HARVEST4D

HARVESTING DYNAMIC 3D WORLDS FROM COMMODITY SENSOR CLOUDS

Publications for Task 8.3

Deliverable 8.31

Date:	29.6.2016
Grant Agreement number:	EU 323567
Project acronym:	HARVEST4D
Project title:	Harvesting Dynamic 3D Worlds from Commodity Sensor Clouds

Document Information

Deliverable number	D8.31
Deliverable name	Publications for Task 8.3
Version	1.0
Date	2016-06-29
WP Number	8
Lead Beneficiary	DELFT
Nature	R
Dissemination level	PU
Status	Final Version
Author(s)	DELFT, TUDA (only corrections)

Revision History

Rev.	Date	Author	Org.	Description
0.1	2016-06-13	Timothy Kol	DELFT	Draft
0.2	2016-06-28	Samir Aroudj	TUDA	Corrections
1.0	2016-06-29	Timothy Kol	DELFT	Final

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This deliverable contains original unpublished work except where clearly indicated otherwise.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 8.3 and how they fit into the work plan of the project.

The objective of task 8.3 was to enable rendering on mobile and web devices. Making use of various devices (cellphones, tablet PCs, high-end workstations or simple multimedia solutions that deliver access to an Internet browser) is a crucial element of Harvest4D. For example, in order to guide scanning processes, a previsualization should be possible from everywhere. In contrast to standard video information, 3D rendering offers many opportunities for customized solutions. One direction lies in fusing nearby client-side rendering with far server-side imagery and multi-client level of detail. Another direction is to adapt video-compression methodologies to reduce data size and transfer. Such solutions are typically limited to temporally changing 2D content but some of its aspects might be transferrable. One particular issue is random access, as many compression schemes exploit continuity and consistency. Standard compression uses key frames at uniform temporal locations to skip parts of the flux. This strategy is not directly applicable to our data. Instead, we explore automatic solutions to cluster data in order to find “key representations“ from which we can efficiently encode subsequent configurations. The second research direction focuses on exploiting the device capacities directly; e.g., a lower resolution implies that a lower resolution image can be provided by the server as well. More generally, the space of possible adaptations is large, including color depth, stereo rendering, frame rate, even environmental lighting that might affect the visibility of the image on the device.

There are so far seven publications that are mainly associated with task 8.3. These can be found in the appendix of this deliverable. The publications are:

- Marco Callieri, Matteo Dellepiane, Roberto Scopigno
  Remote Visualization and Navigation of 3D models of archeological sites
  3D ARCH: 3D Virtual Reconstruction and Visualization of Complex Architectures, 2015
- Mohamed Radwan, Stefan Ohrhallinger, Elmar Eisemann, Michael Wimmer
  Cutting Freely: Occlusion-Aware Surface Processing ***(submitted)***
  SIGGRAPH Asia, 2016
- Markus Schütz, Michael Wimmer
  High-Quality Point Based Rendering Using Fast Single Pass Interpolation
  Proceedings of Digital Heritage, Short Papers, 2015
- Leonardo Scandolo, Pablo Bauszat, Elmar Eisemann
  Compressed Multiresolution Hierarchies for High-Quality Precomputed Shadows
  Computer Graphics Forum (Proc. of Eurographics), 2016

- Jean-Marc Thiery, Emilie Guy, Tamy Boubekeur, Elmar Eisemann
  Animated Mesh Approximation With Sphere-Meshes
  Transactions on Graphics, 2016
- Federico Ponchio, Matteo Dellepiane
  Fast decompression for web-based view-dependent 3D rendering
  3D Web Technology, 2015
- Gianpaolo Palma, Manuele Sabbadin, Massimiliano Corsini, Paolo Cignoni
  Enhanced Visualization of Detected 3D Geometric Differences ***(submitted)***
  Technical Report ISTI-CNR, 2016

Three other papers are related to task 8.3 and can be found in the deliverables they mainly contribute to:

- Dawid Pająk, Robert Herzog, Radosaw Mantiuk, Piotr Didyk, Elmar Eisemann, Karol Myszkowski, Kari Pulli
  Perceptual depth compression for stereo applications (Task 4.1)
  Computer Graphics Forum (Proc. of Eurographics), 33(2), 2014
- Christopher Schwartz, Roland Ruiters, Reinhard Klein
  Level-of-Detail Streaming and Rendering using Bidirectional Sparse Virtual Texture Functions (Task 8.1)
  Computer Graphics Forum (Proc. of Pacific Graphics), 2013
- Bas Dado, Timothy R. Kol, Pablo Bauszat, Jean-Marc Thiery, Elmar Eisemann
  Geometry and Attribute Compression for Voxel Scenes (Task 4.4)
  Computer Graphics Forum (Proc. of Eurographics), 2016

2 DESCRIPTION OF PUBLICATIONS

2.1 OVERVIEW

As our web rendering applications could be used by many users who are not necessarily familiar with 3D navigation on a computer, an obvious first step is to devise an intuitive navigation technique for the complex 3D environments Harvest4D deals with. To this end, we came up with a method that integrates the two prevalent visualization techniques in 3D navigation: the trackball and the first-person camera [Callieri et al. 2015]. While this works well for large navigable scenes, for exploring and understanding 3D models that contain multiple layers of geometry, we need to employ a different strategy. We make use of user-guided, occlusion-free surface selecting and cutting operations to unveil details within nested models, peeling away layer by layer based on user requirements [Radwan et al. 2016].

Since mobile and web rendering often suffer from limited computing power, an obvious step in mitigating this is to increase the performance of the rendering algorithm. For Harvest4D, point

cloud rendering in particular is an interesting topic. For this reason, we propose an efficient single pass point rendering algorithm [Schütz and Wimmer 2015].

Besides computing power, memory is also an important bottleneck due to both network transfer limitations and decreased storage room of mobile devices. To tackle this problem, content compression is desirable which we have pursued in the form of shadow map compression [Scandolo et al. 2016] as well as approximation of animated models by representing them using spheres meshes [Thiery et al. 2016]. For compression, fast decompression is also necessary. This is why our final method regarding storage specifically aims at very fast decompression besides high compression rates [Ponchio and Dellepiane 2015] based on multi-resolution web rendering.

Finally, to bring the time dimension to web rendering, we have developed an online tool for visualizing differences between two 3D models in an intuitive fashion [Palma et al. 2016].

2.2 REMOTE VISUALIZATION AND NAVIGATION OF 3D MODELS OF ARCHEOLOGICAL SITES

In this paper we present a solution for rendering and navigation of complex 3D environments using web browsers where the usual trackball paradigm for navigation cannot be applied [Callieri et al. 2015]. The system integrates two different visualization modalities that account for different needs regarding exploration of complex environments, like archeological sites.

Starting from a multi-resolution web rendering engine based on WebGL, our proposed method integrates two different but complementary navigation modes. First, *bird's eye view* where users can explore a model from its top. Second, *first person mode* where users can walk through the environment. Figure 1 illustrates two corresponding examples. The two modes are linked by a point of interest which helps users navigating from the top and permits intuitive switching between them. Navigation is constrained by an image whereas its channels contain the data needed for realistic interaction, such as ground height for collision tests. So there is no need for proxy 3D models or similar. Moreover, unused image channels can encode other information which can be used for navigation, like location or invisible hotspots. Such an image can be generated with a semi-automatic approach starting from geometry or by hand with usual image processing tools.

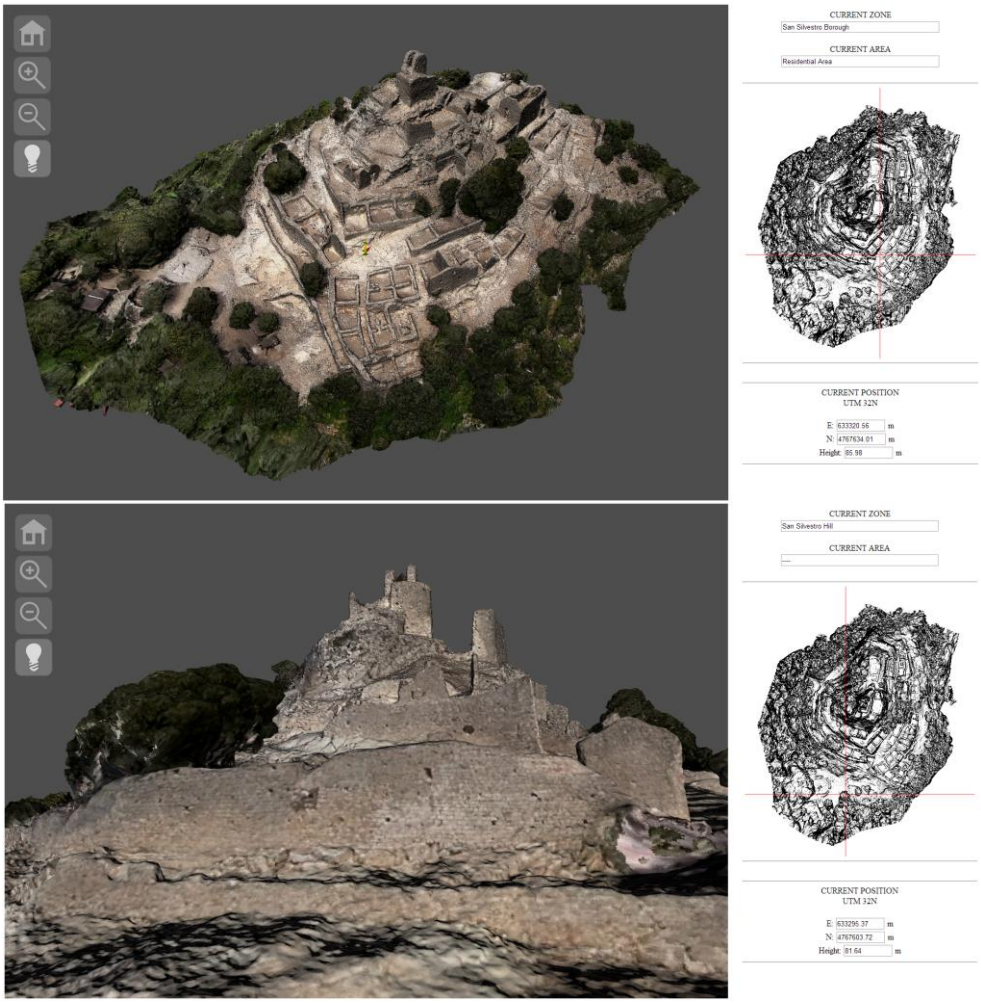


Figure 1: Snapshot of bird's eye (top), first person navigation mode (bottom) and helping minimap (right).

In order to simplify exploration, there is also a minimap with the purpose of giving immediate feedback on the current position, thus, avoiding users to "get lost" when zooming or in first person view. It is also a convenient way to instantly jump to another location.

2.3 CUTTING FREELY: OCCLUSION-AWARE SURFACE PROCESSING

Intuitive navigation and exploration for scenes that contain multiple layers of geometry, such as the heart model in Figure 2, require novel visualization and interaction techniques. In this paper we present an approach for surface selection operations on complex models [Radwan et al. 2016]. User-guided surface selection operations, which are straightforward in a plane, become challenging on non-strictly convex surfaces because of self-occlusions. Since the occlusions change with the view, users have to alternate between moving to an unobstructed view and performing those selection operations. Our method enables operations like selecting or cutting in a single view by mapping user-defined selection or cutting curves or areas as continuous projections onto the object's surface. Our mapping is unaffected by occlusions to guarantee a

seamless brush stroke or a manifold cut. The projected solution includes the farthest surface layer by default. But if that is not the desired layer users can roll back and forth through other solutions. We show that this enables a number of applications in an entirely different and easy way. Example use cases are creating illustrative cutaways from nested models, animating them and removal of spurious interior artifacts from isosurface meshing.

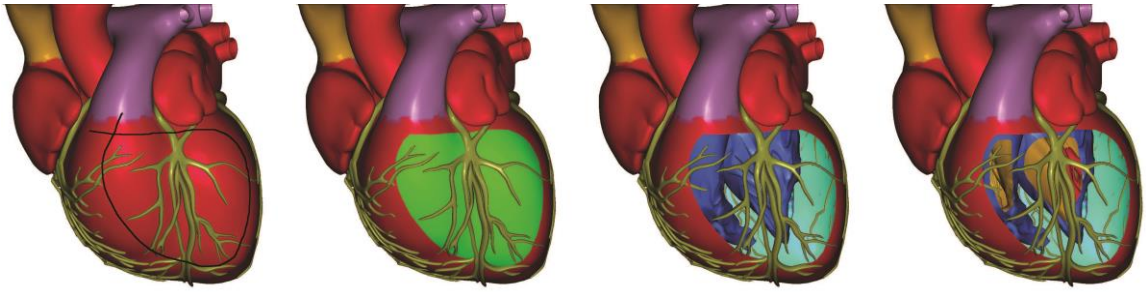


Figure 2: Several surface layers can be cut away intuitively for this model of the human heart.

2.4 HIGH-QUALITY POINT BASED RENDERING USING FAST SINGLE PASS INTERPOLATION

Web rendering of large point clouds requires highly efficient algorithms due to limited computing power. To this end, we present a method to improve visual quality of point cloud renderings through a nearest neighbor-like interpolation of points [Schütz and Wimmer 2015]. This allows applications to render points at larger sizes in order to reduce holes without reducing readability of fine details due to occluding points. Figure 3 presents this algorithm property in comparison to other approaches. The implementation requires only a few modifications to existing shaders which makes it eligible to be integrated in software applications without major design changes.

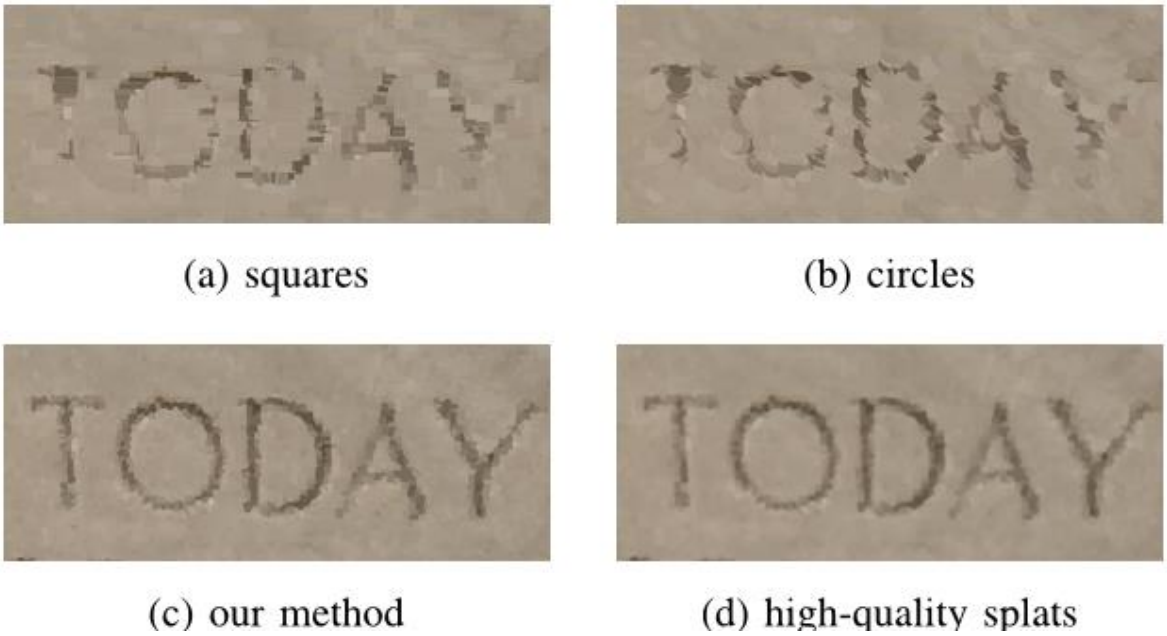


Figure 3: Point rendering via squares (a) and circles (b) suffers from occlusions. Our method (c) and high-quality splats (d) improve readability of high-frequency details.

2.5 COMPRESSED MULTIRESOLUTION HIERARCHIES FOR HIGH-QUALITY PRECOMPUTED SHADOWS

For web rendering, the amount of available memory is limited. For this reason, we need compression in order to render complex scenes. In this paper we propose such a compression technique for shadow maps [Scandolo et al. 2016]. Shadow mapping is traditionally limited by texture resolution. We present a novel lossless compression scheme for high-resolution shadow maps based on precomputed multiresolution hierarchies. Traditional multiresolution trees can compactly represent homogeneous regions of shadow maps at coarser levels but require many nodes for fine details. By conservatively adapting the depth map, we can significantly reduce the tree complexity. Our proposed method offers high compression rates, avoids quantization errors, exploits coherency along all data dimensions and is well-suited for GPU architectures. Our approach can be applied for coherent shadow maps as well, enabling several applications, including high-quality soft shadows and dynamic lights moving on fixed trajectories. Figure 4 shows the shadow quality we achieve despite high compression.

