1\%$ of the diagonal of the bounding box, $e_p = 4.0$ units (the reference models were all in millimeters), $e_n = 20$ degrees, and $e_s = 0.2$. All the results were obtained using this single parameter set.

5.3 Comparisons

According to Section 2, while several recent works are facing the issue of scale estimation, only a few of them can actually be applied on real multi-modal data. This is mainly due restrictions on the data representation (triangulated surfaces, 3D scans) [26], [32], [33] or the strong sensitivity to noise. Other techniques couldn't be reproduced [35], since the code was not made available to the community.

First, we compared our approach with the work of Lin and colleagues [28], which is as far as we know the only approach robust enough to handle multimodal data. According to the authors, the approach is quite sensitive to the amount of overlap between the clouds. Nevertheless, we compared our approach to previous work on datasets proposed in [28], to test the accuracy in relative scale estimation among some state-of-the-art techniques. Results of our approach are shown in Figure 9, and quantitative values in Table 3. Our method outperforms the other also before the use of ICP to refine alignment.

Since our method aims at dealing with challenging real datasets, we also tested our approach on models exhibiting strong differences in density, noise, and coverage (results shown in Figure 10 and in the attached video). According to

1077-2626 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2505287, IEEE Transactions on Visualization and Computer Graphics

Dataset	Ground Truth	Standard Deviation	Mesh Resolution [59]	Keyscale [60]	Standard ICP [61]	Scale Ratio ICP [28]	GLS	GLS + ICP [1]
BUNNY	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
Small Blocks (no change)	2.364	4.855 (105.37%)	1.162 (50.85%)	1.400 (40.78%)	3.029 (28.13%)	2.502 (5.84%)	**2.430 (2.81%)**	**2.382 (1.01%)**
Small Blocks (with change)	2.424	3.833 (58.13%)	1.684 (30.53%)	2.250 (7.18%)	2.561 (5.65%)	2.543 (4.91%)	**2.525 (4.16%)**	**2.505 (3.34%)**
Real blocks	1.696	1.593 (6.07%)	1.607 (5.25%)	1.500 (11.56%)	1.767 (4.19%)	1.607 (5.25%)	**1.662 (3.01%)**	**1.674 (2.30%)**

TABLE 3: Comparison of relative scale estimation for several methods, with estimated scale and percentage error. The last two columns show the result of our method with or without ICP. The models and the table (except for the last two columns) are from [28]. Our results are shown in Figure 9.

these results our approach is able to handle complex scenes. Processing time (see Figure 10) is mainly influenced by the complexity of the scene and the amount of overlap.

We observed that our approach tended to fail for data with very low overlap, or when the geometry was scarcely representative (i.e. vast majority of flat surfaces). We would like to emphasize that any other descriptor-based approach may have similar limitations.

6 Conclusion and Future Improvements

We presented a method for the registration and relative scale estimation of multi-modal geometric data. Our method uses a descriptor based on Growing Least Squares (GLS), which is able to characterize both local geometric details and global shape properties on 3D objects. We introduced a new operator to compare two GLS descriptors at arbitrary scales, compute their similarity in scale-space and estimate a relative scale factor between them. Thanks to its robustness, this operator is a good candidate to compare data with variable amount of details, noise, and sampling. In addition, our approach is easy to implement, and fast to evaluate even on point-clouds composed of millions of points. We evaluated it on data corrupted by acquisition artifacts, and shown its stability and relevance for the study of multi-model 3D data.

We built, upon this point-wise scale estimation, two practical frameworks to register multi-modal data either using a user-assisted or an automatic approach. In both cases, we used them to successfully register models with varying outliers, noise, sampling density, and holes. These challenging meshes and point clouds had been generated either by CG artists, or acquired using laser scanners, LIDAR, multi-view stereo and spherical photogrammetry. In all cases our approaches outperformed existing techniques, and in some case they were the only one able to find a solution.

It is important to emphasize that our approach is easy to implement, and can be easily diffused (using open-source library and softwares), reproduced or implemented in existing pipelines. A typical application example is to use it in systems gathering heterogeneous 3D contributions about real objects, and automatically register them in a common reference system and populate a virtual environment.

Thanks to this approach, the registration of 3D multi-modal data can now be easily obtained. However, there are still interesting and challenging questions we would like to study in future work. First, the detections of pertinent structures on acquired point-clouds is still a complex task requiring further study. It is a critical step used in many processing scenarios, like the *seed* selection in our context. In [5], authors present a continuous measure to detect pertinent structures in scale-space. In practice, it is not straightforward to use this measure to drive adaptive sampling techniques and extract pertinent points. A interesting research direction could be to improve such pertinent point extraction, which could significantly improve the performance and the robustness of the registration.

Moreover, this can be crucial also to handle the issue of partial overlapping between models. Regarding this problem, more advanced strategies to refine the profile parameters could lead to a higher robustness of the method.

Finally, another interesting direction is the improvement of both assisted and automatic approaches, either by designing dedicated user-interfaces, or by optimizing the matching algorithms and implement them on GPU.

Appendix

The relative scale estimation and geometric registration procedure shown in Section 5.2 is described step-by-step in this Appendix. The *PriorityQueue* elements are *Pairs*. Each Pair is described by: a point on the reference model i_{ref}, a point on the input model i_{inp}, an associated scale factor s, and a priority value p. The function *ExtractTriplet* extracts three pairs from the PriorityQueue. The probability to be extracted is correlated to the priority value associated to each pair. The first check on the triplet of pairs is the similarity of estimated scale factor (Algorithm 2). If the errors are above the defined threshold (respectively e_p and e_n), the triplet is discarded. Then, the Priority Queue P is copied on another Priority Queue Q, and every point q_{ref} of Q is checked to validate the triplet (Algorithm 5). The fourth pair selected must have a distance bigger then m_d with all the pairs of the reference triplet, and the triplet+fourth pair Registration fulfils the thresholds e_p and e_n. If the returned scale difference is above the e_s threshold, the triplet is discarded. Otherwise, standard PCA algorithm is applied to calculate the Registration Matrix M that best aligns the pairs of the Triplet. Then the Registration Error (Algorithm 3) and the Normals Error (Algorithm 4) are calculated in the standard way. If a fourth pair that validates the Triplet is found, the Registration Matrix obtained from the four pairs is returned as the final result.

Acknowledgments

The research leading to these results was partially funded by EU FP7 project ICT FET Harvest4D (http://www.harvest4d.org/, G.A. no. 323567), ERC Starting Grant SmartGeometry (StG-2013-335373), and ANR

1077-2626 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2505287, IEEE Transactions on Visualization and Computer Graphics

Fig. 9: Dataset used for scale-estimation comparison, from [28].

Data: Triplet *triplet*
Result: Value *scaleDifference*

avgScale=(triplet.s_1+triplet.s_2+triplet.s_3)/3;
scaleDifference=(abs((triplet.s_1-avgScale)) + abs((triplet.s_2-avgScale)) + abs((triplet.s_3-avgScale)))/3;
return *scaleDifference*

Algorithm 2: The *scaleDiff* function

Data: Pairs *pairs*, Registration Matrix M
Result: Value *regError*

regError=(abs($pairs_1.i_{ref}$- M X $pairs_1.i_{inp}$) + abs($pairs_2.i_{ref}$- M X $pairs_2.i_{inp}$) +...+ abs($pairs_{k1}.i_{ref}$- M X $pairs_k.i_{inp}$))/k ;
return *regError*

Algorithm 3: The *RegistrationErr* function

Mapstyle project (ANR-12-COORD-0025). The authors wish to thank Paolo Cignoni, Niloy Mitra and David Vanderhaghe for their support and advices, as well as Pascal Barla, Gael Guennebaud, Aron Monszpart and Moos Hueting for comments and discussions.

Data: Pairs *pairs*, Registration Matrix M
Result: Value *normError*

normError=(*ang*($pairs_1.n_{ref}$,M X $pairs_1.n_{inp}$)+*ang*($pairs_2.n_{ref}$,M X $pairs_2.n_{inp}$)+...+*ang*($pairs_k.n_{ref}$,M X $pairs_k.n_{inp}$))/k ;
return *normError*

Algorithm 4: The *NormalErr* function. n_i is the normal associated to the i-th point of the pair. *ang()* calculates the angle between the two vectors.

Data: Triplet *triplet*, Pair *q*
Result: Bool *IsValid*

if `minDist`(*triplet*,q) $>m_d$ **then**
  return *false*;
end
M_1 = `ComputeRigidTr`(<*triplet*, *q*>);
if `RegistrationErr`(<triplet,q, M_1>) $<e_p$ *AND* `NormalErr`(<triplet,q, M_1>) $<e_n$ **then**
  return *true*;
end
else
  return *false*;
end

Algorithm 5: The *IsValid* function.

REFERENCES

[1] S. Du, N. Zheng, L. Xiong, S. Ying, and J. Xue, "Scaling iterative closest point algorithm for registration of m-d point sets," *J. Vis. Comun. Image Repr.*, vol. 21, no. 5-6, pp. 442–452, Jul. 2010.

[2] K. Parmar and R. Kher, "A comparative analysis of multimodality medical image fusion methods," in *Modelling Symp. (AMS)*, May 2012, pp. 93–97.