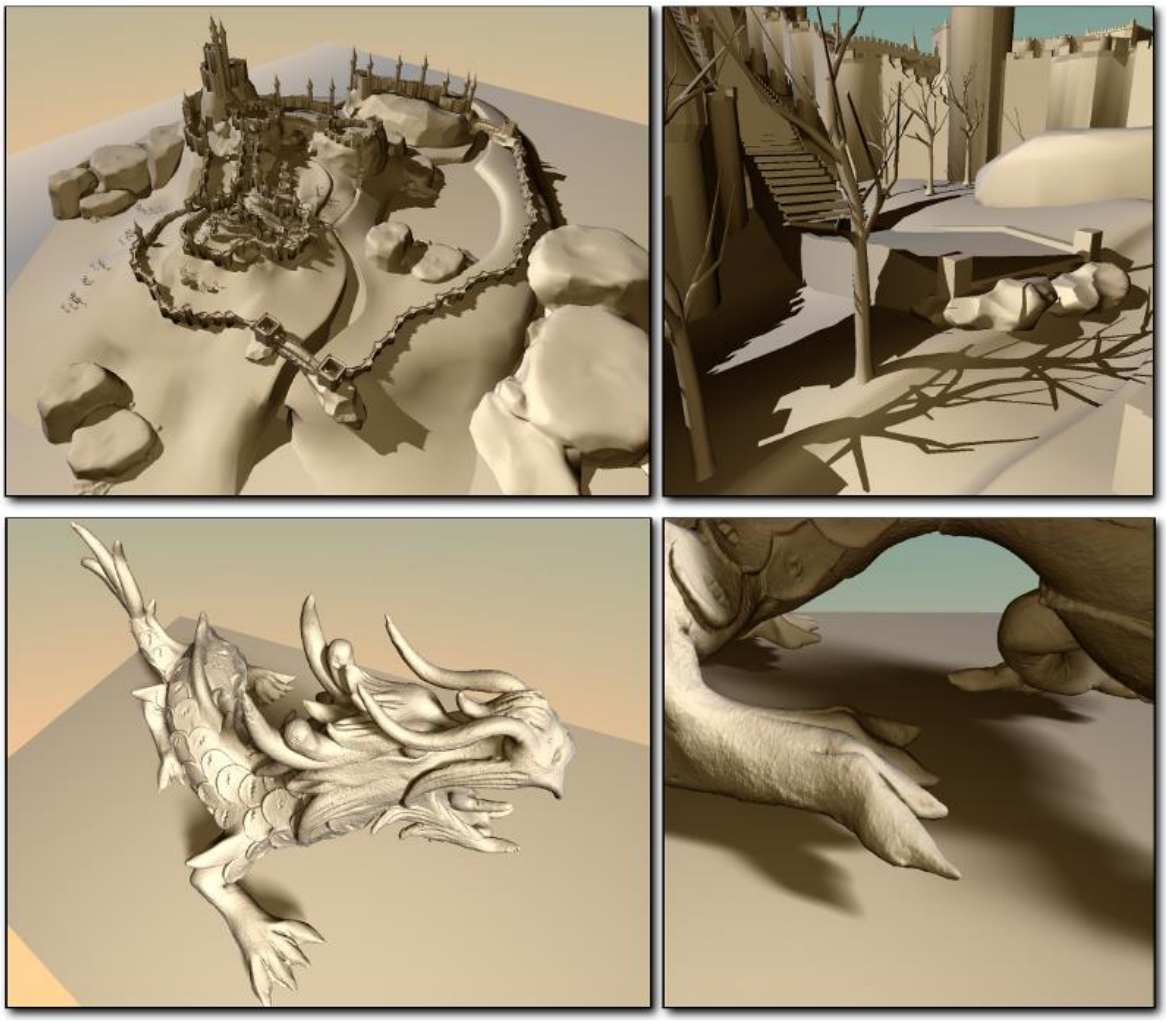


Figure 4: High-quality shadows in a static environment using a compressed 1M² shadow map (top) and precomputed soft shadows stored in 2048 coherent shadow maps, each with a 2048² resolution (bottom).

2.6 ANIMATED MESH APPROXIMATION WITH SPHERE-MESHES

Besides shadows, the mesh itself can also use much memory, which is undesirable for web rendering. Especially complex animated meshes may require compressed approximations, which is exactly what we propose in this paper [Thiery et al. 2016]. Performance capture systems can be used to acquire high-quality animated 3D surfaces, usually in form of a dense 3D triangle mesh. Extracting a more compact, yet faithful representation is often desirable. But existing solutions for animated sequences are surface-based which leads to limited approximation power in the case of extreme simplification. We introduce animated sphere meshes, which are meshes indexing a set of animated spheres. Figure 5 shows seven frames created with the help of our novel approximation. Our solution is the first to output animated volumetric structures to approximate animated 3D surfaces. It optimizes for approximation via spheres, connectivity, and temporal coherence. The result of our algorithm is a multi-resolution structure from which the user can choose the wanted level of simplification for rendering in real-time. We demonstrate the use of

animated sphere-meshes for low-cost approximate collision detection. Additionally, we propose a skinning decomposition, which automatically rigs the input mesh to the chosen level of detail. The resulting set of weights are smooth, compress the animation, and enable easy edits.

Figure 5: Approximating an animated mesh using sphere meshes.

2.7 FAST DECOMPRESSION FOR WEB-BASED VIEW-DEPENDENT 3D RENDERING

In this paper we present a novel multi-resolution WebGL-based rendering algorithm which combines progressive loading, view-dependent resolution and a mesh compression technique that provides good rates and a decoding speed of millions of triangles per second using JavaScript [Ponchio and Dellepiane 2015]. The method is based on a class of multi-resolution structures where the "primitive" of the multi-resolution is a patch made of thousands of triangles. Figure 6 shows this exemplarily. It adopts an improved partition strategy based on unbalanced kD-trees and, more importantly, a novel compression scheme tailored around the need for decompression speed. The compression algorithm encodes patches independently of each other. That is why the method can be efficient and fast even for small meshes and boundary vertices replicated on neighboring patches remain consistent despite compression. Model connectivity is compressed using an adaptation of a state-of-the art algorithm, while vertex attributes are compressed using a global quantization grid. Our method applies an entropy encoding of the input stream of bits using Tunstall coding. Our results prove that the proposed algorithm has very high performance w.r.t. decoding very large models in JavaScript.

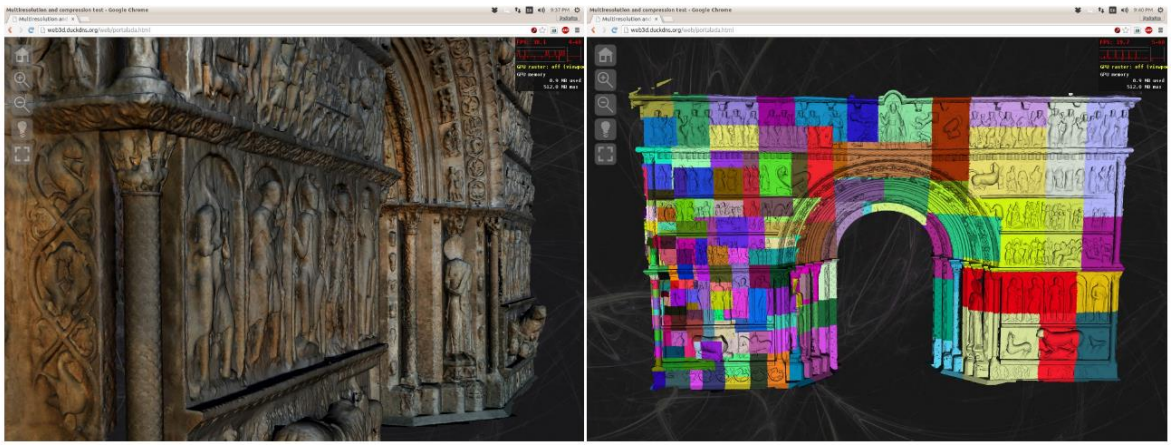


Figure 6: Rendering of a multi-resolution model (left) and its structure (right).

2.8 ENHANCED VISUALIZATION OF DETECTED 3D GEOMETRIC DIFFERENCES

In this paper we propose an interactive technique for better visualization of geometric changes between two colored 3D triangle meshes [Palma et al. 2016]. The two models represent the same scene acquired at different times. Such models can for example be generated by 3D scanning or multi-view reconstruction techniques. The goal is to give users a web visualization tool with three main features:

- to allow a linear interaction model (slider) to switch between the two time steps
- to make scene changes clear and as easy to understand as possible
- to preserve original color and geometry information of the two input models

Starting from the computation of a change probability map using a state-of-the-art method that segments each 3D model into change (dynamic) and no-change (static) areas, the basic idea is to provide a screen space interpolation technique for the renderings of both 3D models according to users' temporal preference. Importantly, we use different interpolation curves for the different types of model areas. For the choice of these interpolation curves, we take insights of cognitive research on the so called Change Blindness phenomenon into account. Two user studies show that the proposed method is effective in visualization of changed geometry w.r.t. time.

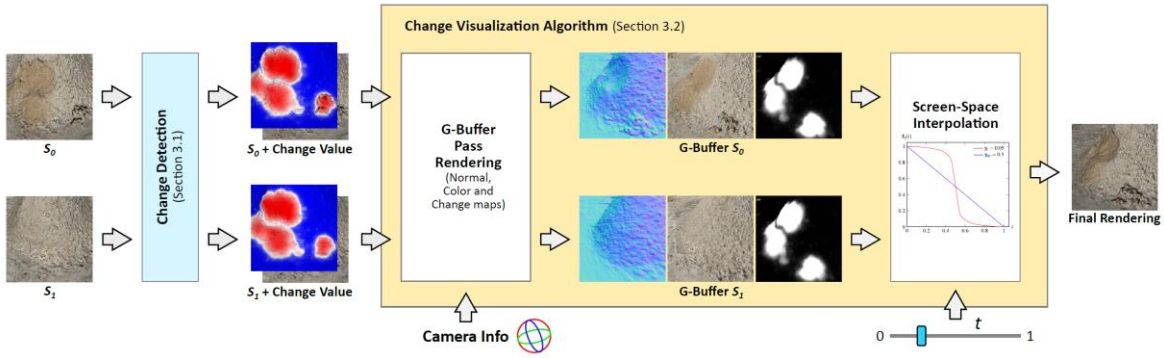


Figure 7: Algorithm overview.

3 REFERENCES

- Marco Callieri, Matteo Dellepiane, Roberto Scopigno
  Remote Visualization and Navigation of 3D models of archeological sites
  3D ARCH: 3D Virtual Reconstruction and Visualization of Complex Architectures, 2015
- Mohamed Radwan, Stefan Ohrhallinger, Elmar Eisemann, Michael Wimmer
  Cutting Freely: Occlusion-Aware Surface Processing ***(submitted)***
  SIGGRAPH Asia, 2016
- Markus Schütz, Michael Wimmer
  High-Quality Point Based Rendering Using Fast Single Pass Interpolation
  Proceedings of Digital Heritage, Short Papers, 2015

- Leonardo Scandolo, Pablo Bauszat, Elmar Eisemann
  Compressed Multiresolution Hierarchies for High-Quality Precomputed Shadows
  Computer Graphics Forum (Proc. of Eurographics), 2016
- Jean-Marc Thiery, Emilie Guy, Tamy Boubekeur, Elmar Eisemann
  Animated Mesh Approximation With Sphere-Meshes
  Transactions on Graphics, 2016
- Federico Ponchio, Matteo Dellepiane
  Fast decompression for web-based view-dependent 3D rendering
  3D Web Technology, 2015
- Gianpaolo Palma, Manuele Sabbadin, Massimiliano Corsini, Paolo Cignoni
  Enhanced Visualization of Detected 3D Geometric Differences ***(submitted)***
  Technical Report ISTI-CNR, 2016

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 in the public Harvest4D webpage (for already published papers), or in the restricted section of the webpage (for papers under submission, conditionally accepted papers, etc.).

The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XL-5/W4, 2015
3D Virtual Reconstruction and Visualization of Complex Architectures, 25-27 February 2015, Avila, Spain

REMOTE VISUALIZATION AND NAVIGATION OF 3D MODELS OF ARCHEOLOGICAL SITES

M. Callieri, M. Dellepiane, R. Scopigno

Visual Computing Lab, ISTI-CNR, Via G. Moruzzi 1, 56124 Pisa (PI), Italy
Web: http://vcg.isti.cnr.it/ Contact: *surname*@isti.cnr.it

ABSTRACT:

The remote visualization and navigation of 3D data directly inside the web browser is becoming a viable option, due to the recent efforts in standardizing the components for 3D rendering on the web platform. Nevertheless, handling complex models may be a challenge, especially when a more generic solution is needed to handle different cases. In particular, archeological and architectural models are usually hard to handle, since their navigation can be managed in several ways, and a completely free navigation may be misleading and not realistic. In this paper we present a solution for the remote navigation of these dataset in a WebGL component. The navigation has two possible modes: the "bird's eye" mode, where the user is able to see the model from above, and the "first person" mode, where the user can move inside the structure. The two modalities are linked by a point of interest, that helps the user to control the navigation in an intuitive fashion. Since the terrain may not be flat, and the architecture may be complex, it's necessary to handle these issues, possibly without implementing complex mesh-based collision mechanisms. Hence, a complete navigation is obtained by storing the height and collision information in an image, which provides a very simple source of data. Moreover, the same image-based approach can be used to store additional information that could enhance the navigation experience. The method has been tested in two complex test cases, showing that a simple yet powerful interaction can be obtained with limited pre-processing of data.

1. INTRODUCTION

The remote visualization (via Web) of complex geometry has become feasible only in the last couple of years. The WebGL framework allowed to overcome the proprietary plugins issue, and three-dimensional content is slowly becoming a usual type of content for web pages.
The issues about the visualization of 3D models are related not only to the necessity to handle complex data, but also to the interaction with them. Navigating 3D environments is not trivial, due to both the nature of data and the fact that most of the users are not used to that. This is especially true when dealing with 3D models of complex environment (terrains with buildings and ruins, often found in the archeological field); even more when 3D models come from sampling technologies (3D scanning or 3D-from-photo), which produce high-resolution, unstructured triangulated models. What is the best way to interact with them? A completely free navigation may be hard to handle, while extremely constrained approaches may limit their potentials. While it is possible to borrow ideas from the entertainment world (i.e. videogames industry), the best solution would be to find a flexible, lightweight system needing only a small amount of preprocessing.
This paper presents a method for the navigation of complex archelogical 3D environments, especially tailored for web visualization. The main contributions include:

- An intuitive navigation paradigm which includes two possible modes: the *bird's eye* view, where the user can explore the model from top; and the *first person* mode, where the user can walk inside the environment in a walk-through fashion. The two modes are linked by the presence of a *point of interest*, which helps the user in the navigation from top, and permits an intuitive passage between the two modes.

- An image-based encoding of the constraints for navigation. Each image channel contains the data needed for realistic interaction (i.e. height from ground, collision), without the need of the presence of proxy 3D models or similar. Moreover, the other channels of the image can encode also other information which can be used for the navigation (i.e. location or invisible hotspots). The image can be generated with a semi-automatic approach starting from the geometry, or by hand with the usual image processing tools.

The approach is especially suited to archeological environments made by one or more "levels", but it could be adapted to other environments. It was tested on two cases: an entire insula in Pompei acquired with terrestrial laser scanners (where location information was used to integrate the 3D model with existing documentation, linked through a website) and a medieval village whose model was acquired with UAV. In both cases, a very intuitive navigation was obtained without the need of complex pre-processing operations.

2. RELATED WORK

Remote visualization of complex architectural environments deals with several general issues. The main ones are the integration of three-dimensional contents on the web, and the navigation of complex environments. While a comprehensive overview of all the related approaches goes well beyond the scope of the paper, we provide an overview of the state of the art in the following subsections.

2.1 3D and the Web

Three-dimensional data have always been considered part of multimedia content, but their role in the context of web pages hasn't been a major one until a few years ago. This was mainly due to the difficulty to handle complex data, but also to a lack of standardization. Hence, their visualization and use was supported through the use of embedded software components, such as Java applets or ActiveX controls (ACT, n.d.).

This contribution has been peer-reviewed.
doi:10.5194/isprsarchives-XL-5-W4-147-2015

Some research effots were devoted to define a common data format (Raggett, 1995, Don Brutzmann, 2007), but 3D scene visualization was still delegated to external software components.
Only an initiative by the Khronos Group (Khronos Group, 2009a), the creation of WebGL standard (Khronos Group, 2009c), was able to generate a remarkable change. WebGL is a mapping of OpenGL|ES 2.0 specifications (Khronos Group, 2009b) in JavaScript. Direct access to the graphics hardware by the web browsers allows to fully exploit the potentials of rendering (Evans et al., 2014).
Following this standard, several actions were made to provide a sort of interface between the low-level OpenGL commands and the data visualization and navigation. We can see two different directions of research: the first one is more dedicated to a declarative approach, based on the concept of *scenegraph*. Two examples of declarative programming solutions are X3DOM (Behr et al., 2009) and XML3D (Sons et al., 2010). In alternative to the declarative approach, several other actions went in the direction of an imperative approach, where a more direct interaction is possible. Several libraries have been developed (most of them based on Javascript as a basic language), ranging from scene-graph-based interfaces, such as Scene.js (Kay, 2009) and GLGE (Brunt, 2010), to more programmer-friendly paradigms, such as SpiderGL (Di Benedetto et al., 2010), WebGLU (DeLillo, 2009), and Three.js (Dirksen, 2013). Several examples of integration of 3D data on the web are appearing now. Among the practical issues that have to be solved to provide usable tools, there are the necessity to integrate the models with other types of data (Jankowski and Decker, 2012, Callieri et al., 2013), but also the need for methods to handle the visualization of single complex objects (Behr et al., 2012, Callieri et al., 2013). The Smithsonian X3D explorer (http://3d.si.edu) is an alternative example where 3D models are associated to additional content, but the structure and flexibility of the proposed system are not known.

2.2 Camera control in Computer Graphics

The issue of 3D camera control is well known in the context of Computer Graphics. Please refer to the work of Christie (Christie et al., 2008) for a comprehensive overview.
The main solution developed to interactively observe and inspect a 3D model is the virtual trackball (Chen et al., 1988, Bell, 1988, Shoemake, 1992, Henriksen et al., 2004), that allows 3D rotations with just 2 Degrees of Freedom input devices, such as the mouse. This interaction works very well when the aim is to rotate around an object, but it tends to fail when the environment is more complex. For example, when the user needs to explore the inside of an object, or when its shape cannot be approximated to a sphere, this approach usually fails.
On the other side, a totally free navigation may be difficult to implement, and it could lead to a *lost-in-space* effect, where the user is not able to find the orientation in the context of a scene. Moreover, there are important object-related issues that should be taken into account (i.e. in an architectural environment, the user does not want go below the ground).
Solutions have been proposed to constrain and guide the trackball paradigm (Fitzmaurice et al., 2008, Hachet et al., 2008), but in the case of more complex environments (i.e. the architectural ones) other methods have been implemented. They could be based on a preliminary analysis of the scene to calculate pre-defined paths (Andujar et al., 2004), or constraining the camera with force-fields to avoid collisions (Wan et al., 2001, McCrae et al., 2009).
Another possible solution is to select a set of possible points of view, and constrain the navigation to these points, which can be connected by the means of pre-computed paths (Hanson and Wernert, 1997). The choice of the points of view describing a complex environment is a very complex task (Scott et al., 2003), and authoring is usually necessary (Burtnyk et al., 2002, Burtnyk et al., 2006).
Following this approach, image-based techniques have been used to remove limitations on scene complexity and rendering quality for interactive applications. Please refer to the recent work by Di Benedetto (Di Benedetto et al., 2014) and its bibliography for a comprehensive overview. Nevertheless, the aim of our work was to provide a solution supporting the free navigation of the model, trying to define implicit constrains and taking advantage of the strong points of different navigation paradigms.

3. EFFICIENT RENDERING

The first problem related to the visualization scheme was to render effectively the complex 3D model representing the area of interest. As we stated in the introduction, we are focusing towards the visualization of complex 3D models of archeological/historical areas, created by sampling the real environment (by 3D scanning or 3D-from-photos). These models are not a ”simple” terrain, that can be easily managed using 2.5D methods, or well known terrain-specific level-of-detail approaches, but require a full 3D management. Additionally, we have to work with extremely complex 3D models, made of millions of triangles.

While not the main contribution of this work, it is nevertheless necessary to present the solution which we have used in order to stream these dataset over the net and render them inside a web browser.

3.1 WebGL and SpiderGL

Our visualization framework is based on WebGL (Khronos Group, 2009c), a component of HTML5. Thanks to this, the 3D content works natively inside the browser, without the need of plugins on most modern browsers (Firefox, Chrome and Internet Explorer) on all platforms. WebGL provides direct access to the GPU of the PC, enabling a programmer to write extremely optimized low-level code, as one would do in standard software programming. However, since WebGL is very low-level, its direct use is not really common, and many developers use support libraries, able to provide higher level functionalities, data structures and helper functions. To this aim, we decided to use SpiderGL (Di Benedetto et al., 2010), a JavaScript support library oriented to Computer Graphics programming.

Using these components we built a simple, yet flexible framework for the creation of 3D presentation of high-res models on the web. Using a declarative-like approach, it is possible to define a simple scene. The framework takes care of asynchronous data loading, event management, rendering and animation. All the components of the scene (3D models, trackball, camera) are configurable and extensible, making it possible to create simple visualization webpages with ease, but at the same time, when a more complex visualization is needed, it is possible to fully exploit the modular/configurable capabilities of the framework. This is what we have done in this work: by tweaking the behavior of the renderer and of the navigation/interaction components, we were able to create a specialized visualization webpage.

One of the core modules of this framework is the management of multiresolution 3D models, a feature that we have used to stream and display the huge 3D models shown in the examples (Section 5.).

3.2 The Multiresolution engine

Displaying high resolution models on a web browser is not just a matter of optimizing the rendering speed, but it also involves considering the loading time and network traffic caused by transferring a considerable amount of data over the network. Loading a high-resolution model as a whole through the web requires to transfer a single chunk of data in the order of tens, if not hundreds, of megabytes: this causes a lot of band usage, and the user has to wait for the transmission to end before seeing any visual result.

Multiresolution techniques provide the solution for both rendering and data transfer. These schemes generally split the geometry in smaller chunks; for each chunk, multiple levels of detail are available. Transmission is *on demand*, requiring only to load and render the portions of the model strictly needed for the generation of the current view. While this approach is key to be able to render very large models at an interactive frame rate, it is also helpful to optimize the data transfer over the network, since the geometry will be divided in small chunks and only transferred when needed. The model is immediately available for the user to browse it, even though at a low resolution, and it is constantly improving its appearance as new data are progressively loaded. On the other hand, since refinement is driven by view-dependent criteria (observer position, orientation and distance from the 3D model sections), only the data really needed for the required navigation are transferred to the remote user.

Figure 1: The multiresoluton Engine. On left: the patches used , smaller and more detailed near the camera, larger and at a lower-resolution farther away. Middle: when rendering, the geometry appear as a continous surface. On right: the multiresolution easily manage the whole 27 million triangles model of the Pompeii Insula.

We implemented one of those multiresolution scheme, Nexus (Cignoni et al., 2005) (http://vcg.isti.cnr.it/nexus/), on top of the SpiderGL library (Di Benedetto et al., 2010), obtaining very good performances. In the two examples shown in this paper, we are dealing with high-resolution 3D models: the Pompeii insula is more than 27 millions triangles (Figure 1), while the San Silvestro Hill is around 22 millions (Figure 7). Both datasets are displayed at interactive rate, with almost immediate startup at the page loading, and the progressive refinement provides good visual results even on slower networks. This makes possible a quick, seamless exploration of the entire dataset, and supports instant jumping from one part of the area to another.

4. NAVIGATION

As we stated, navigation is the most important component of the visualization. Our aim was to implement a navigation simple to use, but able to fully explore mixed ground-buildings 3D dataset like the Pompeii insula or the San Silvestro Hill.

Instead of trying to implement a single all-purpose navigation method, which would result in an extremely complex interface, we decided to implement two different but complementary navigation methods: a ”bird’s eye” view, to explore the dataset from above, and a ”first-person” view to explore the area in a more immersive way. In this way, each navigation mode is easier to use and more specialized but, by combining both navigation methods, it is possible to fully explore the 3D dataset with greater flexibility.

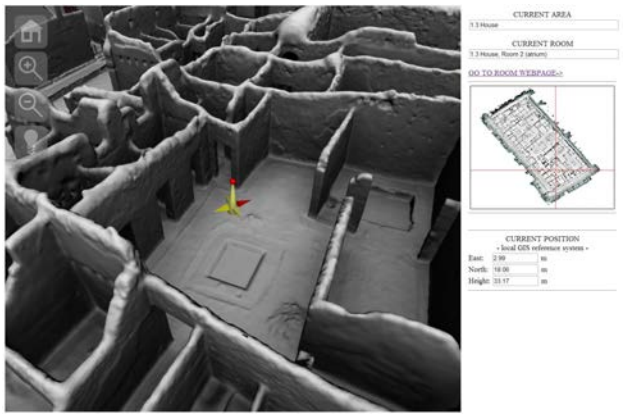


Figure 2: The visualization webpage interface. On left: the 3D view, in bird’s eye view mode, with the current point of interest shown by the 3D locator. On top right: the current area and room name, tracked using the location channel of the map. On middle right: the minimap, graphically showing the current position. On bottom right: the current position of the user, expressed in this specific case as relative coordinates with respect to a local origin defined in a GIS tool

4.1 Bird’s eye mode

Bird’s eye view mode mimics flying above the environment or, with a more precise metaphore, looking from above at a maquette of the area. The user viewpoint hovers the area, looking down at specific details (see Figure 2).

This is, however, not a completely unbound navigation (like in the free-camera paradigm), but is similar to what is available in applications such as Google Earth. Basically, the camera always refers to a *point of interest*, located on the ground of the environment. The user may move this point, panning across the map, and change the camera orientation (vertical tilting, orbiting, and zooming). We did not consider the use of a completely free camera because it is really difficult to be controlled by non-experts (especially in a web browser environment), without providing a significative increase of flexibility. Even considering its constrains, this navigation mode enables the user to easily reach every portion of the environment. This is especially true in the Pompeii dataset, but also in most of the archeological sites: since most of the roofs are missing, it is possible to see almost all of the walls and details of the structure.

The *point of interest* is shown on the 3D view with a locator (a simple 3D model, visible in Figure 2); this helps the user in having a clear feedback on its position inside the area. The locator is nearly human-size, to better understand the scale of the area, and show at its base the cardinal points (the red arrow points towards north).

The implementation of bird’s eye view is quite easy. Since we are dealing with a terrain-like environment, it is simply a matter of defining an absolute XY positioning on the ground (X axis aligned with East-West axis and Y axis on the North-South one). Beside the extents of the dataset area, it is necessary to have a way to follow the ground geometry when moving across the area. Most datasets presents irregular, sloped, uneven terrain; and having a way to accurately position the point of interest is crucial

to cope with these irregularities. We will explain this point in Section 4.4.1. No collision is necessary in this navigation mode, since it would impose unnecessary constrains in user movement, making the navigation of the whole area more difficult.

Bird's eye view is the starting navigation mode of the visualization webpage. The user may, at any time, switch to the first-person view (see next Section). While switching to first person view, location and orientation are preserved, i.e. the point of interest of the bird's eye will become the location of the first-person camera, and the view direction of the bird's eye will be the same of the one used by the first-person (see Figure 3). In this way, it is much easier to go back and forth between views, allowing for a seamless exploration of the dataset. The same correspondence will be preserved when switching back from first-person to bird's eye.

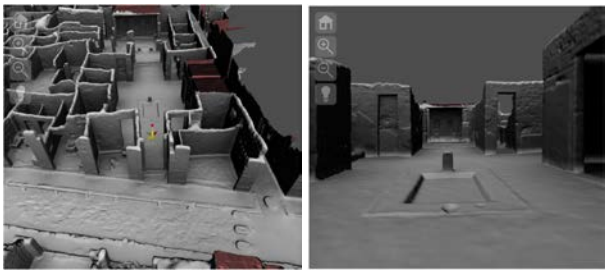


Figure 3: Switching from bird's eye to first-person preserves the user position and orientation, as it is visible in these images.

4.2 First person mode

Even if, in an environment like Pompeii, with most of the ceilings missing, the bird's view is more or less sufficient to explore the entire map, it is sometimes useful/necessary to "jump into" the 3D model, to explore the environment like it would be done in person.

The first-person view mode lets the user moves around the environment at ground level (the camera is at a human-compatible height). Our implementation follows the same mechanism used in many similar tools: the first-person lets the user look in all direction (using the mouse) and move around on the ground (using the mouse or the WASD keys, generally used in videogames).

Similarly to the bird's eye mode, also in the first-person mode it is necessary to follow the ground geometry when moving around. There is, however, an additional constrain often found in similar navigation modes: the collision with walls and other geometries. Implementing these features for a first-person view is always complex, given the unstructured nature of the kind of 3D models we are using, generated from 3D scanning. In most videogames or interactive visualization, the 3D model used comes from manual modeling; it has a lower resolution and, more important, it is highly structured (walls and floors are explicitly marked, and often a collision mesh is also manually created). This helps a lot the navigation, but it is not easily obtainable when dealing with this kind of 3D models.

For the terrain-height following, we have used the method described in Section 4.4.1 , while in Section 4.4.2 we explain the method used for managing walls collision.

4.3 Minimap

In order to simplify the exploration, we also added a mini-map to the visualization page (see Figure 2). The minimap has two purposes: to give an immediate feedback on the current position, thus avoiding users to "get lost" while zooming or when in first-person view, but also to have a way to instantly jump to another position.

The position of the point of interest is shown using a crosshair drawn over the image, dynamically updated at every user movement. Conversely, when the user clicks on a point on the minimap, the point of interest is instantly moved to that location. Both these actions exploit the fact that there is a perfect match of the area covered by the 3D dataset, the minimap and the one covered by the data map (see next section). Locations are passed between these three elements using relative coordinates (0.0-1.0), to gather independence from image resolution.

4.4 The "Navigation Image" encoding

As we stated in the previous section, in order to provide a easy-to-use navigation, it is necessary to implement some mechanism able to follow the terrain height and to manage collisions. Additionally, tracking the user position is another need often present in this kind of visualization. In all cases, it all boils down to knowing, for a given position in the dataset area, a specific information: which is the terrain height in this point? Is there a wall in this point, or is it free-ground? In which room/zone is this point?.

The idea is to have all these location-dependent information stored in a single, easy-to-access structure; knowing the location of the user (the point-of-interest for the bird's eye, and the viewer position in first-person), it is easy to retrieve the location-specific data, and have the visualization page react accordingly. We chose to store all the data in an image file, where each channel encodes a specific information. PNG format is used, given its lossless compression. An example of such image is shown in Figure 4. This "Navigation Image" is read at the page loading; then, it is drawn on a hidden canvas and read back in memory in an array. At this point, the pixels are directly addressable by Javascript code. This structure is accessed using relative coordinates, thus ensuring independency from the image resolution, and an easier mapping between coordinates.

4.4.1 Red channel - Terrain Height Most of the 3D datasets built using sampled data have a high-res non-flat terrain; moving across this terrain requires then to alter the camera height to follow the geometry of the ground. This is generally obtained by creating a low-res collision ground, which is then tested as the user moves. Given that this lower-res collider is generally smooth, encoding its height in a map is not a problem, and makes much easier its use. The red channel of our map is used to store the base height of the terrain, i.e. a smoothed height-field representing the ground without the walls and other occluders. By using this map, the camera always follows the terrain when moving in the bird's eye and first-person view. The height map is smoothed to avoid useless camera shaking, and the walls and small colliders are not included, to have the camera follow the "real" ground level. Its use is straightforward: every time the XY position of the user changes, its Z coordinate is modified according to the value encoded in this map.