1077-2626 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2505287, IEEE Transactions on Visualization and Computer Graphics

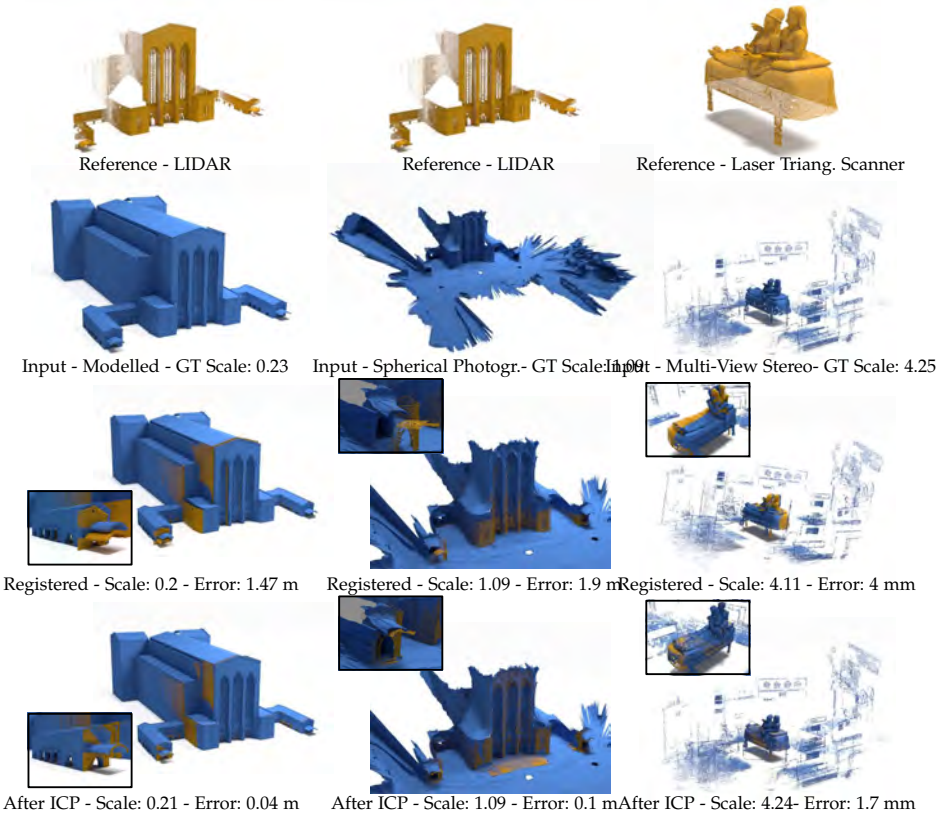


Fig. 10: Results of the method on real datasets. Left Column: LIDAR vs. Modelled, processing time for first registration 170 secs. Middle Column: LIDAR vs. Spherical photogrammetry, 615 secs. Right Column: 3D Scanning vs. Multi-View Stereo, 412 secs.

[3] M. Bhattacharya and A. Das, "Multimodality medical image registration and fusion techniques using mutual information and genetic algorithm-based approaches," in *Software Tools and Algorithms for Biological Systems*, 2011, vol. 696, pp. 441–449.

[4] I. Reducindo, E. Arce-Santana, D. Campos-Delgado, and A. Alba, "Evaluation of multimodal medical image registration based on particle filter," in *Electrical Engineering Computing Science and Automatic Control*, Sept 2010, pp. 406–411.

[5] N. Mellado, G. Guennebaud, P. Barla, P. Reuter, and C. Schlick, "Growing least squares for the analysis of manifolds in scale-space," *Comp. Graph. Forum*, vol. 31, no. 5, Aug. 2012.

[6] J. Salvi, C. Matabosch, D. Fofi, and J. Forest, "A review of recent range image registration methods with accuracy evaluation," *Image Vision Comput.*, vol. 25, no. 5, pp. 578–596, May 2007.

[7] O. van Kaick, H. Zhang, G. Hamarneh, and D. Cohen-Or, "A survey on shape correspondence," *Computer Graphics Forum*, vol. 30, no. 6, pp. 1681–1707, 2011.

[8] G. Tam, Z.-Q. Cheng, Y.-K. Lai, F. Langbein, Y. Liu, D. Marshall, R. Martin, X.-F. Sun, and P. Rosin, "Registration of 3d point clouds and meshes: A survey from rigid to nonrigid," *Visualization and Computer Graphics, IEEE Trans. on*, vol. 19, no. 7, 2013.

[9] H. Pottmann, S. Leopoldseder, and M. Hofer, "Registration without icp," *Comput. Vis. Image Underst.*, vol. 95, no. 1, 2004.

[10] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*, Jun. 2001.

[11] N. J. Mitra, N. Gelfand, H. Pottmann, and L. Guibas, "Registration of point cloud data from a geometric optimization perspective," in *Proc. of the Symp. on Geometry Processing*, 2004.

[12] S. Bouaziz, A. Tagliasacchi, and M. Pauly, "Sparse iterative closest point," *Proc. of the Symp. on Geometry Processing*, vol. 32, no. 5, pp. 1–11, 2013.

[13] P. Heider, A. Pierre-Pierre, R. Li, and C. Grimm, "Local shape descriptors, a survey and evaluation," in *Proc. of EG 3DOR 2011*, Aire-la-Ville, Switzerland, Switzerland, 2011, pp. 49–56.

[14] A. Johnson, "Spin-images: A representation for 3-d surface matching," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, August 1997.

[15] E. Kalogerakis, D. Nowrouzezahrai, P. Simari, and K. Singh, "Extracting lines of curvature from noisy point clouds," *Comput. Aided Des.*, vol. 41, no. 4, pp. 282–292, Apr. 2009.

[16] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *Robotics and Automation, 2009. ICRA'09*. IEEE, 2009, pp. 3212–3217.

[17] X. Li and I. Guskov, "Multi-scale features for approximate alignment of point-based surfaces," in *Proc. of the Symp. on Geometry Processing*. Eurographics Association, 2005.

[18] L. Skelly and S. Sclaroff, "Improved feature descriptors for 3-D surface matching," in *Proc. SPIE Conf. on Two- and 3-Dimensional Methods for Inspection and Metrology*, 2007.

[19] N. Gelfand, N. J. Mitra, L. J. Guibas, and H. Pottmann, "Robust global registration," in *Proc. of the Symp. on Geometry Processing*, 2005.

[20] A. Makadia, A. I. Patterson, and K. Daniilidis, "Fully automatic registration of 3d point clouds," in *Proc. of the CVPR 2006*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 1297–1304.

[21] H. Pottmann, Q.-X. Huang, Y.-L. Yang, and S.-M. Hu, "Geometry and convergence analysis of algorithms for registration of 3d shapes," *Int. J. Comput. Vision*, vol. 67, no. 3, pp. 277–296, 2006.

[22] S. Krishnan, P. Y. Lee, J. B. Moore, and S. Venkatasubramanian, "Global registration of multiple 3d point sets via optimization-on-a-manifold." in *Symposium on Geometry Processing*, 2005, pp. 187–196.

[23] F. Bonarrigo and A. Signoroni, "An enhanced 'optimization-on-a-manifold' framework for global registration of 3d range data," in *Proc. of 3DIM/PVT 2011*. IEEE Computer Society, 2011.

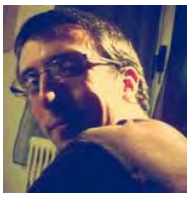
[24] D. Aiger, N. J. Mitra, and D. Cohen-Or, "4-points congruent sets for robust pairwise surface registration," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 85:1–85:10, Aug. 2008.

[25] N. Mellado, D. Aiger, and N. J. Mitra, "Super 4pcs fast global

1077-2626 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2505287, IEEE Transactions on Visualization and Computer Graphics