4.4.2 Blue channel - Collision Another feature which is often sought in this kind of visualization is the collision with walls and other large geometries. This is a non-trivial task, since it requires, in most game engines, the manual creation of collision meshes (often, with specific geometrical-topological constrains, to obtain better performances), and a complex management of the collision check. The blue channel of our map is used to store the collision/validity map: each pixel encodes if the corresponding area is free, occupied by an occluder (mostly, walls), or impassable (the area outside the 3D model, or not suitable for first-person navigation). The collision map is used mostly when the

user is in first-person view; every time the user XY position has to be updated, the map is checked: if the new XY position would cause a collision, the motion is culled at the wall, preventing the collision. Bird's eye view does not use the collision map, since it may be useful for the user to place its point of interest anywhere in the map and, beside this, ignoring the walls and unsurmountable terrain does allow a quicker exploration of the map. The collision map is, however, used in bird's eye view when clicking on the minimap: if the user has asked do go in an "impossible" position, its position is snapped to the closest "safe" position. The same mechanism is used when the view mode is switched from bird's eye to first-person: if the point of interest is currently inside a wall, the user is moved to the closest open ground, to avoid having the user being trapped.

4.4.3 Green channel - Location A situation often occurring in this kind of datasets is the need to determine the exact location of the user, not just in terms of absolute XY coordinates on the map (which we already have), but in terms of areas, sub-areas, rooms, or any other semantical subdivision of the map. Again, this information is easily encoded in an image map. The green channel of our map is used to determine the position of the user point of interest. Each pixel simply stores the ID of the area; the 255 available values are enough to encode the areas for most datasets.
Every time the user position changes, the ID of the pixel under the user XY position is fetched and used to access a JSON (JavaScript Object Notation, an open standard for representing simple data) structure with all the information on the areas: a globally accessible variable is updated and a call-back function is called when the ID changes, in order for the webpage to react accordingly.

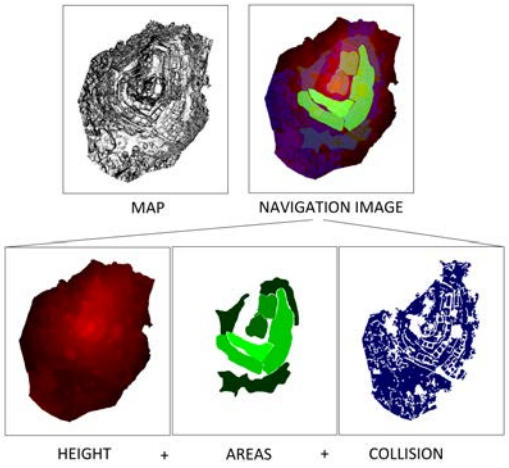


Figure 4: The visualization webpage interface. On left: the 3D view, currently in bird's eye view mode, with the point of interest shown by the 3D locator. On top right: the current area and room name, tracked using the location channel of the map. On middle right: the minimap, graphically showing the current position. On bottom right: the current position of the user, expressed in this specific case as relative coordinates with respect to a local origin defined in a GIS tool

4.4.4 Other channels and encoding Since we are using PNG, it is possible to also have an Alpha channel; at the moment, it is not used. It may contain additional IDs of areas (if more than 255 areas have to be encoded); or could be used in conjunction with the red height channel, to have more resolution on the vertical axis, when dealing with a steeper terrain; or to encode any other location-based data. The assignment of the channels is completely arbitrary, and may be easily altered to cope with specific characteristics of the dataset. It is also possible, if running out of space, to add another image map, adding three/four more channels.

While there are other ways to encode the same type of information, this access strategy exploits already existing and highly optimized mechanisms of the browser. Additionally, given the data contained in the channels, it behave very well when compressed. The most valid alternative would be to keep all this data in a vectorial format; this would surely work, but with a limited impact on the size of the data (depending, for example, on how complex the geometry of the walls is), and, more importantly, it would require a much more complex authoring and editing.

Considering the Pompeii dataset, the PNG image used is only 500kb in size, for a 1142 x 980 resolution, with a physical size of each pixel around 10cm, more than enough to provide an accurate management for terrain height, walls collision and location tracking.

4.4.5 Authoring navigation images From the start, we wanted a method for managing the aforementioned features (ground height, wall collision, area detection) which was simple both when using it in realtime *and* when creating the data needed for its use.

The height channel may be easily created from the 3D model: a simple color-coded (Z value to red value) orthographic rendering from above (or below) is enough to generate this channel. Then, to have a more continuous height, it is possible to smooth the rendered image, it is easier and have better results than starting from a smoothed geometry. We stated that it is better to not consider, in the height map, the walls and other non-terrain geometries. These areas may be manually found and eliminated in the 3D model or in the rendered image (and then smoothly filled). It is also possible to use, on the 3D model, some heuristics to automatically select walls and similar geometries, and then cut them. This automatic approach is viable and time saving; in the cases where the detection is not really accurate, a small manual intervention would help obtaining a good result in a shorter time, with respect to the completely manual approach.

The collision channel is slightly more complex. While it is an option to manually mark all the walls and colliders using an image-editing software, this may be time consuming. The same heuristics mentioned for the height channel may also be applied to find the walls for the collision channel. Also in this case, the result of this automatic detection may be further refined with a limited manual intervention, resulting in a much cleaner result. This is the procedure used to generate the collision maps shown in Figure 4 and Figure 5.

Both of these maps (height and collision), can also be created from GIS raster or vector data: this method is easy and time-effective, making it possible to exploit existing data.

Unfortunately, for most applications, the "location" channel will necessarily should be created manually, since there is not a straightforward way to automatically subdivide a 3D model in an archeological-sensible way. However, creating this image from an existing GIS vector/raster layer is a viable (and recommended) option.

4.4.6 Page interconnection Especially when dealing with complex archeological environment, it is often possible to have existing databases or documentation-rich websites. It would be interesting to connect the visualization page with these existing repositories. This would mean having a way, on one hand, to open from the existing website the 3D visualization page with the point of interest and view parameters already focusing on a specific area

and, on the other hand, from the current point of interest, go back to the existing website, directly accessing the section related to the current 3D location.

This is indeed possible, thanks to the parametric nature of the visualization page. In order to open the visualization page, the *Query String* mechanism can be used. It is possible to specify completely the position and view parameters; for example, the following url query string opens the visualization with the point of interest located at 36.0 meters East, 12.6 meters North, with camera looking at 45 degree with respect to north direction, 60 degree azimuthal angle, zoom level 2.0:

```
http://aaa.com/pompeii3D.html?StartNS
   =36.0&StartEW=12.6&StartPhi=45.0&
       StartTheta=60.0&StartZoom=2.0
```

It is also possible to ask the webpage to go to one of the "areas" or "rooms" defined in the JSON structure mentioned in section 4.4.3. This structure contains a list of areas/subareas, each one with an associated ID, name, description, and URL (see next paragraph) and parameters for setting point of interest and view direction to frame it. By passing the name of the desired area to the URL of the 3D visualization webpage, the page will open by framing the desired area, fetching the appropriate parameters in this JSON structure. The following url query string opens the visualization with the point of interest location and camera orientation stored in the JSON entry for the "house of Torello di Bronzo":

```
http://aaa.com/pompeii3D.html?
     StartRoom=1.7_house_torello
```

Going in the other direction is also easy. Since at all times, the webpage knows where the user point of interest is located, in absolute coordinates or at area/room level (thanks to the location map), it is possible to open back the existing website at the desired section. This can be implemented by using the same JSON structure just mentioned: the URL stored in each area/room is used for this specific purpose.

5. RESULTS

The proposed rendering and navigation method was applied on two real datasets, in order to test the usability and the possible uses of the image map encoding. The first one represented a nearly planar but slightly sloped, complex architectural structure, while the second one represented a less planar, irregularly urbanized medieval borough on top of an hill.

5.1 The Pompei Insula V.1

The first test case was the model of the Insula V.1 of Pompei. This wide area (1330 square meters), was acquired in the context of a project coordinated by the Swedish Archeological School (Dell Unto et al., 2013) using phase shift laser scanners. The acquisition campaigns and the subsequent processing generated several complete models of the insula at different resolutions. The model (which was composed by 27M triangles, with a resolution of around 3cm) was also referenced w.r.t. the reference system of the archeologists.
The original map of the insula was used as the minimap, while the height and collision were created in a semi-automatic fashion. The walls were removed from the 3D model, using an automatic selection function of MeshLab, and the collision map was obtained by producing a snapshot of a blue colored model from the same point of view as the minimap (see Figure 5,right).
The height map was obtained through filling the missing portions of the geometry via Poisson reconstruction, and encoding the height from the ground in the red channel using a per-vertex color function of MeshLab tool (see Figure 5,left). Some examples of the navigation of the Pompei dataset are shown in Figures 2 and 3. In this case, the green channel of the image map was used to link the 3D model to an already existing website that contained the documentation (text and images) for each single room of the Insula. By encoding in the green channel the Id of each room, it was possible to access directly from the navigation context the corresponding page on the Pompei Project website.
Figure 6 shows an example of the link between the position of the focus point and the corresponding data sheet on the website. A possible future extension of this system would be a perfect integration with the website, where the user could jump in and out of the two websites in an integrated fashion.

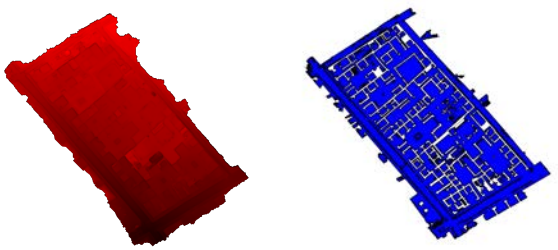


Figure 5: The height and collision maps for the Pompei example

Figure 6: Linking the navigation with a data sheets website.

5.2 San Silvestro village

San Silvestro was a miners medieval village which is currently part of a minerary park in Tuscany, Italy. The village was the target of an acquisition campaign using a UAV and Multi-view stereo matching techniques. Several different flights around the village allowed to extract more than 400 frames that were used for obtaining a complete multi-view stereo reconstruction.

The model of the village can be extremely valuable for several aspects, and the possibility to freely navigate a three-dimensional model gives the chance to overcome its peculiar placement on top of an hill. The final 3D model (with color) was made of 22M triangles, with a resolution of nearly one point every 5 cm.
As in the previous example, the model was scaled to real measures, and geo-referenced using the documentation provided by the park authorities. Then, it was processed to be used with multi-resolution visualization. The mini map (see Figure 4,top-left) for navigation was obtained through a non photo-realistic rendering of the model from top. The complete image map can be seen in Figure 4 with all the channels encoded: the height on the red channel, the location on the green one, and the collision in the blue one.
The height channel was obtained in an automatic fashion by

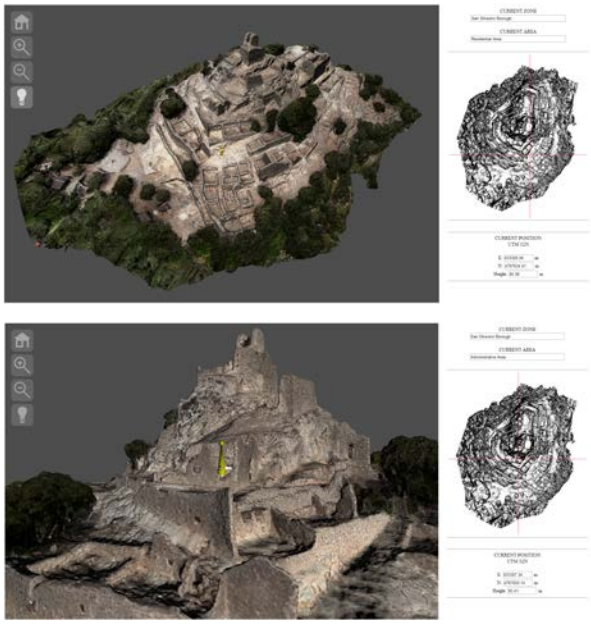


Figure 7: Two snapshots of bird's eye navigation of San Silvestro model

encoding the color w.r.t. the height from the ground using a per-vertex color function of MeshLab tool, and generating a snapshot from the top view. The collision channel was obtained in a semi automatic fashion, by first removing the vertical walls from the geometry, and then modifying by hand the blue colored remaining geometry. In this case, the collision surface was less easy to define, due to the presence in the model of architectural remains, rock and vegetation. The location channel was easily produced by hand starting from the minimap. It defines the different (i.e. residential, industrial, church) areas of the village.
Figure 7 shows two points of view that can be reached using the bird's eye modality: it's easy to get a view from the top, but also to explore portions of the village. On the top right of each snapshot, the peculiar area associated to the focus point is shown. Moreover, the UTC coordinates are available (bottom right).
Figure 8 shows two points of view that can be reached with first person navigation: one that can re-create a real visit (the one from the ground), the other providing a point of view not available to visitors (it is currently not possible to climb on top of the tower). The test on this model shows that it is possible to prepare a dataset where the ground is not planar, with only a minimal intervention in the creation of the image maps, and providing an extremely flexible yet intuitive navigation.

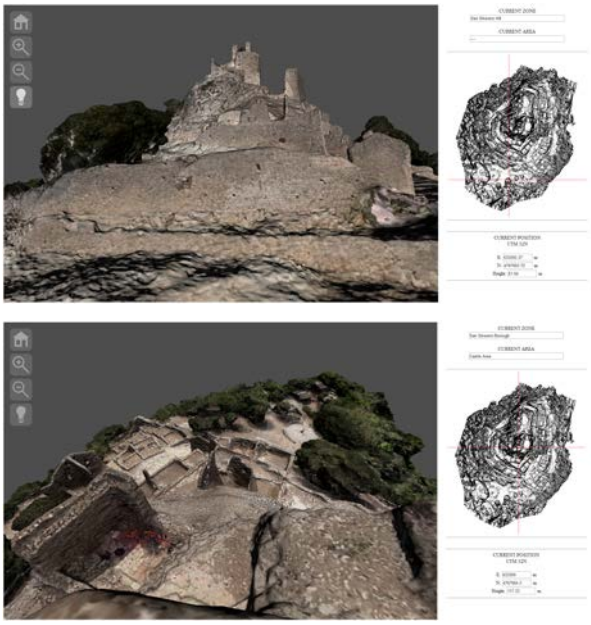


Figure 8: Two snapshot of first person navigation of San Silvestro model. Top: on top of the tower. Bottom: from the industrial area outside the village.

6. CONCLUSIONS AND FUTURE WORK

This paper presented a solution for the navigation of 3D environments on which the usual trackball paradigm for navigation cannot be applied. The system integrates two different visualization modalities (linked by the concept of "point of interest"), that could account for different needs in the exploration of a complex architecture.
Moreover, the important information needed to ensure a realistic experience are stored in a image map, that could be used also to store additional information (i.e. the location of the current point of interest). The image map can be produced with a limited effort by the user, and it's easily integrated in the navigation environment.
The main limitations of the current approach are: the need for data that are not organized on different levels (although the navigation could "switch" from one layer to another when needed), the use of bird's eye view when the model has roofs (although this could be overcome by using ad-hoc visualization, like x-ray-like shaders), and the necessity to produce the image map "by hand" (although the procedure is quite straightforward and could be automatized, for example using a MeshLab script).
Nevertheless, the system could be directly applied to a number of possible test cases in the field of Cultural Heritage, from archeological excavation to city navigation, or even for the exploration of artifacts with small detail (like engraving or small decorations).
Regarding future improvements of the system, three possible directions of work can be outlined: the implementation of simple interaction functionalities (measurement, sections, simple annotation), the creation of authoring tools to help in preparing new scenes, and a better integration with existing web visualization systems.

ACKNOWLEDGEMENTS

The research leading to these results was funded by EU FP7 project ICT Harvest4D (http://www.harvest4d.org/ , GA n. 323567) and EU INFRA Project Ariadne (GA n. 313193, http://www.ariadne-infrastructure.eu/). We'd like to thank the University of Lund,

Scuola Archeologica Svedese, and Andrea Berton, for their support.

REFERENCES

ACT, n.d. Microsoft ActiveX Controls. http://msdn.microsoft.com/en-us/library/aa751968(VS.85).aspx.

Andujar, C., Vazquez, P. and Fairen, M., 2004. Way-finder: guided tours through complex walkthrough models. Computer Graphics Forum 23(3), pp. 499–508.

Behr, J., Eschler, P., Jung, Y. and Zöllner, M., 2009. X3dom: a dom-based html5/x3d integration model. In: Proceedings of the 14th International Conference on 3D Web Technology, Web3D '09, ACM, New York, NY, USA, pp. 127–135.

Behr, J., Jung, Y., Franke, T. and Sturm, T., 2012. Using images and explicit binary container for efficient and incremental delivery of declarative 3d scenes on the web. In: Proceedings of the 17th International Conference on 3D Web Technology, Web3D '12, ACM, New York, NY, USA, pp. 17–25.

Bell, G., 1988. Bell's trackball. Written as part of the "flip" demo to demonstrate the Silicon Graphics (now SGI) hardware.

Brunt, P., 2010. GLGE: WebGL for the lazy. http://www.glge.org/.