pointcloud registration via smart indexing," *Computer Graphics Forum*, vol. 33, no. 5, pp. 205–215, 2014.
[26] E. Rodola', A. Albarelli, F. Bergamasco, and A. Torsello, "A scale independent selection process for 3d object recognition in cluttered scenes," *Int. Journal of Computer Vision*, vol. 102, no. 1-3, 2013.
[27] M. Corsini, M. Dellepiane, F. Ganovelli, R. Gherardi, A. Fusiello, and R. Scopigno, "Fully automatic registration of image sets on approximate geometry," *International Journal of Computer Vision*, vol. 102, no. 1-3, pp. 91–111, 2013.
[28] B. Lin, T. Tamaki, F. Zhao, B. Raytchev, K. Kaneda, and K. Ichii, "Scale alignment of 3d point clouds with different scales," *Machine Vision and Applications*, vol. 25, no. 8, pp. 1989–2002, 2014.
[29] R. Pintus, E. Gobbetti, and R. Combet, "Fast and robust semi-automatic registration of photographs to 3d geometry," in *Proc. of VAST 2011*, Aire-la-Ville, Switzerland, Switzerland, 2011, pp. 9–16.
[30] C. Wu, B. Clipp, X. Li, J.-M. Frahm, and M. Pollefeys, "3d model matching with viewpoint-invariant patches (vip)," in *Computer Vision and Pattern Recognition, CVPR 2008*, 2008.
[31] H. Kim and A. Hilton, "Evaluation of 3d feature descriptors for multi-modal data registration," in *Proc. of 3DV, International Conference on 3D Vision*, June 2013.
[32] S. Lee, M. Park, and K. Lee, "Full 3d surface reconstruction of partial scan data with noise and different levels of scale," *Journal of Mechanical Science and Technology*, vol. 28, no. 8, 2014.
[33] L. Quan and K. Tang, "Polynomial local shape descriptor on interest points for 3d part-in-whole matching," *Computer-Aided Design*, vol. 59, no. 0, 2015.
[34] D. Cohen-Steiner, P. Alliez, and M. Desbrun, "Variational shape approximation," *ACM Tr. on Graph.*, vol. 23, pp. 905–914, Aug. 2004.
[35] R. Raguram and J.-M. Frahm, "Recon: Scale-adaptive robust estimation via residual consensus," *Computer Vision, IEEE International Conference on*, vol. 0, pp. 1299–1306, 2011.
[36] C. Wang, M. M. Bronstein, A. M. Bronstein, and N. Paragios, "Discrete minimum distortion correspondence problems for non-rigid shape matching," in *Proc. of SSVM*, Berlin, 2012.
[37] D. Raviv, A. M. Bronstein, M. M. Bronstein, R. Kimmel, and N. Sochen, "Affine-invariant geodesic geometry of deformable 3d shapes," *Computers & Graphics*, vol. 35, no. 3, 2011.
[38] R. M. Rustamov, "Laplace-beltrami eigenfunctions for deformation invariant shape representation," in *Proc. of the Symp. on Geometry Processing*, 2007, pp. 225–233.
[39] A. Dubrovina and R. Kimmel, "Matching shapes by eigendecomposition of the laplace-beltrami operator," in *Proc. 3DPVT*, vol. 2, no. 3, 2010.
[40] J. Sun, M. Ovsjanikov, and L. Guibas, "A concise and provably informative multi-scale signature based on heat diffusion," in *Proc. of the Symp. on Geometry Processing*, 2009.
[41] M. M. Bronstein and I. Kokkinos, "Scale-invariant heat kernel signatures for non-rigid shape recognition." in *CVPR*, 2010, pp. 1704–1711.
[42] K. Crane, C. Weischedel, and M. Wardetzky, "Geodesics in heat: A new approach to computing distance based on heat flow," *ACM Trans. Graph.*, vol. 32, no. 5, pp. 152:1–152:11, Oct. 2013.
[43] G. Patané and M. Spagnuolo, "Heat diffusion kernel and distance on surface meshes and point sets," *Computers & Graphics*, vol. 37, no. 6, 2013.
[44] H. Fadaifard, G. Wolberg, and R. Haralick, "Multiscale 3d feature extraction and matching with an application to 3d face recognition," *Graphical Models*, vol. 75, no. 4, pp. 157 – 176, 2013.
[45] B. Romeny, *Front-End Vision and Multi-Scale Image Analysis: Multi-scale Computer Vision Theory and Applications, written in Mathematica*, 1st ed. Springer Publishing, 2009.
[46] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. of the International Conference on Computer Vision*. Washington, DC, USA: IEEE Computer Society, 1999.
[47] P. Scovanner, S. Ali, and M. Shah, "A 3-dimensional sift descriptor and its application to action recognition," in *Proc. of the 15th International Conference on Multimedia*, 2007, pp. 357–360.
[48] G. Flitton, T. Breckon, and N. Megherbi Bouallagu, "Object recognition using 3d sift in complex ct volumes," in *Proc. of BMVS 2010*, 2010, pp. 11.1–11.12.
[49] A. Zaharescu, E. Boyer, K. Varanasi, and R. Horaud, "Surface feature detection and description with applications to mesh matching," in *CVPR 2009*, 2009.
[50] C. Maes, T. Fabry, J. Keustermans, D. Smeets, P. Suetens, and D. Vandermeulen, "Feature detection on 3d face surfaces for pose normalisation and recognition," in *Biometrics: Theory Applications and Systems, 2010.*, 2010.
[51] T. Darom and Y. Keller, "Scale-invariant features for 3-d mesh models," *IEEE Trans. on Image Processing*, vol. 21, 2012.
[52] G. Guennebaud and M. Gross, "Algebraic point set surfaces," *ACM Trans. Graph.*, vol. 26, no. 3, Jul. 2007.
[53] M. Berger, J. A. Levine, L. G. Nonato, G. Taubin, and C. T. Silva, "A benchmark for surface reconstruction," *ACM Trans. Graph.*, vol. 32, no. 2, pp. 20:1–20:17, Apr. 2013.
[54] M. Corsini, P. Cignoni, and R. Scopigno, "Efficient and flexible sampling with blue noise properties of triangular meshes," *Trans. on Visualization and Computer Graphics*, vol. 18, no. 6, 2012.
[55] N. Mellado, G. Ciaudo, G. Guennebaud, and P. Barla, "Patate lib," http://patate.gforge.inria.fr/, 2013.
[56] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "Meshlab: an open-source mesh processing tool," in *Sixth Eurographics Italian Chapter Conference*, 2008, p. 129.
[57] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.
[58] A. Albarelli, E. Rodola, and A. Torsello, "Loosely distinctive features for robust surface alignment," in *Computer Vision–ECCV 2010*. Springer, 2010, pp. 519–532.
[59] A. Johnson and M. Hebert, "Using spin images for efficient object recognition in cluttered 3d scenes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 21, no. 5, May 1999.
[60] T. Tamaki, S. Tanigawa, Y. Ueno, B. Raytchev, and K. Kaneda, "Scale matching of 3d point clouds by finding keyscales with spin images," in *International Conference on Pattern Recognition*, 2010.
[61] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 14, no. 2, pp. 239–256, Feb 1992.



Nicolas Mellado Nicolas Mellado is a post-doctoral researcher at IRIT, University of Toulouse. His research interests include point cloud processing, multiscale analysis and registration. Dr. Mellado received a PhD degree in Computer Science from the University of Bordeaux, France, December 2012.

Matteo Dellepiane is a Researcher at CNR-ISTI. He received an advanced degree in Telecommunication Engineering (Laurea) from the University of Genova in 2002, and a PhD in Information Engineering from the University of Pisa in 2009. His research interests include 3D scanning, digital archeology, color acquisition and visualization on 3D models and web visualization of 3D models.

Roberto Scopigno is a Research Director with CNR-ISTI and leads the Visual Computing Lab. He graduated in Computer Science at the University of Pisa in 1984. He is engaged in research projects concerned with 3D scanning, surface reconstruction, multiresolution, scientific visualization, and cultural heritage. He's currently Editor-in-chief of the ACM J. on Computing and Cultural Heritage.

1077-2626 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

EUROGRAPHICS 2016 / J. Jorge and M. Lin
(Guest Editors)

Volume 35 (2016), Number 2

Geometry and Attribute Compression for Voxel Scenes

Bas Dado, Timothy R. Kol[†], Pablo Bauszat[‡], Jean-Marc Thiery and Elmar Eisemann[§]

Delft University of Technology

Figure 1: *Compressed voxelized scene at different levels of detail, rendered in real time using raytracing only. Our hierarchy encodes geometry and quantized colors at a resolution of* $128K^3$. *Despite containing* 18.4 *billion colored nodes, it is stored entirely on the GPU, requiring* 7.63 *GB of memory using our compression schemes. Only at the scale shown in the right bottom image the voxels become apparent.*

Abstract

Voxel-based approaches are today's standard to encode volume data. Recently, directed acyclic graphs (DAGs) were successfully used for compressing sparse voxel scenes as well, but they are restricted to a single bit of (geometry) information per voxel. We present a method to compress arbitrary *data, such as colors, normals, or reflectance information. By decoupling geometry and voxel data via a novel mapping scheme, we are able to apply the DAG principle to encode the topology, while using a palette-based compression for the voxel attributes, leading to a drastic memory reduction. Our method outperforms existing state-of-the-art techniques and is well-suited for GPU architectures. We achieve real-time performance on commodity hardware for colored scenes with up to* 17 *hierarchical levels (a* $128K^3$ *voxel resolution), which are stored fully in core.*

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—Raytracing

1. Introduction

With the increase of complexity in virtual scenes, alternative representations, which enable small-scale details and efficient advanced lighting, have received a renewed interest in computer graphics [LK10]. Voxel-based approaches encode scenes in a high-resolution grid. While they can represent complicated structures, the memory cost grows quickly. Fortunately, most scenes are sparse – i.e., many voxels are empty. For instance, Fig. 1 shows a scene represented by a voxel grid of 2.25 quadrillion voxels ($128K^3$), but 99.999% are actually empty. Although hierarchical representations like sparse voxel octrees (SVOs) [JT80, Mea82] exploit this sparsity, they can only be moderately successful; a large volume like the one in Fig. 1 still contains over 18 billion filled voxels.

For large volumes, specialized out-of-core techniques and compression mechanisms have been proposed, which often result in additional performance costs [BRGIG*14]. Only recently, directed

† t.r.kol@tudelft.nl
‡ p.bauszat@tudelft.nl
§ e.eisemann@tudelft.nl

© 2016 The Author(s)
Computer Graphics Forum © 2016 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.

acyclic graphs (DAGs) have shown that even large-scale scenes can be kept entirely in memory while being efficiently traversable. They achieve high compression rates of an SVO representation with a single bit of information per leaf node [KSA13]. Their key insight is to merge equal subtrees, which is particularly successful if scenes exhibit geometric repetition. Unfortunately, extending the information beyond one bit (e.g., to store material properties) is challenging, as it reduces the amount of similar subtrees drastically.

Our contribution is to associate attributes to the DAG representation, which are compressed separately while maintaining efficiency in rendering tasks. To this extent, we introduce a decoupling of voxel attributes from the topology and a subsequent compression of these attributes. Hereby, we can profit from the full DAG compression scheme for the geometry and handle attributes separately. Although the compression gain is significant, the representation can still be efficiently queried. In practice, our approach enables real-time rendering of colored voxel scenes with a $128K^3$ resolution in full HD on commodity hardware while keeping all data in core. Additionally, attributes like normals or reflectance can be encoded, enabling complex visual effects (e.g., specular reflections).

Our main contributions are the decoupling of geometry and voxel data, as well as the palette compression of quantized attributes, delivering drastic memory gains and ensuring efficient rendering. Using our standard settings, high-resolution colored scenes as in Fig. 1 require on average well below one byte per voxel.

2. Related work

We only focus on the most related methods and refer to a recent survey by Balsa Rodríguez et al. [BRGIG*14] for other compression techniques, particularly for GPU-based volume rendering.

Large datasets can be handled via streaming; recent approaches adapt a reduced representation on the GPU by taking the ray traversals through the voxel grid into account [GMIG08, CNLE09, CNSE10]. Nonetheless, transfer and potential disk access make these methods less suited for high-performance applications. Here, it is advantageous to keep a full representation in GPU memory, for which a compact data structure is of high importance.

Dense volume compression has received wide attention in several areas – e.g., in medical visualization [GWGS02]. These solutions mostly exploit local coherence in the data. We also rely on this insight for attribute compression, but existing solutions are less suitable for sparse environments. In this context, besides SVOs [JT80, Mea82], perfect spatial hashing can render a sparse volume compact by means of dense hash and offset tables [LH06]. While these methods support efficient random access, exploiting only sparsity is insufficient to compress high-resolution scenes.

Efficient sparse voxel octrees (ESVOs) observe that scene geometry can generally be represented well using a contour encoding [LK11]. Using contours allows early culling of the tree structure if the contour fits the original geometry well, but this can limit the attribute resolution (e.g., color). While it is possible to reduce the use of contours in selected areas, this choice also impacts the compression effectiveness drastically. Voxel attributes are compressed using a block-based DXT scheme, requiring one byte for colors and two bytes for normals per voxel on average. For high-resolution scenes, a streaming mechanism is presented.