Burtnyk, N., Khan, A., Fitzmaurice, G. and Kurtenbach, G., 2006. Showmotion: camera motion based 3d design review. Proceedings of the 2006 symposium on Interactive 3D graphics and games pp. 167–174.

Burtnyk, N., Khan, A., Fitzmaurice, G., Balakrishnan, R. and Kurtenbach, G., 2002. StyleCam: Interactive Stylized 3D Navigation using Integrated Spatial & Temporal Controls. Vol. 4Number 2, ACM, pp. 101–110.

Callieri, M., Leoni, C., Dellepiane, M. and Scopigno, R., 2013. Artworks narrating a story: a modular framework for the integrated presentation of three-dimensional and textual contents. In: Web3D, 18th International Conference on 3D Web Technology.

Chen, M., Mountford, S. J. and Sellen, A., 1988. A study in interactive 3-d rotation using 2-d control devices. In: Proceedings of the 15th annual conference on Computer graphics and interactive techniques, SIGGRAPH '88, ACM, New York, NY, USA, pp. 121–129.

Christie, M., Olivier, P. and Normand, J.-M., 2008. Camera control in computer graphics. Computer Graphics Forum 27(8), pp. 2197–2218.

Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F. and Scopigno, R., 2005. Batched multi triangulation. In: Proceedings IEEE Visualization, IEEE Computer Society Press, Conference held in Minneapolis, MI, USA, pp. 207–214.

DeLillo, B., 2009. WebGLU: A utility library for working with WebGL . http://webglu.sourceforge.org/.

Dell Unto, N., Ferdani, D., Leander, A.-M., Dellepiane, M., Callieri, M. and Lindgren, S., 2013. Digital reconstruction and visualization in archaeology case-study drawn from the work of the swedish pompeii project. In: Digital Heritage 2013 International Conference, IEEE, pp. 621–628.

Di Benedetto, M., Ganovelli, F., balsa Rodriguez, m., Jaspe Villanueva, A., Gobbetti, E. and Scopigno, R., 2014. exploremaps: Efficient construction and ubiquitous exploration of panoramic view graphcs of complex 3d environments. In: Computer Graphics Forum, 33(2), 2014. Proc. Eurographics 2014, To appear.

Di Benedetto, M., Ponchio, F., Ganovelli, F. and Scopigno, R., 2010. Spidergl: a javascript 3d graphics library for next-generation www. In: Proceedings of the 15th International Conference on Web 3D Technology, Web3D '10, ACM, New York, NY, USA, pp. 165–174.

Dirksen, J. (ed.), 2013. Learning Three.js: The JavaScript 3D Library for WebGL. Packt Publishing.

Don Brutzmann, L. D., 2007. X3D: Extensible 3D Graphics for Web Authors. Morgan Kaufmann.

Evans, A., Romeo, M., Bahrehmand, A., Agenjo, J. and Blat, J., 2014. 3d graphics on the web: A survey. Computers & Graphics 41(0), pp. 43 – 61.

Fitzmaurice, G., Matejka, J., Mordatch, I., Khan, A. and Kurtenbach, G., 2008. Safe 3d navigation. In: Proceedings of the 2008 symposium on Interactive 3D graphics and games, I3D '08, ACM, New York, NY, USA, pp. 7–15.

Hachet, M., Decle, F., Knodel, S. and Guitton, P., 2008. Navidget for easy 3d camera positioning from 2d inputs. In: 3D User Interfaces, 2008. 3DUI 2008. IEEE Symposium on, pp. 83 –89.

Hanson, A. J. and Wernert, E. A., 1997. Constrained 3d navigation with 2d controllers. In: Proceedings of the 8th conference on Visualization '97, VIS '97, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 175–ff.

Henriksen, K., Sporring, J. and Hornbæk, K., 2004. Virtual trackballs revisited. IEEE Transactions on Visualization and Computer Graphics 10(2), pp. 206–216.

Jankowski, J. and Decker, S., 2012. A dual-mode user interface for accessing 3d content on the world wide web. In: Proceedings of the 21st international conference on World Wide Web, WWW '12, ACM, New York, NY, USA, pp. 1047–1056.

Kay, L., 2009. SceneJS. http://www.scenejs.com.

Khronos Group, 2009a. Khronos: Open Standards for Media Authoring and Acceleration.

Khronos Group, 2009b. OpenGL ES - The Standard for Embedded Accelerated 3D Graphics.

Khronos Group, 2009c. WebGL - OpenGL ES 2.0 for the Web.

McCrae, J., Mordatch, I., Glueck, M. and Khan, A., 2009. Multiscale 3d navigation. In: Proceedings of the 2009 symposium on Interactive 3D graphics and games, I3D '09, ACM, New York, NY, USA, pp. 7–14.

Raggett, D., 1995. Extending WWW to support platform independent virtual reality. Technical Report.

Scott, W. R., Roth, G. and Rivest, J.-F., 2003. View planning for automated three-dimensional object reconstruction and inspection. ACM Comput. Surv. 35(1), pp. 64–96.

Shoemake, K., 1992. Arcball: a user interface for specifying three-dimensional orientation using a mouse. In: Proceedings of the conference on Graphics interface '92, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 151–156.

Sons, K., Klein, F., Rubinstein, D., Byelozyorov, S. and Slusallek, P., 2010. Xml3d: Interactive 3d graphics for the web. In: Proceedings of the 15th International Conference on Web 3D Technology, Web3D '10, ACM, New York, NY, USA, pp. 175–184.

Wan, M., Dachille, F. and Kaufman, A., 2001. Distance-field based skeletons for virtual navigation. In: Proceedings of the conference on Visualization '01, VIS '01, IEEE Computer Society, Washington, DC, USA, pp. 239–246.

Cutting Freely: Occlusion-Aware Surface Processing

Submission No. 0166

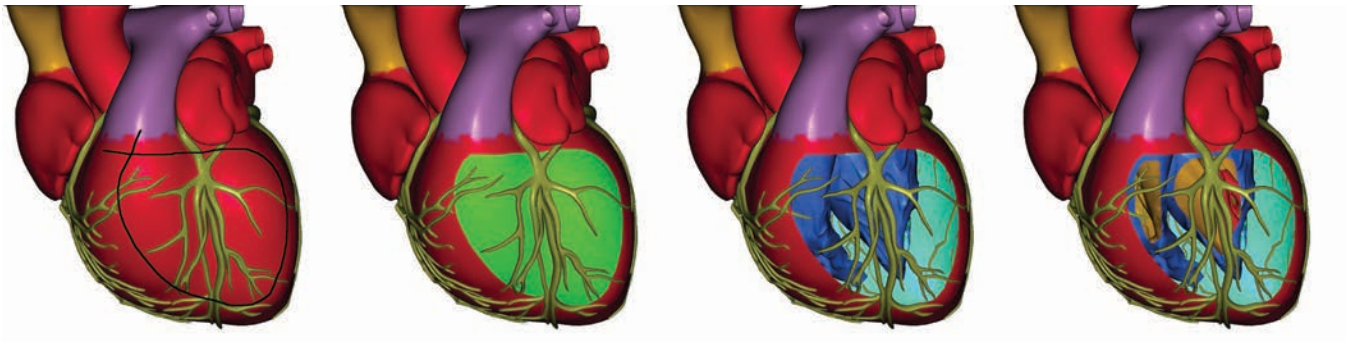


Figure 1: *Occlusion-aware generation of a cutaway. From left to right: User-drawn curve, selected region (shaded green), region cut out to reveal interior, final cutaway (after applying several such steps).*

Abstract

User-guided surface selection operations which are straightforward in a plane become challenging on non-strictly convex surfaces because of self-occlusions. Since the occlusions change with the view, users have to alternate between moving to an unobstructed view and performing those operations. Our method enables operations like selecting or cutting in a single view, by mapping these curves or areas as continuous projections onto the object's surface, unaffected by occlusions to guarantee a seamless brush stroke or a manifold cut. By default the projected solution includes the farthest surface layer, but if that is not the desired layer, users can roll back and forth through other solutions. We show that this enables a number of applications in an entirely different and easy way, of which we show a few as examples, creating illustrative cutaways from nested models, animating them and removal of spurious interior artifacts from isosurface meshing.

Keywords: occlusion-aware, surface selection, geometry processing

Concepts: •**Computing methodologies** → *Mesh geometry models;*

1 Introduction

Selecting a subset of the surface of a model is a very useful feature for 3D designers. One important application is inspecting multi-components models by generating cutaways: illustrations where the model is cut to reveal the interior or occluded parts. The feature also supports many surface manipulation scenarios, such as to blend out occluding parts to paste a texture, or cutting an outer layer to remove inner spurious primitives (useful after isosurface meshing). Other than cutting, the subset can be deformed, extruded, simplified, leading to many other exciting applications.

However, drawing a boundary on an object with moderate or high depth complexity is non-trivial. Occlusions can make it impossible in some cases to draw a curve that adapts to the surface from a single viewpoint. Tools in current 3D design software products only permit simple strokes to select a region.

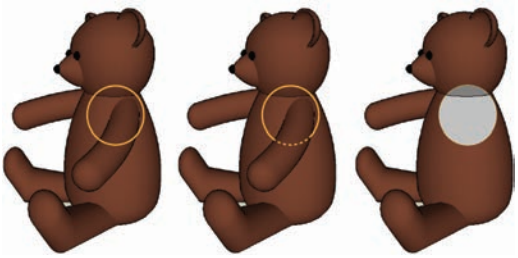


Figure 2: *Occlusion aware surface cut from user-drawn 2D curve. (a) 2D curve. (b) Projected, closed curve on the bear surface. (c) Occlusion-aware cut in the model.*

We propose a new method for generating view-independent cuts on a surface, which are inferred from a user-drawn curve. This curve is drawn in screen space, however our method makes the selection boundary follow the object's surface instead of moving onto occluding parts. To handle arbitrary occlusions on a manifold, we use an illustration buffer [Carnecky et al. 2013]. Figure 2 gives an example of how the bear's arm can be amputated without having to rotate the object into a view where the arm does not occlude the body.

Our main contributions are:

- An algorithm which creates manifold boundaries and marking of their contained surface subsets on 3D meshes with arbitrary depth complexity from a single input loop in the view plane, without requiring pre-processing or user-fitted primitives.
- An acceleration data structure which constructs efficiently to enable interactive performance of this algorithm and permits extension to point clouds and deforming meshes.

SIGGRAPH 2016 Posters, July 24-28, 2016, Anaheim, CA
ISBN: 978-1-4503-ABCD-E/16/07
DOI: http://doi.acm.org/10.1145/9999997.9999999

In Section 2, previous related approaches are summarized and reviewed. We explain the algorithm in Section 3. Connectivity operations in the screen space data structure are described in Section 4. We show our results and discuss them in Section 5 and conclude in Section 6 with a future outlook.

2 Related Work

There is a plenty of previous research concerning cutting and segmenting 3D models. Generating curves on 3D meshes has been used in many applications, such as creating cut views, segmentation (e.g. [Au et al. 2011]), and non-photorealistic rendering (e.g. [JUDD et al. 2007], [COLE et al. 2008], [DECARLO et al. 2003]). These methods can be generally categorized as data driven or user driven. We focus on the user driven aspect, like our algorithm does.

[Fan et al. 2011] introduce a painting brush for segmentation that allows the user to progressively select a region of interest. [Meng et al. 2011] present iCutter, which constructs a scalar field on the surface induced from a user-drawn stroke and finds the best isoline based on centerness and concavity. They aim at meeting the user's intention, but our method is more "faithful" to the 2D drawn curve, and thus gives the user full control over the cut boundary. [PINDAT et al. 2013] drill holes into layers to reveal inner objects. [LAMAR et al. 2001] and [WANG et al. 2005], extend the *Magic Lenses* [BIER et al. 1993], which permits users to magnify interesting parts and push away occluding parts. [MCGUFFIN et al. 2003] and [CORREA et al. 2006] provide cutting tools for volumetric data. All those approaches assume that the model is decomposed into pre-defined connected components. The sketching interface of [IGARASHI et al. 1999], and the cross-sections texturing of [OWADA et al. 2004] use a silhouette-intersecting stroke drawn by the user to specify the cut. They assume that the 2D stroke does not project on occluding parts of the surface. [Kndel et al. 2009] recognize four gestures from the user: line, corner, circular, and ribbon, with each gesture corresponding to a cut shape. Unlike this approach, cut boundaries made by our tool can take any arbitrary shape the user desires.

3 The Occlusion-Aware Selection Algorithm

Our method can be applied to arbitrary meshes in 3D with a manifold surface, but it is beneficial especially for objects with high depth complexity. The surface can contain holes and its triangles can even self-intersect as long as it does not affect the mesh connectivity.

The user draws a 2D curve in screen space, which we use as input to generate the selection boundary. The algorithm consists of the following two main steps. First, we convert this 2D curve to a 3D curve which maps onto the surface of the model. There is a default unique mapping, but other mappings may exist and are determined as well, which can be presented as choice to the user. Then, the region bounded by the chosen 3D curve is marked as input for further processing.

3.1 Mapping the Curve From Screen Space to Model

The user draws a 2D input curve $\hat{C}$ in the view plane which projects onto the object. In order to determine the selection boundary for the object, we need to map $\hat{C}$ to the desired 3D curve C on the surface of the model represented by a point set P. We require C to be closed since we use it to delimit a subset of the object's surface. There may not be a single solution but a *candidate set* $\mathbb{C}$ of such curves projecting to $\hat{C}$ and some of them may be entirely invisible, that is backfacing. So we first restrict the choice to a subset $\mathbb{C}_{vis} \subseteq \mathbb{C}$ of (partially or fully) visible candidate curves. Then, we order the set $\mathbb{C}_{vis}$ by increasing depth of their respective farthest visible curve segment. We define the default solution $C \in \mathbb{C}_{vis}$ as its last element, as shown in Figure 3. The user can always roll through this *depth ordered set* to select a nearer $C_i \in \mathbb{C}_{vis}$. This is for the that case C is not the desired choice, e.g. if that curve is barely visible, or if it does not represent the intended cut.

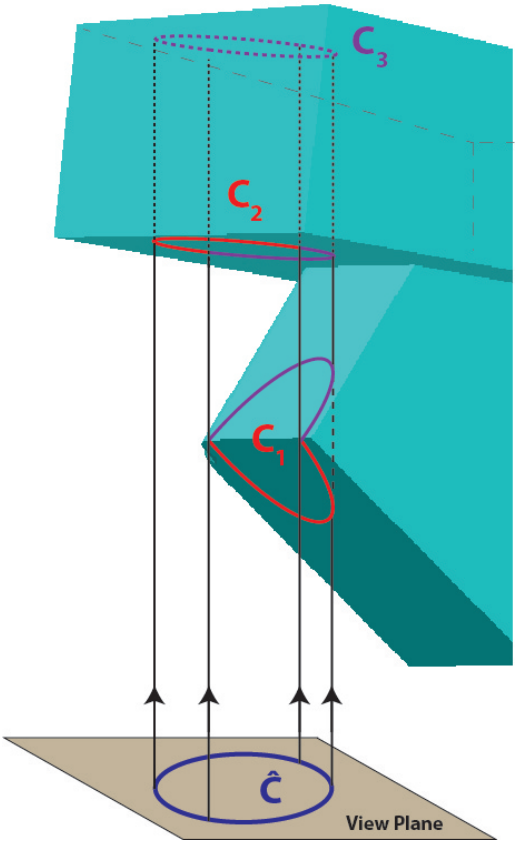


Figure 3: *The blue curve $\hat{C}$ drawn on the view plane is projected onto the model surface as P_C, the union of points of the three curve segments $C_{1..3}$. The curve segments are colored red (visible), or magenta (occluded), dotted if backfacing. The candidate set $\mathbb{C} = \{C_1, C_2, C_3\}$ and its visible subset is $\mathbb{C}_{vis} = \{C_1, C_2\}$, with the default (farthest) selection boundary $C = C_2$. C_1 remains as additional choice.*

We define $P_C = \{\cup P_i | \hat{p}_i \in \hat{C}\}$. Intuitively, P_C is the intersection of the projecting curve $\hat{C}$ that stencils the boundary of the object in view direction. Then, $\mathbb{C} \in P_C$ and therefore $C \in P_C$.

To determine C in P_C, we rely on the fact that P_C can be decomposed into a disjoint set of curve segments $\mathbb{S} \equiv P_C$ because the curves lie on a manifold. We define the $S_i \in \mathbb{S}$ as the maximum connected set of points in P_C such that each $p_j \in S_i$ is occluded by a constant number of points of the model as seen from the camera (or *visible* if that number is zero). See Figure 4 for an example of this decomposition into multiple disjoint curve segments and how those compose the set of closed curves $\mathbb{C}$. By our above definition, all $C_i \in \mathbb{C}_{vis}$ contain a visible $S_i \in C_i$.