Recently, Kämpe et al. observed that besides sparsity, geometric redundancy in voxel scenes is common. They proposed to merge equal subtrees in an SVO, resulting in a directed acyclic graph (DAG) [KSA13]. The compression rates are significant and the method was even used for shadow mapping [SKOA14, KSA15]. Nonetheless, the employed pointers to encode the structure of the DAG can become a critical bottleneck. Pointerless SVOs (PSVOs) [SK06] completely remove pointer overhead and are well-suited for offline storage. However, they do not support random access and cannot be extended to DAGs, as PSVOs require a fixed, sequential memory layout of nodes. While several reduction techniques for pointers have been proposed [LK11,LH07], they are typically not applicable to the DAG. These methods assume that pointers can be replaced by small offsets, but in a DAG, a node's children are not in order but scattered over different subtrees. Concurrent work presented a pointer entropy encoding and symmetry-based compression for DAGs, but does not support attributes [JMG16].

Adding voxel data reduces the probability of equal subtrees, making DAGs unsuitable for colored scenes. The recently proposed Moxel DAGs [Wil15] address this problem. In every node, they store the number of empty leaf voxels (assuming a complete grid) in the first child's subtree. During traversal, two running sums are kept – the number of empty leaves and total leaves – to compute a sequential unique index for every existing leaf voxel, with which the corresponding attributes are retrieved from a dense but uncompressed array. Our method is more efficient (with only one running sum) and requires less memory, as the number of empty leaf voxels grows to quadrillions for scenes like in Fig. 1, leading to large storage requirements for the additional index per node. Furthermore, Moxel DAGs do not encode a multi-resolution representation and, hence, cannot directly be used for level-of-detail rendering.

Uncompressed voxel attributes quickly become infeasible for higher resolutions, especially on GPU architectures where memory is limited. Here, attribute compression can be used. Specialized algorithms exist for textures [SAM05, NLP*12], colors (via effective quantization [Xia97]) or normals (octahedron normal vectors (ONVs) [MSS*10]). For the latter, careful quantization is necessary [CDE*14]. We decouple the geometry of a voxel scene from its attributes, which enables exploring such compression schemes.

3. Background

A voxel scene is a cubical 3D grid of resolution 2^{N^3} with N a positive integer. Each voxel is either empty or contains some information, such as a bit indicating presence of matter, or multiple bits for normal or material data. SVOs encode these grids by grouping empty regions; each node stores an 8-bit mask denoting for every child if it exists – i.e., is not empty. A pointer connects the parent to its children, which are ordered in memory. Thus, 8 bits are needed for the childmask, plus a pointer of typically 32 bits. Furthermore, for level-of-detail rendering, parent nodes usually contain a representation of the children's data (e.g., an average color). If only geometry is encoded, the childmask gives sufficient information and no data entries are needed. Note that literature typically considers

© 2016 The Author(s)
Computer Graphics Forum © 2016 The Eurographics Association and John Wiley & Sons Ltd.

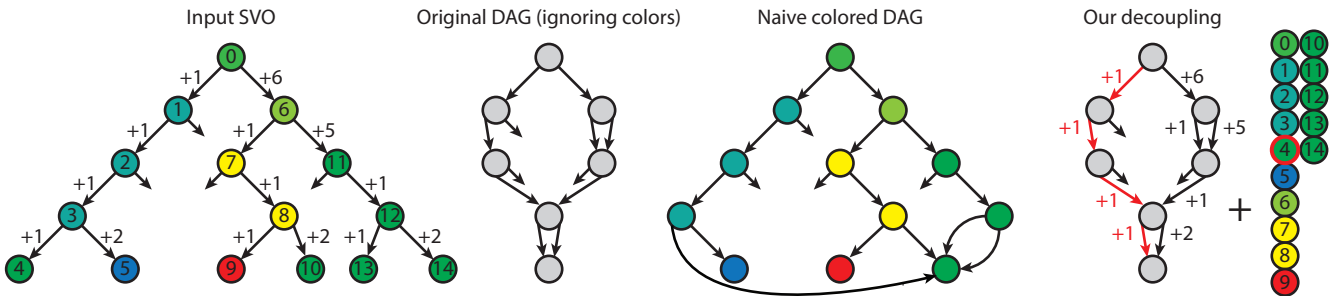


Figure 2: *The input to our approach is an SVO with data (left). DAGs are only efficient when storing the topology (middle-left); when considering attributes, merging fails to compress the SVO sufficiently (middle-right). Our approach decouples data (colors in this case) from topology by storing offsets in the pointers, enabling us to apply the DAG principle on the geometry (right). The offsets then allow access to an attribute array, which is compressed independently. The red descent shows how the accumulated offsets deliver the correct array element.*

SVO nodes that are not leaves as voxels as well, so that reported voxel counts equal the number of tree nodes.

The DAG algorithm [KSA13] is an elegant method to exploit redundancy in a geometry SVO and forms the basis of our topology encoding. For ease of illustration, Fig. 2 uses a binary tree, but the extension to more children is straightforward. On the left, a sparse, colored, binary tree is shown. Dangling pointers refer to empty child nodes without geometry. We ignore the colors and numbers for now and only focus on the topology. The DAG is constructed in a greedy bottom-up fashion. Starting with the leaves at the lowest level, subtrees are compared and, if identical, merged by changing the parent pointers to reference a single common subtree. The DAG contains significantly fewer nodes than the SVO (Fig. 2, middle-left). Note that for a DAG as well as an SVO, leaf nodes do not require pointers, and, when encoding geometry only, the leaves can even be stored implicitly by using the parent childmask.

One disadvantage of the DAG in comparison to an SVO is that pointers need to be stored for *each* child, because they can no longer be grouped consecutively in memory (in which case, a single pointer to the first child is sufficient). In practice, the 40 bits per node in a geometry SVO (8-bit childmask and a 32-bit pointer), become around $8 + 4 \times 32 = 136$ bits in a DAG – assuming a node has four children on average, e.g., for a voxelized surface mesh. The high gain of the DAG stems from the compression at low levels in the tree. For example, an SVO with 17 hierarchical levels usually has billions of nodes on the second-lowest level while a DAG has at most 256 – the amount of possible unique combinations of eight child voxels having each one bit. For higher levels, the number of combinations increases, which reduces the amount of possible merging operations; this also reflects the difficulty that arises when trying to merge nodes containing attribute data. With only three different data elements (colors of leaves), the merging process already stops after the lowest level (Fig. 2, middle-right).

4. Our approach

The possibility of merging subtrees is reduced when voxel attributes such as normals and colors are used. While the data usually exhibits some spatial coherence, exploiting it with a DAG is difficult because the attributes are tightly linked to the SVO's topology. We propose a novel mapping scheme that decouples the voxel geometry from its additional data, enabling us to perform specialized compression for geometry and attributes separately, which greatly amortizes the theoretical overhead caused by the decoupling.

Using our decoupling mechanism, which is described in Sec. 4.1, the geometry can be encoded using a DAG. The extracted attributes are stored in a dense *attribute array*, which is subsequently compressed. During DAG traversal, the node's attributes can efficiently be retrieved from the array. The attribute array itself is processed via a palette-based compression scheme, which is presented in Sec. 4.2. It is based on the key insight that the array often contains large blocks of similar attributes due to the spatial coherence of the data (e.g., a large meadow containing only a few shades of green). In consequence, using a local palette, the indices into this palette require much less memory than the original attributes.

While the original design for the palette compression is lossless, we show in Sec. 4.3 that compression performance can be significantly improved by quantizing attributes beforehand. Hereby, a trade-off between quality and memory reduction is possible, which can be steered depending on the application. We demonstrate that significant compression improvements can already be achieved by using perceptually almost indistinguishable quantization levels.

Finally, we show in Sec. 4.4 that the DAG itself can also be further compressed using pointer and offset compression, as well as an entropy-based pointer encoding, which is a valuable addition to the original DAG method as well. These techniques greatly amortize the additional storage required for the decoupling.

4.1. Voxel attribute decoupling

To decouple data from geometry, we first virtually assign *indices* to all nodes in the initial SVO in depth-first order (Fig. 2, left, the numbers inside the nodes). Next, for every pointer, we consider an *offset* (Fig. 2, left, the positive numbers next to the edges), which equals the difference between the index of the child and parent associated with this pointer. Summing all offsets along a path from the root to a node then reproduces its original index.

© 2016 The Author(s)
Computer Graphics Forum © 2016 The Eurographics Association and John Wiley & Sons Ltd.

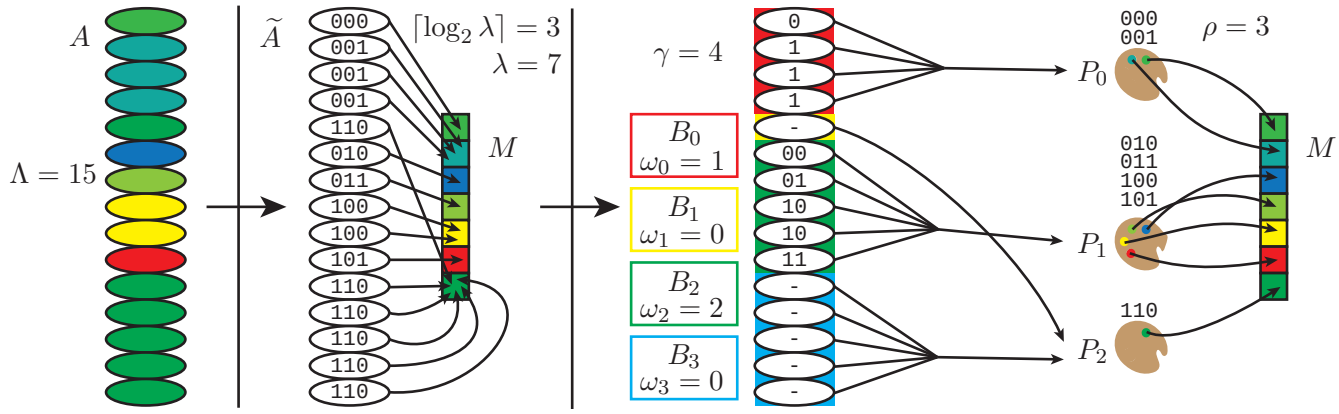


Figure 3: *Palette compression. From left to right: the initial attribute array $A = \{a_0, ..., a_{14}\}$ stores 24-bit colors; we construct the material array $M = \{\widetilde{a}_0, ..., \widetilde{a}_6\}$ to store the 24-bit colors while $\widetilde{A}$ contains 3-bit indices into M; four blocks $\{B_0, B_1, B_2, B_3\}$ are created, containing 0-bit to 2-bit palette indices into the three associated palettes $\{P_0, P_1, P_2\}$, which in turn contain 3-bit material indices into M.*

Based on this insight, we propose to store these offsets together with each child pointer and to extract and store the node attributes in a dense *attribute array* in the same depth-first order (Fig. 2, right, the stacked colors). During traversal from the root, a node's index is reconstructed via these offsets. This index can then be used to efficiently retrieve the corresponding voxel attribute from the array.

While our mapping introduces an overhead in the form of an additional offset for every child pointer, it has the benefit that subtrees with identical topology can be merged to a DAG again. In fact, a depth-first indexing automatically leads to identical offsets in geometrically identical subtrees. Further, we show in Sec. 5.3 that these offsets can be compressed very efficiently. Fig. 2, right, illustrates an exemplary index retrieval in the DAG-compressed tree for the node with index 4, where the red arrows denote the tree descent.

4.2. Palette compression

After decoupling and storing the geometry in a DAG, we are left with an efficient representation of the topology, but the uncompressed attribute array still requires a large amount of memory. We propose a variable-length compression scheme for the attribute array, which is efficient and still allows for fast accessing at run-time. To explain our method, we first describe the use of a global material array, making it possible to store indices instead of full attributes. Because of spatial coherence in the scene, consecutive indices will often be similar, which leads to the idea of working on blocks of entries in the attribute array. For each block, we define a palette (local index array) and each entry in a block only stores a local index into this palette. The palette then allows us to access the correct entry in the global material array.

Specifically, our approach works as follows. We denote the attribute array as $A = \{a_0, ..., a_{\Lambda-1}\}$, where Λ is the total number of entries. Note that Λ equals the voxel count in the original SVO. We observe that A usually contains many duplicates and the number of unique voxel attributes λ is typically orders of magnitude smaller than Λ. For this reason, a first improvement is to construct a material array $M = \{\widetilde{a}_0, ..., \widetilde{a}_{\lambda-1}\}$, which stores all λ unique attributes in the scene, and replace A with an indexed version pointing into M. We denote the index array as $\widetilde{A} = \{m_0, ..., m_{\Lambda-1}\}$, where m denotes an index into M. Since indices require fewer bits than attributes, it usually results in a reduced memory footprint and decouples the content of the material array from the attribute array. An example is provided in Fig. 3.

Since the data in A is ordered depth-first, we retain most of the spatial coherence of the original scene. Consequently, if a large area exhibits a limited set of attributes (e.g., a blue lake represented by millions of blue voxels with little variation) they are likely to be consecutive in A. Hence, it would be beneficial to partition the attribute array into multiple *blocks* of consecutive entries, where each only contains a small number of different indices. We describe how to determine these blocks later.

Each block has an associated *palette*, which is an array of the necessary unique indices into the material array to retrieve all attributes in the block. The block itself only stores (possibly repeating) indices into its associated palette. While each index in a block originally requires $\lceil \log_2 \lambda \rceil$ bits, it is now replaced by a new index with only ω bits, where ω depends solely on the number of unique entries inside the block. Note that there is no one-to-one correspondence between palettes and blocks; a palette can be shared by several blocks, but each block is linked to a single palette only.

Blocks have a variable length, which makes it necessary to keep a block directory to indicate where blocks start and what their corresponding palette is. The block directory has its entries ordered by the starting node index, which makes it possible to perform a binary search to find the corresponding block information given a node index. Generally, the memory overhead of the directory is negligible.

Our representation ultimately consists of an array of blocks $\{B_0, ..., B_{\gamma-1}\}$ and an array of palettes $\{P_0, ..., P_{\rho-1}\}$, where γ and ρ denote the total number of blocks and palettes, respectively. For the example in Fig. 3, it can be seen that we obtain three palettes

© 2016 The Author(s)
Computer Graphics Forum © 2016 The Eurographics Association and John Wiley & Sons Ltd.

and four blocks (i.e., $\rho = 3, \gamma = 4$), because B_1 and B_3 use an identical palette that does not have to be stored twice.

Algorithm 1 Palette compression

```
1:  function FINDLARGEBLOCKS({m_i,...,m_j})
2:      if j < i then return
3:      ω ← 0
4:      while ω < 4 do
5:          {m_k,...,m_l} ← largest block with 2^ω unique m
6:          B ← {m_k,...,m_l}
7:          if MEMORY(B,ω) < (l−k+1)·(ω+1) then
8:              P ← CREATEPALETTE(B)
9:              for all m ∈ B do m ← index into P
10:             FINDLARGEBLOCKS({m_i,...,m_{k−1}})
11:             FINDLARGEBLOCKS({m_{l+1},...,m_j})
12:             return
13:         else
14:             ω ← ω+1
15:     FINDREMAININGBLOCKS({m_i,...,m_j})
16: function FINDREMAININGBLOCKS({m_i,...,m_j})
17:     if j < i then return
18:     ω ← {0,...,8}
19:     for all ω do
20:         {m_i,...,m_{k_ω}} ← largest block with 2^ω unique m from m_i
21:         B_ω ← {m_i,...,m_{k_ω}}
22:         S_ω ← MEMORY(B_ω,ω) /(k_ω − i+1)
23:     B,k ← B_ω,k_ω with minimal S_ω
24:     P ← CREATEPALETTE(B)
25:     for all m ∈ B do m ← index into P
26:     FINDREMAININGBLOCKS({m_{k+1},...,m_j})
27: function MEMORY({m_i,...,m_j},ω)
28:     return (j−i+1)·ω+2^ω·⌈log_2 λ⌉+size(directory entry)
```

Palette selection Finding the optimal set of blocks with respect to their memory requirement is a hard combinatorial problem, and the attribute array contains billions of entries for high-resolution scenes. Hence, we propose a greedy heuristic to approximate the optimal block partitioning.

The algorithm consists of two phases (see Alg. 1). First, we greedily find the largest blocks that only require a few bits per entry, as these blocks form the best opportunities for high compression rates. This first phase takes a consecutive subset of $\widetilde{A}$ as its parameter, and is initially invoked for the complete array ($\{m_i,...,m_j\}$ with $i = 0$ and $j = \Lambda - 1$). It finds the largest block that appears in this set consisting of 2^ω unique material indices in a brute-force fashion (line 5). Since we start with $\omega = 0$ (line 3), it first finds the largest consecutive block with only one unique index. If the total overhead introduced by creating a palette is outweighed by the memory reduction (line 7), we generate a palette (if we could not find an existing matching palette) and replace the material indices m with indices into this palette (lines 8 and 9). The remainder of $\widetilde{A}$ is then processed recursively (lines 10 and 11). If the criterion is not satisfied, we increment ω and repeat (line 14). When ω becomes too large, we stop the first phase, as finding the largest block becomes computationally infeasible. In our case, we terminate for $\omega \geq 4$, corresponding to 16 unique indices or more (line 4).

The second phase is invoked for the data that could not be assigned to blocks in phase one (line 15) which is now partitioned into blocks sequentially. For this, nine possible blocks (for each $\omega = \{0,...,8\}$) are considered, all starting at m_i (line 20). Of these nine blocks, the one with the minimal memory per entry (including the directory overhead) is used (line 23), and a palette is attributed to this block, after which we replace the indices again (lines 24 and 25). This is repeated for the remaining data (line 26). To compute a block's memory overhead (line 28), we multiply the block entries by the bits required for a palette index ($(j-i+1)\cdot\omega$) and add the palette entries multiplied by the bits required for a material index ($2^\omega \cdot \lceil \log_2 \lambda \rceil$). Finally, we add the block's directory entry overhead.

For the example in Fig. 3, only B_3 is created in phase one, as other possible blocks do not satisfy the memory criterion (line 7). The remaining data is processed in phase two, which results in three additional palettes, one of which can be shared.

4.3. Attribute quantization

The palette-based compression scheme for the attribute array is lossless and can already provide a significant reduction in memory. However, since human perception is not as flawless as a computer's, and many scenes exhibit similarity in voxel attributes, we can apply a certain degree of quantization on many kinds of attributes without losing much visual quality. This can greatly improve the compression capability of our proposed approach.

In principle, any standard quantization could be applied to the attribute array, but specializing the method based on the data type leads to improved results. In particular, we present solutions for colors and normals, as they seem most valuable to be supported for voxel scenes. Detailed scenes can potentially result in millions of different colors with small variations in the attribute array. Fortunately, color quantizers can reduce the amount of distinct values significantly without resulting in perceivable differences [Xia97]. While Xiang's original method relied on a clustering in a scaled RGB space, we improve the result by working in the (locally) perceptually uniform CIELAB color space. The amount of colors can be freely chosen by the user; we typically use 12-bit (4096) colors throughout the paper. Note that the method is a data-driven clustering and requires preprocessing to analyze the colors, but yields high-quality results even for a small amount of colors.

For normals, we rely on octahedron normal vectors (ONVs), leading to an almost uniformly distributed quantization [MSS*10, CDE*14]. Using ONVs is beneficial as it yields higher precision for the same number of bits compared to storing one value per dimension. Again, the bit depth of the quantization can be freely chosen.

4.4. Geometry compression

By using an attribute array, we still have to encode additional offsets in the DAG structure, which increases its size. We propose to reduce the DAG's memory consumption by compressing the introduced offsets, as well as the child pointers, which typically make up a large part of the total memory usage.

© 2016 The Author(s)
Computer Graphics Forum © 2016 The Eurographics Association and John Wiley & Sons Ltd.