The two above-mentioned properties of C (closed and containing the farthest visible curve segment) allow for its easy determina-

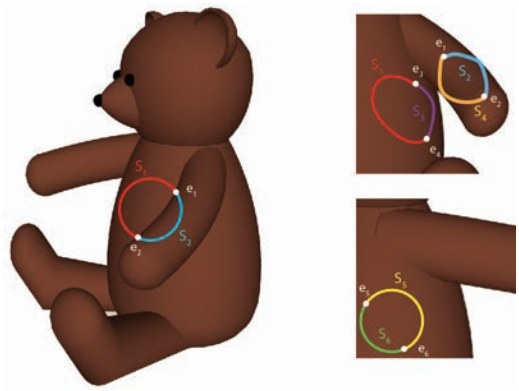


Figure 4: *Left: $\hat{C}$ is projected onto the model, intersecting with its surface in curve segments $S_{1..6}$. Only S_1 and S_2 are visible from that view point. Right: The remaining curve segments become visible from other view points (Above: Front, Below: Back). $C_1 = S_1 \cup S_3$, $C_2 = S_2 \cup S_4$ and $C_3 = S_5 \cup S_6$ each form a closed curve. The visible set is $\mathbb{C}_{vis} = C_1, C_2$, with the default boundary $C = C_1$.*

tion: Start from the farthest visible segment S_0 and add incident segments in $\mathbb{S}$ to a set until it forms a closed curve C_0. While $\hat{C}_0 \subset \hat{C}$, repeat this process starting with the farthest visible segment S_i, $\hat{S}_i \cup \hat{C}_0 = \emptyset$. This yields a surjectively mapping of C_0 onto $\hat{C}$, since C_0 may contain backfacing curve segments. Our algorithm handles the case of $\hat{C}$ intersecting silhouettes of the object in the view plane without any further modifications, since it follows the adjacent backfacing curve segments. For the non-default solutions, additional curves $C_i \in \mathbb{C}_{vis}$ can be generated by processing the remaining curve segments, until there are no more visible ones. Since we require the curve to be closed, it cannot intersect a hole in a bounded manifold. Therefore, open curves are ignored.

3.2 Marking the Selected Surface Region

We now have found the selection boundary C, which is either the default solution, or a nearer solution chosen by the user. C may be a set of multiply connected curves, and it partitions the object P into at least two subsets. In order to decide which subsets of the object should be marked, we need to consider the orientation based on $\hat{C}$. Any visible point $p_i \in C$ assigns an orientation to adjacent visible points in P based on whether it is located inside the closed loop $\hat{C}$ in screen space. The propagation of the orientation on the surface of the object using the illustration buffer is described in Section 4. If C cuts a handle of the object, the propagation will yield a single orientation and that would lead to selecting the entire manifold. In that case, a second curve is required to bound a region together with the first curve.

4 Implementation in a Discrete Acceleration Structure

In order to determine the curve segments, we have to intersect the curve drawn in screen space with the 3D model. However, this requires to search all its N primitives, for each curve fragment. A tree would require slow $O(NlogN)$ construction and $O(|\hat{A}|logN)$ per curve, with $\hat{A}$ as the screen space area inside the selected curve, plus any traversed backfacing parts of surface. Instead, we construct what we call a *connectivity buffer* based on a screen space buffer structure: We require a functionality similar to the illustration buffer [Carnecky et al. 2013], but only store unsorted list of fragments per pixel together with their references to the original triangle. This reduces construction against traversal cost, which is beneficial for our case since only a subset of fragments will be traversed.

Our proposal is based on the following reasons: Construction time for the buffer is much faster with $O(N)$, on top of that N is smaller since culled to screen space, so the construction time is almost output-sensitive. Although the buffer has to be constructed anew for each view in which the user draws a curve, this coincides with user interaction and is therefore not noticable. The critical part that thappens after completed user interaction - marking the selected subset of the surface - is then much faster with $O(\|\hat{A}\|logk)$. Keeping the fragments of all layers in the buffer permits us to quickly compute transparency or rolling through all possible solutions of curves. On top of that, our buffer structure is easily extensible to contain point clouds or allow for surface deformations, since we compute it for each relevant view.

Note that our implementation is not yet optimized: Only the construction of the connectivity buffer is done on the GPU and since we follow connectivity currently on the CPU, most of the time on our proof-of-concept implementation is wasted by CPU-GPU transfers that will eventually become redundant. Potentially it can become as fast as order independent transparency which uses the illustration buffer.

4.1 Connectivity Buffer Construction

We construct the buffer structure by simply rendering the model mesh from the viewpoint of the user with the depth test disabled, so that we keep all projected fragments. A fragment tuple stores the distance from the view plane together with a reference to its incident triangle. In each pixel, the visible fragment, i.e. the closest to the view plane, is moved to the front of the fragments list. Since there can be several visible fragments sharing the nearest vertex, anyone of them will serve as it is used as a starting point for tracing. Figure 5 illustrates the data structure and its construction process.

Unlike the illustration buffer, we do not accumulate connectivity information but locate it while tracing. The time required to dynamically locate connectivity is more than offset because tracing the curve and interior surface area only ever affects a subset of the fragments. Binning the mesh into fragment lists on the other hand is faster than traversing the mesh itself, since the relevant pixels can be looked up directly. We consider fragments stored in the same pixel as connected only if their referenced triangles share an edge. Two fragments in adjacent pixels, on the other hand, are only connected if they reference the same triangle (see Figure 6).

4.2 Tracing Curves and Marking Interior Surface Areas

Since P_C is now discretized into fragments, curve segments on the mesh can be traced by following "connected" fragments (8-connected pixels) and subsequently join these segments to curves. A curve is discarded it does not end at the same fragment it started from, i.e. open curve, since then it does not project surjectively on the user-drawn curve. Closed curves are sorted by the depth of the farthest visible fragment.

After the boundary curve is determined (bu default, or by the user), our algorithm marks the surface area interior to the curve w.r.t. screen space. The output is a list of triangles, but some triangles may be intersected by the curve. First, we split those triangles which the curve intersects, by simply connecting the intersection

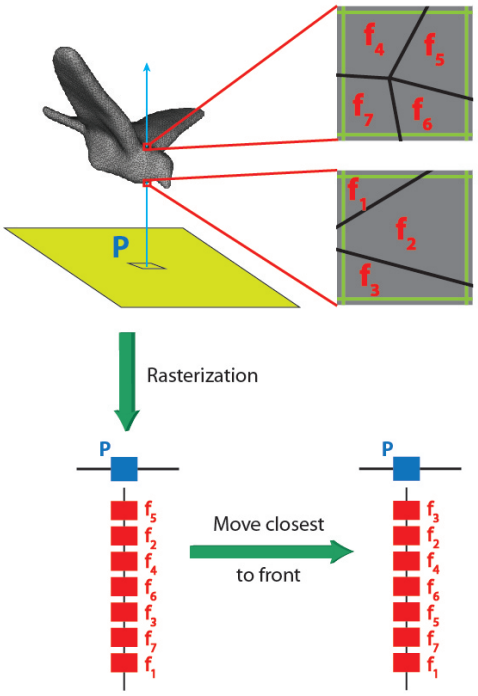


Figure 5: *In the rasterization stage, pixel P accumulates its fragments in a list and the closest to the camera moved to its front.*

points with their edges, and mark the two halves as interior and exterior, respectively. Then, we initialize a stack with the interior triangles and connect unoriented triangles recursively while unconnected triangles remain. The stack forms the output as list of triangles representing the interior area of the curve.

When dealing with self-intersecting components which are not connected (e.g. ear sticking out of bear head in Figure 7), the user might also want to remove such components that fall into the bounded region. We leave *remove-intersecting* as a feature for the user to toggle.

5 Results and Discussion

We tested our method on various data sets, including self-intersecting meshes. In the following subsections, we explain how

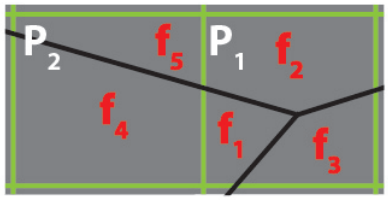


Figure 6: *Connectivity definitions: Three triangles are projected on two pixels. Fragment f_1 in pixel P_1 is connected to both f_2 and f_3 because it shares an edge with each of them. It is also connected to f_4 in pixel P_2, because they have the same incident triangle.*

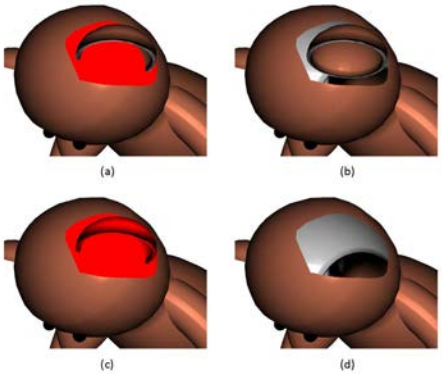


Figure 7: *Top row:* remove-intersecting=off*: (a) Bounded region marked in red. (b) Cut does not affect intersecting ear. Bottom row:* remove-intersecting=on*: (c) Bounded region marks ear as well. (d) Cut removes ear.*

the algorithm proved to be useful in several applications and scenarios, due to its occlusion-awareness property and/or its interactive performance.

5.1 Model Inspection and Generation of Cutaways

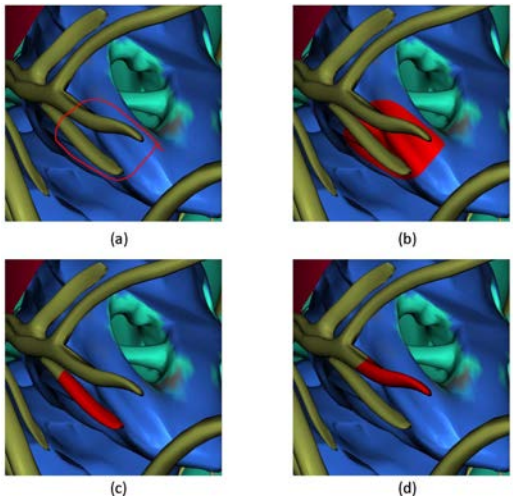


Figure 8: *Inspecting a heart model. The user draws a curve (a), and then he rolls among different candidate regions to select back wall (b) or one of the occluding arteries (c,d), sorted descendingly by depth.*

The obvious application of our method is to better explore models with significant depth complexity. Many models are self-occluding, and some even contain very numerous interleaving components (e.g. wires, tree branches, or blood vessels) which block the vision of the viewer from most or all perspectives. Using our algorithm, the user can quickly reveal the hidden areas by cutting the occluding parts with simple curve sketching. Figure 8 shows an example where several candidate-for-selection regions are detected with a single drawn 2D curve. The user can then simply roll back and forth between those to choose the desired region.

Once experts have inspected such complex models, they may wish to show selected features to non-expert users. Cutaways and transparent illustrations are very good tools to expose and high-light

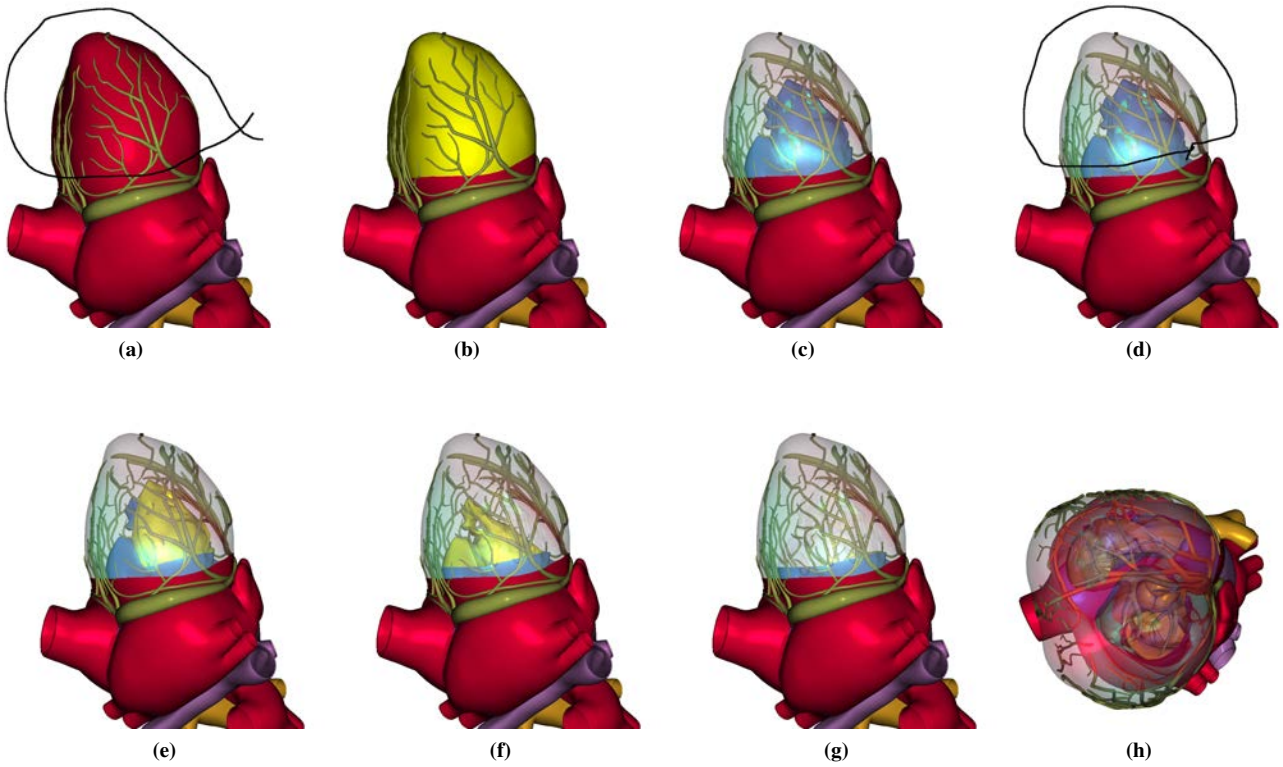


Figure 9: *Generating a transparent illustration of the heart model: (a) The user draws a curve, for which its interior region is marked yellow (b), and turned transparent (c). A second curve is drawn (d). The user selects two other regions from that curve (e),(f) which are also turned tansparent. The final illustration (g), also from another view point (h).*

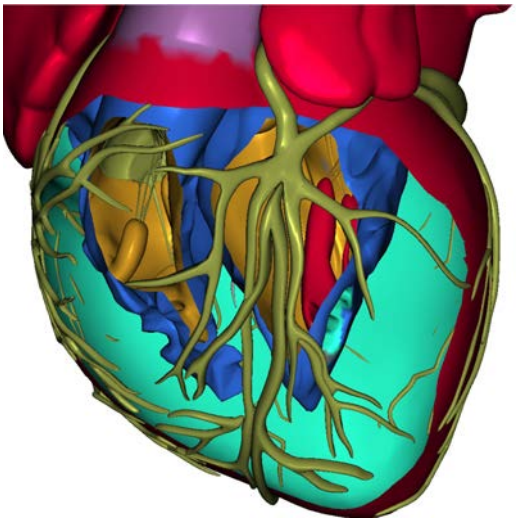


Figure 10: *Cutaway illustration of a heart model.*

details of such objects and have been used e.g. for centuries in medical instruction and biology. Those drawings reveal the inner components by manipulating the visibility of the outer layers. Our algorithm locates projections on all different layers, and is therefore ideally suited to quickly create such illustrations. Figure10 shows a cutaway of the heart model and Figure9 shows a transparent illustration, together with the steps used to create it. We were able to

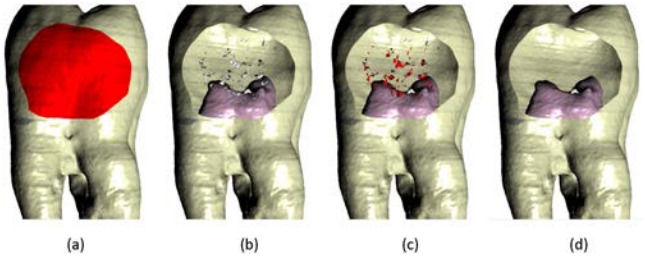
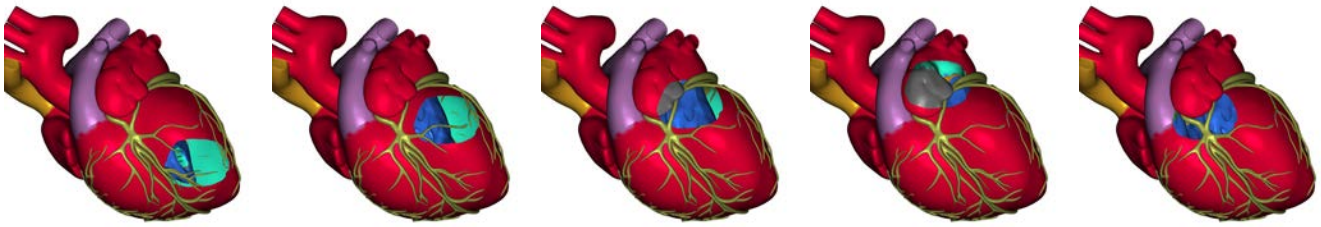


Figure 12: *(a) A tooth mesh reconstructed from an isosurface of a CT scan, with the selected region marked red. (b) The selected region is cut, revealing a number of spurious connected components or triangles. (c) The user cleans the model by selecting groups of these artifacts. (d) All artifacts have been removed and the cut can be reversed.*

create each of these drawings just from a single viewpoint.

5.2 Boundary Animation

We can even go one step further and animate our cutaways. Frames of an animation which displays a cutaway dragged along the surface, oblivious to any occluding objects, are shown in Figure 11. While the cut boundary retains its shape in screen space, it is projected automatically on the current layer on the model.



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

Figure 11: *Cut boundary animation: A few frames of a cut that is dragged*

5.3 Surface Editing and Manipulation

The algorithm provides a smart tool for 3D selection, since it is not hindered by occlusions. Other than cutting, the user can perform all kinds of surface operations to the marked region, e.g. deform, replicate, extrude, and many more. The interactive nature of the algorithm makes it an efficient support tool for such surface manipulations. As an example application we show how meshes created from isosurfaces (resulting e.g. from CT scans) can be cleaned efficiently: Figure12 shows a mesh, where the user first cuts the outer layer in order to reach the hidden spurious primitives, which are then picked and removed by a simple selection tool. After the model interior has been cleaned, the cutting operation is simply undone and the model is ready for e.g. 3D printing.

6 Conclusion

We introduced a method to select subsets on meshes without having to remove occlusions of the mesh itself or other objects. The meshes may self-intersect and contain holes. The results show that users can easily create these curves and roll back and forth if there exist choices on several layers. Objects which intersect with the curve can be selected as well together with the primarily targeted surface. The operations on the selection boundary or its interior surface that can be executed subsequently are plenty: Painting, extrusion, cutting, modifying surface attributes (e.g. transparency) - we demonstrate the last two as examples as well as animating illustrative cut boundaries.

Limitations: At the moment, our algorithm does not support non-manifold meshes, i.e. containing edges with more than two incident triangles. This is not a big limitation since the kind of meshes used for such processing should be clean in that respect anyway. If required, tracing could bifurcate for such cases, however resulting in combinatorial complexity. Our algorithm also does not support drawing curves through holes. This can be easily fixed by storing boundaries and using them to complete the curves. We do not currently detect if a curve is drawn through a handle. The user has to take care to draw additional curves such that an interior surface is partitioned off.

Future work:

Our discrete buffer data structure permits easy incorporating of all kinds of surface representations, as long as they provide manifold connectivity. It can be easily extended to using point clouds, e.g. with the discretized surface construction proposed by [Radwan et al. 2014], which creates the implied connectivity.

Converting the code into a Blender plugin would allow simple application of existing surface operations, like extrusion, mesh simplification, painting/texturing and so on. Similarly, creation of plugins for other operations on surface subsets like mesh mixing is straightforward.

Extension to segmentation: The user-drawn 2D curve could serve as guidance to fitting a 3D curve with local optimization criteria, e.g. based on concavity isolines as in [Fan et al. 2011].

7 Acknowledgements

References

ATTENE, M., FALCIDIENO, B., AND SPAGNUOLO, M. 2006. Hierarchical mesh segmentation based on fitting primitives. *Vis. Comput. 22*, 3 (mar), 181–193.

AU, O. K.-C., ZHENG, Y., CHEN, M., XU, P., AND TAI, C.-L. 2011. Mesh segmentation with concavity-aware fields. *Visualization and Computer Graphics, IEEE Transactions on 18*, 7 (July), 1125 – 1134.

BIER, E. A., STONE, M. C., PIER, K., BUXTON, W., AND DEROSE, T. 1993. Toolglass and magic lenses: The see-through interface. *Proceedings of ACM SIGGRAPH 93*, 73–80.

BRUCKNER, S., AND GRO LLER, M. E. 2005. Volumeshop: An. *Proceedings of IEEE Visualization*.

BURNS, M., AND FINKELSTEIN, A. 2008. Adaptive cutaways for comprehensible rendering of polygonal scenes. *ACM Trans. Graph. 27*, 5 (dec), 154:1–154:7.

CARNECKY, R., FUCHS, R., MEHL, S., JANG, Y., AND PEIKERT, R. 2013. Smart transparency for illustrative visualization of complex flow surfaces. *IEEE Transactions on Visualization and Computer Graphics 19*, 5, 838–851.

CHEN, X., GOLOVINSKIY, A., AND FUNKHOUSER, T. 2009. Benchmark for 3d mesh segmentation. *ACM Trans. Graphics 28*, 3.

COLE, F., GOLOVINSKIY, A., LIMPAECHER, A., BARROS, H. S., FINKELSTEIN, A., FUNKHOUSER, T., AND RUSINKIEWICZ, S. 2008. Where do people draw lines? *ACM Transactions on Graphics 27*, 3.

CORREA, C., SILVER, D., AND CHEN, M. 2006. Feature aligned volume manipulation for illustration and visualization. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (September/October), 1069–1076.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive contours for conveying shape. *ACM Transactions on Graphics 22*, 3.

DIEPSTRATEN, J., WEISKOPF, D., AND ERTL, T. 2003. Interactive cutaway illustrations. *Computer Graphics Forum 22*, 3 (September), 523–532.

EISEMANN, E., PARIS, S., AND DURAND, F. 2009. A visibility algorithm for converting 3d meshes into editable 2d vector graphics. *ACM Trans. Graph. 28*, 3 (jul), 83:1–83:8.

FAN, L., LIC, L., AND LIU, K. 2011. Paint mesh cutting. *Computer Graphics Forum 30*, 2 (Aprit), 603–612.

FEINER, S., AND SELIGMANN, D. D. 1992. Cutaways and ghosting: Satisfying visibility constraints in dynamic 3d illustrations. *The Visual Computer*, 292–302.

GOLOVINSKIY, A., AND FUNKHOUSER, T. 2008. Randomized cuts for 3d mesh analysis. *ACM Transactions on Graphics 27*, 5 (December).

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3d freeform design. *Proceedings of ACM SIGGRAPH*, 409–416.

JUDD, T., DURAND, F., AND ADELSON, E. H. 2007. Apparent ridges for line drawing. *ACM Transactions on Graphics 26*, 3.

KNDEL, S., HACHET, M., AND GUITTON, P. 2009. Interactive generation and modification of cutaway illustrations for polygonal models. *SG '09 Proceedings of the 10th International Symposium on Smart Graphics*, 140 – 151.

KRGER, J., SCHNEIDER, J., AND WESTERMANN, R. 2006. Clearview: An interactive context preserving hotspot visualization technique. *IEEE TVCG 12*, 5, 941–948.

LAMAR, E., HAMANN, B., AND JOY, K. I. 2001. A magnification lens for interactive volume visualization. *9th Pacific Conference on Computer Graphics and Applications*, 223–232.

LI, W., RITTER, L., AGRAWALA, M., CURLESS, B., AND SALESIN, D. 2007. Interactive cutaway illustrations of complex 3d models. *ACM Trans. Graph. 26*, 3 (jul).

LIEN, J.-M., AND AMATO, N. M. 2006. Approximate convex decomposition of polygons. *Special Issue on the 20th ACM Symposium on Computational Geometry 35*, 1-2 (August), 100–123.

MCGUFFIN, M. J., TANCAU, L., AND BALAKRISHNAN, R. 2003. Using deformations for browsing volumetric data. *Proceedings of IEEE Visualization*, 401–408.

MENG, M., FAN, L., AND LIU, L. 2011. icutter: a direct cut-out tool for 3d shapes. *Computer Animation and Virtual Worlds 22*, 4 (August).

OWADA, S., NIELSEN, F., OKABE, M., AND IGARASHI, T. 2004. Volumetric illustration: designing 3d models with internal textures. *ACM Transactions on Graphics 23*, 3 (August), 322–328.

PINDAT, C., PIERTRIGA, E., CHAPUIS, O., AND CLAUDE, P. 2013. Drilling into complex 3d models with gimlenses. *VRST '13 Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*, 223–230.

RADWAN, M., OHRHALLINGER, S., AND WIMMER, M. 2014. Efficient collision detection while rendering dynamic points. *Proceedings of the 2014 Graphics Interface Conference* (May), 25–33.

REZK-SALAMA, C., AND KOLB, A. 2006. Opacity peeling for direct volume rendering. *Computer Graphics Forum 25*, 3, 597–606.

VIOLA, I., KANITSAR, A., AND GRLLER, M. E. 2004. Importance-driven volume rendering. *Proc. of IEEE Visualization*, 139–145.

VIOLA, I., KANITSAR, A., AND GROLLER, E. 2005. Importance driven feature enhancement in volume visualization. *IEEE Transactions on Visualization and Computer Graphics 11*, 4, 408–418.

WANG, L., ZHAO, Y., MUELLER, K., AND KAUFMAN, A. E. 2005. The magic volume lens: An interactive focus+context technique for volume rendering. *Proceedings of IEEE Visualization*, 367–374.

High-Quality Point-Based Rendering Using Fast Single-Pass Interpolation

Markus Schütz
Institute of Computer Graphics
Vienna University of Technology
Vienna / Austria
mschuetz@potree.org

Michael Wimmer
Institute of Computer Graphics
Vienna University of Technology
Vienna / Austria
wimmer@cg.tuwien.ac.at

***Abstract*—We present a method to improve the visual quality of point cloud renderings through a nearest-neighbor-like interpolation of points. This allows applications to render points at larger sizes in order to reduce holes, without reducing the readability of fine details due to occluding points. The implementation requires only few modifications to existing shaders, making it eligible to be integrated in software applications without major design changes.**

***Index Terms*—Computer graphics, point clouds, WebGL**

I. Introduction

3D scanning methods such as laser scanning or photogrammetry produce enormous amounts of point cloud data. Unlike polygon meshes, point clouds do not contain connectivity between points, and surface normals are not always available. Due to the missing connectivity and normals, points are often rendered using screen-aligned squares or circles. If the size of these primitives is too small, holes appear, and if the size is too large, points will occlude each other and reduce the visibility of high-frequency features such as text.

This paper presents a method that allows using larger point sizes in order to avoid holes and at the same time, solve undesirable occlusions by performing a nearest-neighbor-like interpolation of points. The interpolation is achieved by rendering points as 3d shapes through manipulation of fragment depths. Additional passes are not required.

Our method can be seen as a trade-off between the performance of the commonly used screen-aligned square and circle primitives, and the high quality of multi-pass splatting algorithms.

II. Related Work

Related works include high-quality point-based rendering as well as fast Voronoi diagram generation methods.

A. High-Quality Splatting

Previous high-quality splatting methods for the GPU [1] require three rendering passes. First, a visibility pass builds a depth map with a small offset. The blending or attribute pass then builds a weighted sum of all attributes that pass the depth test. The last pass normalizes attribute values by dividing the weighted sum of attributes by the sum of weights. These methods also render points as oriented disks or ellipses. The results have a very high quality. The need for three rendering passes, however, significantly reduces performance, and rendering oriented disks requires normals, which are not always available.

Deferred Blending [2] is a GPU-accelerated method that is able to render opaque point clouds in a single geometry pass and an additional compositing pass. This method is also able to render simple transparency effects in a two-pass approach and higher-quality transparencies in 3 passes.

These methods have in common that they require multiple rendering passes. They achieve high-quality results with smoothly blended points at a high performance cost.

B. Voronoi Diagram Generation

The results of our method closely resemble Voronoi diagrams. In fact, the idea of rendering points as 3D shapes has already been used in previous works for fast generation of Voronoi diagrams. Kenneth et al. [3] create two-dimensional Voronoi diagrams by rendering points as cones and lines as tents with a cone at each corner.

Jump flooding [4] uses a flooding algorithm to generate Voronoi diagrams by repeatedly spreading pixels in a texture until the whole texture is filled. Instead of distributing pixels to their closest empty neighbors in each step, they are propagated over larger distances, thus reducing the number of necessary repetitions.

III. Interpolation Shader

This section covers the theory as well as implementation details of our interpolation shader.

The idea behind our method is similar to creating Voronoi diagrams by rendering points as cone meshes. [3]. However, instead of using meshes, points are rendered as view-aligned quads, as commonly used in point cloud renderers. The three-dimensional shape is achieved by adding an additional offset to the fragment depth values. This offset depends on the distance to the center of the quad and the type of weight function.

Figure 1 shows shapes produced by different weight functions. Weights are calculated for each fragment and subsequently used as an offset to the depth value. For spheres, the weight function is not defined for fragments outside the sphere's boundaries. These fragments are therefore discarded,

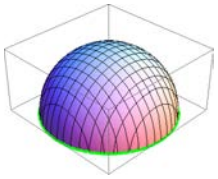
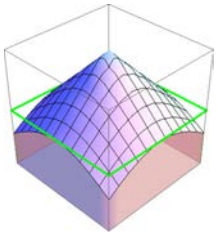
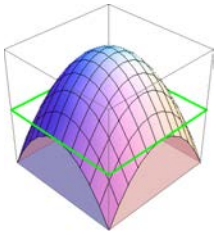
	Weight function	Shape
sphere	$\sqrt{1-(u^2+v^2)}$	
cone	$1-\sqrt{(u^2+v^2)}$	
paraboloid	$1-(u^2+v^2)$	

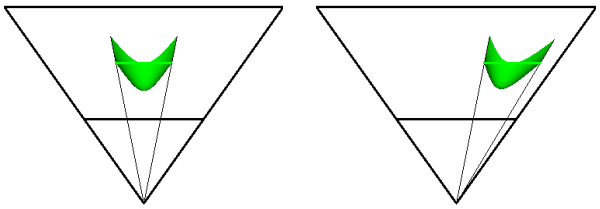
Fig. 1. Shapes produced by different weight functions. $u, v \in [-1, 1]$

resulting in circular shapes on screen. The cone and paraboloid weight functions are well defined for all fragments and can therefore be used for squares as well.

Point clouds are usually rendered with view-aligned rather than camera-facing quads. Applying weights as depth offsets leads to rendering distorted shapes because of the perspective projection of view-aligned quads. Figure 2 shows how using a paraboloid weight function results in rendering distorted paraboloids. In practice, this has shown to work fine and to significantly increase quality at a low cost despite the distortion. Figure 3 shows the same points rendered as simple view-aligned squares and paraboloids. In the latter case, the point closest to the camera is less likely to occlude all points behind it.

All of the listed weight functions reduce occlusion problems. There is a subtle difference, though, and we decided to chose the paraboloid function for some of its properties. First of all, it is the simplest one to calculate. It is also defined for all fragments, unlike the spherical function, and can therefore be used with square-shaped point primitives as well. But most importantly, the intersections between paraboloids at different distances remain straight, whereas the intersections of cones and spheres appear rounded.

All of the weight functions assume that the radius of the point is 1 and the generated weights range from -1 to 1. The final depth offset is obtained by multiplying the weight by the world-space point radius. If the world-space radius is not known, it can be approximated by taking the pixel size and inverting the projection, as described in Section III-A.



(a) distortion at center (b) distortion off-center

Fig. 2. The resulting shapes are distorted due to perspective projection.

.

A. Implementation

The implementation requires field of view in radians and screen height in pixels as additional uniform inputs to the vertex shader and the projection matrix as additional input to the fragment shader. Field of view and screen height are used to approximate the world-space point radius from the pixel size of the point primitive. The projection matrix is used to recalculate the projected depth after modifying the view-space depth value.

Even if the world-space point radius is known, it may still be necessary to approximate it from the pixel size instead. For example, since this method is sensible to overdraw, our implementation limits the point size to a maximum of 50 pixels. The pixel size is therefore no longer guaranteed to be the screen projection of the radius.

For the approximation, the projection factor from a world-space radius to screen-space pixel size is calculated. The pixel size is then divided by the projection factor to obtain an approximation of the world-space radius. This also allows integrating the algorithm into systems that use fixed or camera-dependent point sizes with no assumptions of point radii.