Figure 4: *Datasets used for evaluation. From left to right: citadel, city, San Miguel and arena.*

Offset compression We observe that the offset from a node to its first child is always $+1$ (see Fig. 2), implying that this offset can be stored implicitly. Further, offsets are typically small in the lower levels of the tree due to the depth-first assignment. Hence, fewer bits are required to represent the offset. To this extent, we analyze each level and find the minimum number of bits required to encode offsets in this level. We round up to bytes for performance reasons, as a texture lookup on the GPU retrieves at least a single byte. In practice, a two-byte offset is sufficient for the lowest five levels in all our examples, leading to a significant improvement. Four or even five bytes are still required for offsets on the highest levels, but these represent much fewer nodes ($\approx 0.1\%$), which makes the increased memory usage non-critical.

Pointer compression We apply the same compression technique as for the offsets to the child pointers as well. While this leads to a slight improvement, the compression does not work as well as for offsets, since the levels that contain most pointers generally require the full four bytes per pointer. However, we observe that some subtrees are used significantly more often than others, which makes entropy encoding [BRGIG*14] a well-suited candidate for memory reduction. We create a table of the most common pointers per level – much in the spirit of our material array in Sec. 4.2 – which is sorted by occurrence in descending order. In the DAG, we then store only an index into the pointer table, which is usually smaller than the original pointer and can be represented with fewer bits.

In practice, we found the following setup to be most effective: each pointer is initially assumed to be one byte. Its first two bits store the type, which then indicates the pointer's actual bit length. Two bits can encode four types; the first three are used to indicate if 6, 14, or 22 bits are used to encode a pointer into the lookup table, and the last type is reserved to indicate that the remaining 30 bits correspond to an absolute pointer (as before, this ensure multiples of bytes). The latter could also be increased to 46 bits, but 30-bit pointers proved sufficient for the DAG nodes in all our examples. While we achieve significant compression with the entropy encoding, it does decrease the performance, as evaluated in Sec. 5.

5. Results

Our method aims at large sparse navigable scenes. For evaluation, we deliberately choose a set of very distinct datasets (see Fig. 4): architectural structures (the *citadel* and *city* scene); complex geometry (tree and plants in the *San Miguel* scene); and a 3D model obtained from real-life photographs using floating-scale surface reconstruction [FG14] (*arena* scene), which is noisy, contains diverse colors, and is a good test case for a realistic dataset. The datasets were produced by voxelizing triangle meshes through depth peeling [Eve01], using the standard extension proposed by Heidelberger et al. [HTG03]. While our compression schemes can handle any spatially coherent data, in practice, we evaluate our method using colors and normals, which are crucial for realistic lighting. We define the compression rate as the memory size of the compressed data over that of the uncompressed data, expressed as a percentage.

Table 1: *Decoupling and palette compression. The numbers are computed for a $64K^3$ resolution (16 hierarchical tree levels) using non-quantized, 24-bit colors for the attributes.*

	Scenes			
Geometry	*Citadel*	*City*	*San Miguel*	*Arena*
DAG voxels (M)	18.3	10.2	18.8	34.7
DAG size (MB)	382	207	395	737
With offsets (MB)	693	374	719	1342
24-bit colors				
Λ (M)	4760	10487	14788	3263
λ (M)	1.66	1.42	3.15	1.57
A (MB)	13619	30004	42309	9336
$\widetilde{A}+M$ (MB)	11922	26257	38792	8174
PC (MB)	10124	24051	10877	2220
Compression rate	74%	80%	26%	24%

In the following, we discuss the results of our compression components separately as described in Sec. 4. Starting with the palette approach, we then analyze the gain of lossless and lossy compression, and present the results of our offset and pointer compression. Next, we compare our approach to existing techniques. Finally, we present results to illustrate the performance of our method and discuss its properties before showcasing several application scenarios.

5.1. Decoupling and palette compression

We show statistics for the DAG-based geometry encoding and the attribute array for our four test scenes in Tab. 1. We list the number of DAG voxels in millions, as well as the memory footprints in MB of the standard and offset-augmented version of the DAG, which is needed to decouple geometry and attributes. The additional offset and pointer compression is analyzed in Sec. 5.3.

© 2016 The Author(s)
Computer Graphics Forum © 2016 The Eurographics Association and John Wiley & Sons Ltd.

Table 2: *Attribute quantization memory footprints and quality. The numbers are computed for a $64K^3$ resolution using 24-, 14- and 12-bit colors, and for a $32K^3$ resolution using 32- and 12-bit ONVs.*

	Scenes			
24-bit colors	*Citadel*	*City*	*San Miguel*	*Arena*
A (MB)	13619	30004	42309	9336
14-bit colors				
PC (MB)	3645	8163	4495	656
Mean RGB err.	2/1/2	2/1/2	2/2/2	2/1/2
Max. RGB err.	20/4/5	6/7/7	9/26/11	34/5/4
Mean ΔE	1.05	0.91	0.80	0.90
Max. ΔE	2.57	2.44	3.00	2.36
Comp. rate	27%	27%	11%	7.0%
12-bit colors				
PC (MB)	2609	5703	3099	438
Mean RGB err.	3/2/3	3/2/3	3/2/3	3/2/3
Max. RGB err.	10/7/29	11/12/11	11/9/25	48/5/3
Mean ΔE	1.72	1.60	1.47	1.75
Max. ΔE	4.25	3.85	5.28	4.38
Comp. rate	19%	19%	7.3%	4.7%
32-bit ONVs				
A (MB)	4496	9994	14081	3110
PC (MB)	912	417	2678	2151
Comp. rate	20%	4.2%	19%	69%
12-bit ONVs				
PC (MB)	135	85.9	407	464
Mean/max. err.	2°/8°	2°/8°	2°/8°	2°/8°
Comp. rate	3.0%	0.9%	2.9%	14%

Further, Tab. 1 shows the number of attributes Λ, which equals the SVO node count, and the number of unique attributes λ, both in millions. The memory size in MB of the attribute array (indicated as A) exceeds that of the DAG by far. Still, λ is usually much smaller than Λ; indeed, the memory cost of the attribute array can already be decreased by using a material array (indicated as $\widetilde{A}+M$). Using our palette compression (indicated as PC) reduces the cost again; we report the compression rate by comparing to the original attribute array A. While overall significant, the use of a lossless scheme seems overly conservative in most practical scenarios and implies that even very similar attributes will have to be represented individually. By allowing for a slightly lossy quantization, the attribute costs can be reduced significantly.

5.2. Attribute quantization

Data quantization might impact precision, but leads to an often similar appearance and a large memory benefit. In Tab. 2, we show the size of the attribute array for 24-bit colors again, and the drastic memory gain of our result using palette compression and quantized colors (14 and 12 bits). To assess the fidelity of our quantization, we report the mean absolute error for each RGB channel over all voxels, as well as the maximum deviation. However, since these numbers do not always give a good impression of perceptual quality, we further report mean and maximum ΔE-values as defined by the CIE94 standard. We use $k_L = 1$, $K_1 = 0.045$ and $K_2 = 0.015$ and the D65 illuminant as the reference white, as per graphics industry standards [Kle10]. Finally, we show the compression rates obtained with our palette approach for quantized colors.

To illustrate the impact during rendering, Fig. 5 shows images from two viewpoints in the citadel scene. These exhibit many unique values, as well as color and normal gradients, which represent a difficult case for quantization. We provide SSIM values (a perceptual similarity metric, where SSIM = 1 means identical) for every image [WBSS04], comparing the result to its non-quantized counterpart. We note that 14-bit colors produce very good results, and even for 12-bit colors the only indication of quantization is the presence of minor banding artifacts at some locations. For 10-bit colors the quality is reduced, as evidenced by the color difference image, but the result is still relatively close to the reference.

Similarly, we report memory footprints for normals in Tab. 2. We consider a $32K^3$ resolution with 32-bit ONVs as a reference, since 96-bit normals at 15 levels could not be handled by our hardware, and the mean error for 32-bit ONVs compared to regular 96-bit normals is only 0.001°, with a theoretically proven maximum error below 0.004°. For quantization, we use 12-bit ONVs, for which we report mean and maximum errors in degrees and show the attained compression rates. As expected, normals compress better than colors for scenes that contain many aligned surfaces, like the city scene. Visually, 16-bit normals produce results indistinguishable from the 32-bit reference while 12-bit and 10-bit normals produce minor and more visible banding on smoothly varying surfaces, respectively (see Fig. 5). Nonetheless, for diffuse shading, such artifacts are barely perceivable and even 10 bits may suffice. For effects like specular reflections, 16-bit normals are preferred.

Tab. 1 and 2 show that the memory footprint of the attributes is now potentially compressed to a similar order of magnitude as the geometry. We can see that the combined use of quantization and our palette compression is very fruitful in practice.

Furthermore, combining colors, normals or even reflectance information rarely leads to a linear increase of memory. For example, the night-time version of the city scene at a $32K^3$ resolution (Fig. 8, left) uses 12-bit colors, 10-bit normals, and 8-bit reflectance information. The total memory footprint is 1492 MB, compared to 1186 MB for just encoding colors. This means that the normals and reflectance information yield a 25.8% overhead, even though the voxel data grew by 150%. This outcome is a consequence of materials with similar colors often having similar normals and reflectance values as well (e.g., a roughly uniformly colored wall).

5.3. Offset and pointer compression

To evaluate our geometry compression, we compare the effect of all our offset and pointer optimizations separately. In Tab. 3, we reiterate the memory footprint of the standard offset-augmented DAG (as

© 2016 The Author(s)
Computer Graphics Forum © 2016 The Eurographics Association and John Wiley & Sons Ltd.

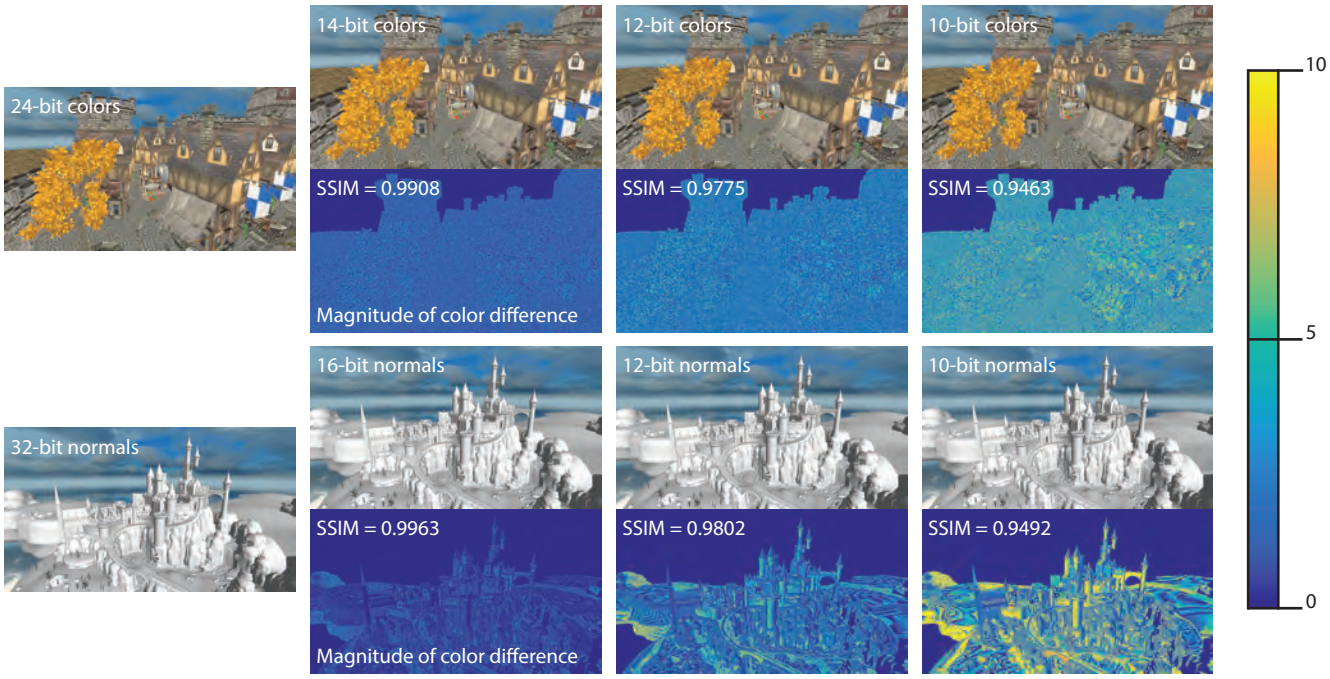


Figure 5: *Perceptual quality of our color and normal quantization. We show the quantized result for 14-bit, 12-bit and 10-bit colors, with their corresponding magnitude of the color difference (i.e., $\sqrt{dR^2+dG^2+dB^2}$) per pixel. We do the same for quantized normals, showing the results for 16-bit, 12-bit and 10-bit normals. The difference values are mapped using the color map on the right, where a difference of 10 corresponds to a bright yellow color. We further report SSIM values for each image to assess the perceptual similarity [WBSS04].*

Table 3: *Offset and pointer compression for a $64K^3$ resolution.*

	Scenes			
DAG size (MB)	*Citadel*	*City*	*San Miguel*	*Arena*
Uncompressed	693	374	719	1342
Implicit offset	623	335	647	1210
Offset compression	499	271	505	907
Pointer compression	629	330	623	1228
8-bit childmask	641	345	665	1243
Pointer entropy	543	290	531	1027
Combined	348	186	316	591
Compression rate	50%	50%	44%	44%

in Tab. 1). We then report results for implicitly storing the first child offset; the per-level byte-precise offset compression; the per-level byte-precise compression for pointers; 8-bit childmasks (the original DAG uses 24 bits of padding); pointer entropy encoding; and, finally, a combination of all these techniques, for which the shown compression rate compares to the standard offset-augmented DAG.

We can see that our approach is quite effective, as we observe that the final memory footprint is on par or even less than the memory cost of the original DAG without the offsets (see Tab. 1).

5.4. Comparison

Now that we have discussed all components, we can compare our compression scheme to existing techniques. As we use 12-bit colors for the comparison, the memory footprint of our complete data structure now equals the geometry size for the combined methods in Tab. 3 plus the attribute size for 12-bit colors in Tab. 2. We compare the cost per voxel of our approach to four other techniques; SVOs, PSVOs [SK06], ESVOs [LK10], and CDAGs (naively adding colors to the original DAG [KSA13]).

For the standard SVO implementation, the memory footprint is computed as follows: we have an 8-bit childmask, a 32-bit pointer, and a 12-bit color value for every node – note that the leaf nodes have no childmask or pointer. The PSVO contains exactly the same data in every node, except for the child pointer.

Besides voxel attributes, ESVOs store additional contour data, but also make use of compression. For color, a DXT1 compression is used while normals are compressed using a novel scheme, which is also lossy, but provides up to 14 bits of precision per axis. In this section, we report ESVO memory footprints as obtained by using the implementation supplied by the authors, which makes use of the aforementioned attribute and contour-based compression.

Finally, we have the original DAG [KSA13], augmented with color data, so that every node contains a 32-bit childmask, one to eight 32-bit pointers, and a 12-bit color value (CDAGs).

For a direct comparison of our attribute compression to that used

© 2016 The Author(s)
Computer Graphics Forum © 2016 The Eurographics Association and John Wiley & Sons Ltd.

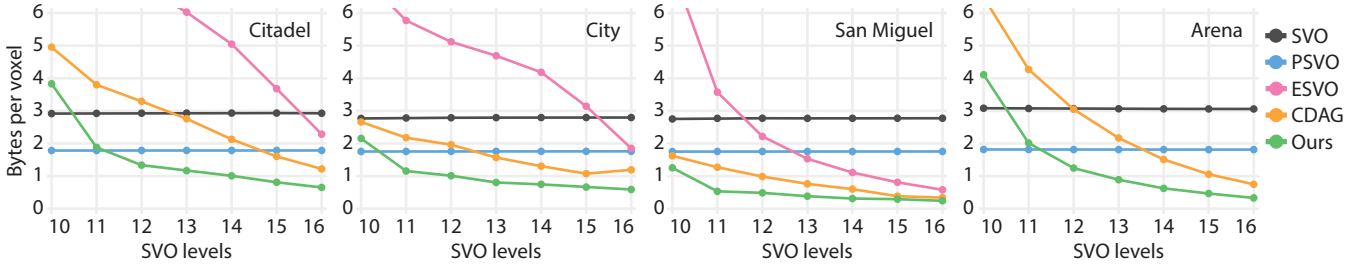


Figure 6: *Memory usage per voxel for our test scenes at different SVO levels. We compare our approach to a colored SVO, PSVOs [SK06], ESVOs [LK11] and a naive colored DAG implementation (CDAGs). Note that the ESVO implementation was unable to load the arena scene.*

Table 4: *Comparison to the state of the art for a $64K^3$ resolution.*

	Scenes			
	Citadel	*City*	*San Miguel*	*Arena*
SVO voxels (M)	4760	10487	14788	3263
Size (MB)	13285	27966	39122	9520
Bytes/voxel	**2.93**	**2.80**	**2.77**	**3.06**
PSVO size (MB)	8105	17595	24748	5638
Bytes/voxel	**1.79**	**1.76**	**1.75**	**1.81**
ESVO voxels (M)	1533	2782	1168	
Size (MB)	10374	18506	8174	
Bytes/voxel	**2.29**	**1.85**	**0.58**	
CDAG voxels (M)	286	629	251	117
Size (MB)	5540	11922	4791	2326
Bytes/voxel	**1.22**	**1.19**	**0.34**	**0.75**
Our voxels (M)	18	10	19	34
Geometry (MB)	348	186	316	591
Attributes (MB)	2609	5703	3099	438
Total size (MB)	2957	5889	3415	1029
Bytes/voxel	**0.65**	**0.59**	**0.24**	**0.33**
Compression rate	22%	21%	8.7%	11%

Table 5: *Construction times in minutes at different resolutions for our four test scenes, using the naive colored DAG implementation and our decoupling and palette compression, with 12-bit colors.*

	Scenes			
Resolution	*Citadel*	*City*	*San Miguel*	*Arena*
$4K^3$	1.60/2.58	0.75/6.01	1.40/5.31	1.08/1.83
$8K^3$	2.65/17.1	2.93/30.3	5.90/22.0	3.13/7.35
$16K^3$	10.8/64.5	13.2/217	20.0/157	10.0/21.3

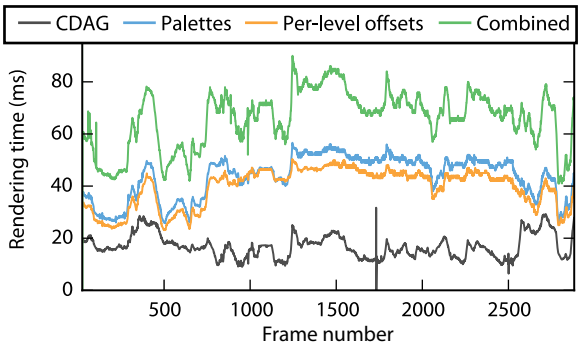


Figure 7: *Rendering times while navigating through the citadel scene at a $32K^3$ resolution, obtained by raycasting in full HD.*

by ESVOs, we built the Sibenik scene at a $8K^3$ resolution. Here, ESVOs reported a memory footprint of 2120 MB without using contours [LK10]. Not using contours is important, as, contrary to geometry, a similar quality as regular colored SVOs can only be achieved for attributes if they are not cut off during traversal. Further, as the data quality for ESVOs is not evaluated, it is difficult to provide a comparison; hence, we use 24-bit colors and 32-bit ONV normals for the palette compression, which ensures better quality than the lossy schemes applied by ESVOs. Our palette compression is more flexible when compared to the constant DXT1 rate, which results in only 1171 MB for our entire data structure.

Fig. 6 and Tab. 4 illustrate that our approach outperforms other methods by a significant margin. We report bytes per voxel for all techniques, *always* with reference to the SVO node count. We list the actual voxel count in millions for SVOs, ESVOs, CDAGs, and our method separately. We show the geometry and attribute size separately and combined for our method. Finally, we report compression rates as compared to a standard SVO implementation.

5.5. Performance

Construction As our focus was mostly on compression quality, not performance, we did not investigate significant acceleration techniques for the DAG algorithm, nor for our palette compression. Still, the construction times for building our data structure are interesting, as they illustrate the computational overhead of involving attributes. In Tab. 5, we report timings on an i5 CPU in minutes for both standard colored DAGs (before the slash) and our method

© 2016 The Author(s)
Computer Graphics Forum © 2016 The Eurographics Association and John Wiley & Sons Ltd.

Figure 8: *Several applications of our compressed SVO. From left to right: encoding reflectance information in materials for the city scene, rendered at a resolution of* $32K^3$*; color bleeding in the CrySponza scene, at an SVO resolution of* $4K^3$*, using 16 samples per pixel, and secondary and tertiary ray tracing at a* 512^3 *resolution; rendering of a dense dataset of a Christmas tree at a* $512 \times 512 \times 999$ *resolution.*

(after the slash); for the latter, we see an increase up to an order of magnitude. Still, it is feasible to compute high-resolution scenes on commodity hardware, and the slow construction does not hurt performance during rendering. The construction time depends mostly on the number of compression attempts that our algorithm explores; if large blocks are already found in the first phase, as for the arena scene, the cost of the palette compression is significantly reduced.

Rendering We did not particularly optimize our rendering algorithm; in each frame, we cast rays from the camera and traverse the SVO with a standard stack-based approach to find the first intersecting voxel that projects to an area smaller than a pixel. To still demonstrate that our method is capable of real-time performance, Fig. 7 shows timings for a walk-through in the citadel scene in full HD at a $32K^3$ resolution, obtained using an NVIDIA Titan X. We compare the rendering times for palette compression, per-level byte-precise offsets, and using all our compression techniques, to naive colored DAGs (CDAGs). We can conclude that palettes and offset compression have some impact on the performance, but still enable real-time rendering while yielding significant compression rates. The entropy encoding on the other hand has a bigger influence and we only achieve interactive rates. Still, it can be useful for memory gain, especially when the geometry is relatively large, like for the arena scene (see Tab. 4). Further, the rendering cost is several orders of magnitude lower than for pointerless solutions while still avoiding high memory costs.

5.6. Applications

To demonstrate the versatility of our approach, and of SVOs in general, we showcase several applications. Like the original DAG, we are able to obtain high-resolution hard shadows for the whole scene. With normals, however, we can look into more interesting applications, such as reflections, by shooting secondary rays while maintaining real-time performance (Fig. 8, left).

We have also implemented a simple approach to color bleeding through single-bounce global illumination (Fig. 8, middle). We shoot multiple secondary rays via stratified sampling of the hemisphere – which means the samples are uniformly distributed, but contain a random offset – and then shoot tertiary rays from the intersecting voxels to determine if they are in shadow. We attain interactive rates with 8 secondary rays per pixel.

Since our method, like the DAG, exploits both similarity and sparsity, we can to some extent compress dense data as well (Fig. 8, right). For the shown Christmas tree, we are able to obtain a lossless compression rate of 38.6% when comparing our complete data structure to the original input file, which is approaching state-of-the-art methods for dense datasets (29.4%) [GWGS02]. When applying a simple filtering to remove scanning noise in the air, we additionally profit from the sparsity and achieve rates below 10%.

6. Conclusions

We have presented a novel SVO compression scheme, which relies on the decoupling of geometry from additional voxel data. Our mapping is efficient and introduces little overhead, enabling separate compression methods for topology and voxel attributes. Furthermore, we introduced compression schemes for child pointers, which also reduces the cost of traditional DAGs. For attribute compression, we proposed a combination of our quantization and our lossless palette approach, which implicitly exploits spatial coherence. We showed that our solution reduces memory usage from 4.49 times (for the citadel) up to 11.5 times (for San Miguel) compared to standard SVO implementations. Our method outperforms state-of-the-art SVO compression methods for all test scenes.

The high compression rates allow us to store colored SVOs with up to 17 levels (a voxel resolution of $128K^3$) completely on the GPU. We demonstrated real-time rendering performance using commodity hardware and showcased several applications such as color bleeding and reflections, for which additionally normal and reflectance attributes were encoded.

For future work, investigating more advanced material properties for the voxel data (e.g., BRDFs encoded via spherical harmonics or transparency values) are interesting directions.

Acknowledgements

This work was partially supported by the FP7 European Project Harvest4D and the Intel VCI.

© 2016 The Author(s)
Computer Graphics Forum © 2016 The Eurographics Association and John Wiley & Sons Ltd.

References

[BRGIG*14] BALSA RODRÍGUEZ M., GOBBETTI E., IGLESIAS GUITIÁN J., MAKHINYA M., MARTON F., PAJAROLA R., SUTER S.: State-of-the-art in compressed GPU-based direct volume rendering. *Computer Graphics Forum 33*, 6 (2014), 77–100. 1, 2, 6

[CDE*14] CIGOLLE Z. H., DONOW S., EVANGELAKOS D., MARA M., MCGUIRE M., MEYER Q.: A survey of efficient representations for independent unit vectors. *Journal of Computer Graphics Techniques 3*, 2 (2014), 1–30. 2, 5

[CNLE09] CRASSIN C., NEYRET F., LEFEBVRE S., EISEMANN E.: GigaVoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proc. of I3D* (2009), pp. 15–22. 2

[CNSE10] CRASSIN C., NEYRET F., SAINZ M., EISEMANN E.: *GPU Pro*. AK Peters, 2010, ch. X.3 Efficient rendering of highly detailed volumetric scenes with GigaVoxels, pp. 643–676. 2

[Eve01] EVERITT C.: *Interactive order-independent transparency*. Tech. rep., NVIDIA Corporation, 2001. 6

[FG14] FUHRMANN S., GOESELE M.: Floating scale surface reconstruction. *Trans. on Graphics 33*, 4 (2014), 46. 6

[GMIG08] GOBBETTI E., MARTON F., IGLESIAS GUITIÁN J. A.: A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer 24*, 7 (2008), 797–806. 2

[GWGS02] GUTHE S., WAND M., GONSER J., STRASSER W.: Interactive rendering of large volume data sets. In *Proc. of VIS* (2002), pp. 53–60. 2, 10

[HTG03] HEIDELBERGER B., TESCHNER M., GROSS M. H.: Real-time volumetric intersections of deforming objects. In *Proc. of VMV* (2003), pp. 461–468. 6

[JMG16] JASPE VILLANUEVA A., MARTON F., GOBBETTI E.: SSVDAGs: Symmetry-aware sparse voxel DAGs. In *Proc. of I3D* (2016). 2

[JT80] JACKINS C. L., TANIMOTO S. L.: Oct-trees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing 14*, 3 (1980), 249–270. 1, 2

[Kle10] KLEIN G. A.: *Industrial color physics*. Springer, 2010. 7

[KSA13] KÄMPE V., SINTORN E., ASSARSSON U.: High resolution sparse voxel DAGs. *Trans. on Graphics 32*, 4 (2013), 101. 2, 3, 8

[KSA15] KÄMPE V., SINTORN E., ASSARSSON U.: Fast, memory-efficient construction of voxelized shadows. In *Proc. of I3D* (2015), pp. 25–30. 2

[LH06] LEFEBVRE S., HOPPE H.: Perfect spatial hashing. *Trans. on Graphics 25*, 3 (2006), 579–588. 2

[LH07] LEFEBVRE S., HOPPE H.: Compressed random-access trees for spatially coherent data. In *Proc. of EGSR* (2007), pp. 339–349. 2

[LK10] LAINE S., KARRAS T.: *Efficient sparse voxel octrees – analysis, extensions, and implementation*. Tech. rep., NVIDIA Corporation, 2010. 1, 8, 9

[LK11] LAINE S., KARRAS T.: Efficient sparse voxel octrees. *Trans. on Visualization and Computer Graphics 17*, 8 (2011), 1048–1059. 2, 9

[Mea82] MEAGHER D.: Geometric modeling using octree encoding. *Computer Graphics and Image Processing 19*, 2 (1982), 129–147. 1, 2

[MSS*10] MEYER Q., SÜSSMUTH J., SUSSNER G., STAMMINGER M., GREINER G.: On floating-point normal vectors. *Computer Graphics Forum 29*, 4 (2010), 1405–1409. 2, 5

[NLP*12] NYSTAD J., LASSEN A., POMIANOWSKI A., ELLIS S., OLSON T.: Adaptive scalable texture compression. In *Proc. of HPG* (2012), pp. 105–114. 2

[SAM05] STRÖM J., AKENINE-MÖLLER T.: iPACKMAN: High-quality, low-complexity texture compression for mobile phones. In *Proc. of HWWS* (2005), pp. 63–70. 2

[SK06] SCHNABEL R., KLEIN R.: Octree-based point-cloud compression. In *Proc. of SPBG* (2006), pp. 111–120. 2, 8, 9

[SKOA14] SINTORN E., KÄMPE V., OLSSON O., ASSARSSON U.: Compact precomputed voxelized shadows. *Trans. on Graphics 33*, 4 (2014), 150. 2

[WBSS04] WANG Z., BOVIK A. C., SHEIKH H. R., SIMONCELLI E. P.: Image quality assessment: From error visibility to structural similarity. *IEEE Trans. on Image Processing 13*, 4 (2004), 600–612. 7, 8

[Wil15] WILLIAMS B. R.: *Moxel DAGs: Connecting material information to high resolution sparse voxel DAGs*. Master's thesis, California Polytechnic State University, 2015. 2

[Xia97] XIANG Z.: Color image quantization by minimizing the maximum intercluster distance. *Trans. on Graphics 16*, 3 (1997), 260–276. 2, 5

© 2016 The Author(s)
Computer Graphics Forum © 2016 The Eurographics Association and John Wiley & Sons Ltd.