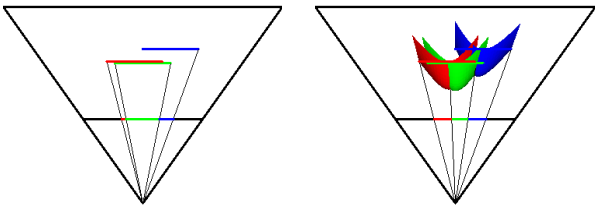
```
float projFactor = 1.0 / tan(fov / 2.0);
projFactor = projFactor / -vViewPos.z;
projFactor = projFactor * screenHeight / 2.0;
...
vRadius = gl_PointSize / projFactor;
```

The fragment shader provides coordinates that indicate the fragment position inside the point primitive. To calculate the weight, these coordinates have to be transformed from an interval of [0,1] to an interval of [-1,1]. The following code sample uses the paraboloid weight function.

```
float u = 2.0 * gl_PointCoord.x - 1.0;
float v = 2.0 * gl_PointCoord.y - 1.0;
float w = 1.0 - ( u*u + v*v );
```

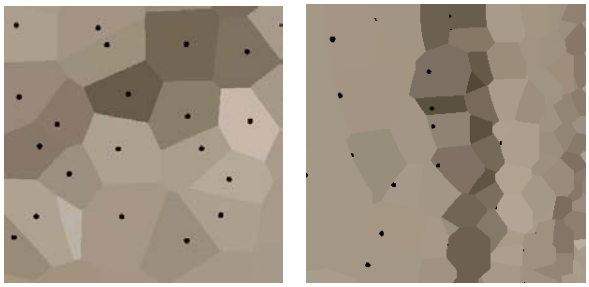
The weight is multiplied by the radius of the point and added to the screen-space depth value. The resulting position is then projected and its projected depth value is used as the new fragment depth.

```
vec4 pos = vec4(vViewPos, 1.0);
pos.z += w * vRadius;
pos = projectionMatrix * pos;
pos = pos / pos.w;
gl_FragDepthEXT = (pos.z + 1.0) / 2.0;
```



(a) without depth offset (b) with depth offset

Fig. 3. Top view showing (a) points occluding other points behind them and (b) using a fragment depth offset to reduce undesirable occlusions.



(a) front (b) steep angle

Fig. 4. Point centers are indicated by black dots. (a) Front view showing similarities of the results to a Voronoi diagram. (b) Similarities to Voronoi diagrams decrease at steep angles.

IV. Results and Limitations

In this section, we show images of our results and comparisons to screen-aligned squares and circles. We also compare results to a three-pass high-quality splatting method using screen-aligned circles because our datasets do not contain the normals necessary for rendering oriented splats.

The results of this method, as seen in Figure 4, show strong similarities to a Voronoi diagram.

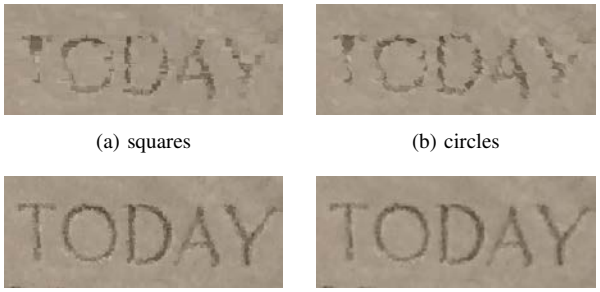
Figure 5 and 7 show images generated by the different rendering methods. Squares and circles both suffer from occlusions. Camera rotations also cause flickering as points change their order and occluding points suddenly become occluded points. The interpolation and high-quality splat-rendering modes do not suffer from this problem.

Due to its nearest-neighbor-like behavior, our method is as susceptible to noise as squares and circles. The high-quality splatting methods, on the other hand, blend multiple points together and therefore reduce the impact of noise, as shown in Figure 6.

V. Performance

All performance tests were done on a notebook with an Intel Core i7-4712MQ and a NVIDIA GTX 860M. We used WebGL and the Chrome web browser to render into a 1920x955 pixel canvas element.

Figure 8 shows frames per second (FPS) for different modes and point sizes. The size parameter is a multiplier. A value of 0 results in a size of 1 pixel. With a value of 1, pixel



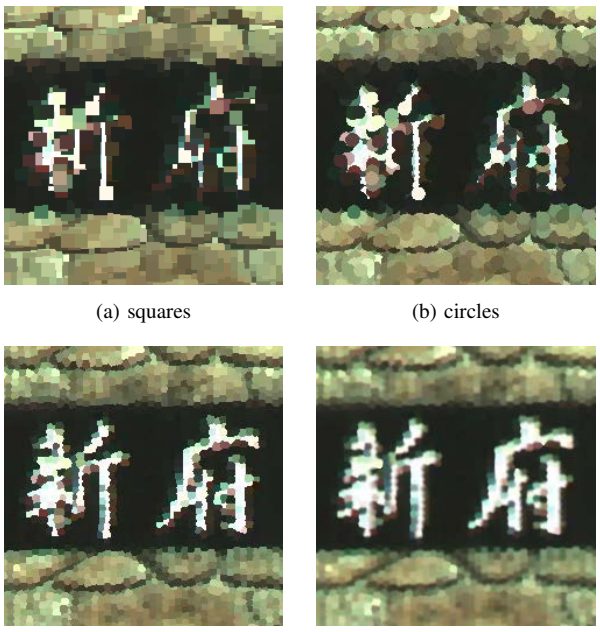
(a) squares (b) circles

(c) our method (d) high-quality splats

Fig. 5. Squares (a) and circles (b) suffer from occlusions. Our method (c) and high-quality splats (d) improve readability of high-frequency details such as text.

(a) our method (b) high-quality splats

Fig. 6. A limitation of our approach: It does not improve noisy datasets. High-quality splats are better suited in such cases.



(a) squares (b) circles

(c) our method (d) high-quality splats

Fig. 7. Improved readability of text with our method, comparable to high-quality splats.

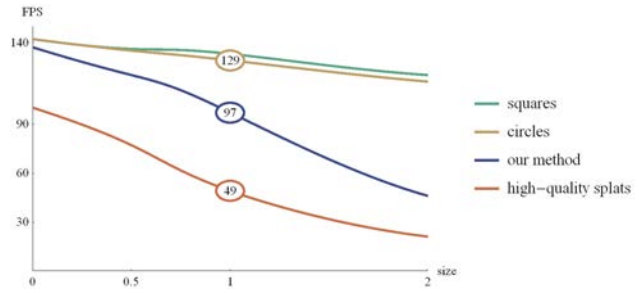


Fig. 8. Performance of squares, circles, interpolation and high-quality splats (in top-to-bottom order) in frames per second (FPS). A size factor of 1 covers holes while minimizing overdraw. Lower values cause holes while larger values increase overdraw. .

size is chosen in a way to close holes but minimize overdraw. Lower values lead to holes and larger values cause increasingly higher overdraw. Too much overdraw is problematic since interpolation and high-quality splatting depend on features that do not allow for early depth testing.

VI. Conclusion and Future Work

We have presented a single-pass method that significantly increases quality at a lower impact on performance than previous high-quality methods that require two or even three rendering passes. Implementation is simple and requires adding a few lines of code, as described in Section III-A, to existing shaders.

It is especially useful for close-up views of datasets with sharp features such as text or edges.

This method can be seen as a trade-off between the performance of screen-aligned squares and circles, and the high quality of multi-pass splatting algorithms.

This method was developed for the WebGL point cloud renderer Potree [5] with the help of the three.js library. [6]. A reference implementation of this method is available in the Potree github repository.

Manipulating the fragment depth can disable some GPU optimizations such as early depth testing. Other possible approaches to render points as paraboloids are geometry shaders and instancing. We did not explore these options since WebGL does not support geometry shaders at this time and instancing is not supported by three.js.

VII. Acknowledgements

This research was supported by the EU FP7 project HARVEST4D (no. 323567) [7]. The statue is part of the Arene de Lutece dataset, courtesy of HARVEST4D. The point cloud depicting the Japanese sign is courtesy of Anan Survey [8].

References

[1] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt, "High-quality surface splatting on today's gpus," in *Proceedings of the Second Eurographics / IEEE VGTC Conference on Point-Based Graphics*, ser. SPBG'05. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2005, pp. 17–24. [Online]. Available: http://dx.doi.org/10.2312/SPBG/SPBG05/017-024

[2] Y. Zhang and R. Pajarola, "Deferred blending: Image composition for single-pass point rendering," *Comput. Graph.*, vol. 31, no. 2, pp. 175–189, Apr. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.cag.2006.11.012

[3] K. E. Hoff, III, J. Keyser, M. Lin, D. Manocha, and T. Culver, "Fast computation of generalized voronoi diagrams using graphics hardware," in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 277–286. [Online]. Available: http://dx.doi.org/10.1145/311535.311567

[4] G. Rong and T.-S. Tan, "Jump flooding in gpu with applications to voronoi diagram and distance transform," in *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, ser. I3D '06. New York, NY, USA: ACM, 2006, pp. 109–116. [Online]. Available: http://doi.acm.org/10.1145/1111411.1111431

[5] M. Schütz, "Potree," http://potree.org, accessed: 2015-07-02.

[6] R. Cabello, "three.js," http://threejs.org/, accessed: 2015-07-02.

[7] "Harvest4d," https://harvest4d.org/, accessed: 2015-04-16.

[8] "Anan survey," http://anan.skr.jp/, accessed: 2015-02-14.

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

Volume 35 (2016), Number 2

Compressed Multiresolution Hierarchies for High-Quality Precomputed Shadows

Leonardo Scandolo, Pablo Bauszat, and Elmar Eisemann

Delft University of Technology, Netherlands

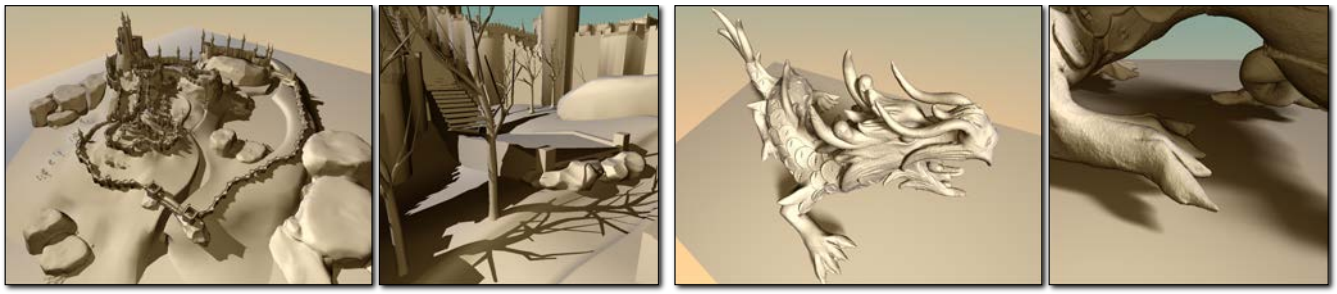


Figure 1: **Left:** High-quality shadows in a static large-scale environment rendered in 1 millisecond using a 32-bit shadow map with a resolution of *1.048.576 x 1.048.576* pixels. The shadow map is compressed from four terabytes down to 160.6 MB (26124:1 ratio) without loss of precision. **Right:** Precomputed soft-shadows from a high-detail model using 2.048 coherent shadow maps, each with a resolution of *2.048 x 2.048* pixels, rendered with 32 samples in 14 milliseconds and stored in 145 MB (227:1 ratio).

Abstract

The quality of shadow mapping is traditionally limited by texture resolution. We present a novel lossless compression scheme for high-resolution shadow maps based on precomputed multiresolution hierarchies. Traditional multiresolution trees can compactly represent homogeneous regions of shadow maps at coarser levels, but require many nodes for fine details. By conservatively adapting the depth map, we can significantly reduce the tree complexity. Our proposed method offers high compression rates, avoids quantization errors, exploits coherency along all data dimensions, and is well-suited for GPU architectures. Our approach can be applied for coherent shadow maps as well, enabling several applications, including high-quality soft shadows and dynamic lights moving on fixed-trajectories.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture, I.4.2 [Computer Graphics]: Compression (Coding)—Exact coding

1. Introduction

High-quality shadows are an important challenge in many real-time rendering applications in computer graphics. Shadow mapping [Wil78] is today's standard for real-time shadows, however, its quality is often limited by texture resolution. High-quality shadows in complex scenes can easily require resolutions up to two orders of magnitude larger than currently feasible for commodity GPUs. Adaptive approaches such as Adaptive Shadow Maps [FFBG01] or Cascaded Shadow Maps [Eng06,ZSXL06] are a common real-time solution, but come at the cost of reduced run-time performance. As virtual scenes often consist of large static parts (e.g., terrains or buildings), precomputing shadows has become a common practice.

Recent advances have shown that precomputed compressed high-resolution shadows maps can be a competitive alternative. Such techniques fully handle shadows cast by static objects on both static and dynamic receivers. Dynamic shadow casters are handled using standard shadow mapping techniques at run-time, with the added benefit of not having to render the static parts of the scene. Unfortunately, conventional image compression is not suitable for shadow maps, because lossless encoding (which is required to avoid light and shadow leaks) does not result in satisfactory compression rates and many techniques rely on run-length encoding, which prohibits random-access queries. Fast random-access compression of color textures has been investigated in the context of

GPU architectures ([DJ98, HIN04, IM06]), but these algorithms rely on quantizing data or lead to low compression rates (up to 5%), which is insufficient for higher resolutions. Consequently, custom schemes for shadow-map compression have recently been proposed. These approaches typically exploit the fact that, in static scenes, any depth value between the depth of the first and second surface underneath a pixel leads to a conservative occlusion test. However, previous approaches do not fully exploit the data coherency or rely on depth quantization.

We introduce a novel compression scheme for high-resolution shadow maps based on multiresolution hierarchies. We propose a sparsification process, which exploits the concept of dual shadow mapping (a shadow map for the front faces and one for the back faces) to create an extremely sparse, but conservative, multiresolution decomposition of the original (front) shadow map. This decomposition is efficiently encoded in a compressed regular tree for fast random access during run-time. We show that our approach achieves higher compression rates than all previous approaches, can be queried with real-time performance, and can be efficiently built using GPU architectures. Our approach is the first to exploit coherency along all data dimensions, does not rely on quantization (maintains full 32-bit precision) and supports shadow maps from arbitrary light sources. Another benefit is that it naturally incorporates all information required for hierarchical filtering operations since it offers a multiresolution representation and every level by itself is a complete shadow map. Finally, we show that our approach can be directly extended from single shadow maps (2D encoding using quadtrees) to a coherent set of shadow maps (3D encoding using octrees). Our approach is the first to enable efficient compression of and rendering with high-quality shadow map sets to produce soft shadows, and moving light sources with known trajectories (e.g., sun lighting).

2. Related Work

We will briefly discuss previous approaches for precomputed compressed shadows and compression of tree hierarchies. For a comprehensive overview of real-time shadow generation, we refer to the surveys of Eisemann et al. [ESAW11] and Woo et al. [WP12].

Compressing with line segments Based on the assumption that shadows are not cast inside of objects, Woo et al. [Woo92] proposed midpoint shadow mapping as a solution to self-shadowing artifacts. Midpoint shadow mapping computes a new shadow map, which represents the intermediate surface lying between the two surfaces closest to the light source. Since all depth values stay between the front facing geometry (the surfaces that represent the original shadow map) and the back facing geometry (the first exit point out of the object), the resulting occlusion test is conservative. An extension of this approach is dual shadow mapping [WE03], where the shadow maps are kept separate and shadow biasing can be performed adaptively. Based on this concept, Arvo et al. [AH05] introduced Compressed Shadow Maps (CSM) and showed how to compress a shadow map by representing each scan-line with a set of line segments approximating the midpoint surface. This approach shows that shadow map compression can be understood as signal compression with a specific spatially-variant bound. Although our approach can be interpreted as a 2D or 3D extension, finding the exact analytic equivalent in higher dimensions is a significantly more complex task.

Ritschel et al. [RGKM07] similarly compresses a set of coherent shadow maps by encoding the depth values of each pixel for all images by using a set of lines. However, both approaches do not fully exploit data coherency along all dimensions (e.g., only along the vertical dimension or "through" the image stack) and, therefore, cannot achieve optimal compression rates. Additionally, since these compression schemes are non-hierarchical they do not adapt well to the underlying data and efficient filtering along dimensions other than the compression dimension becomes impractical.

Precomputed Voxelized Shadows Recently, Sintorn et al. [SKOA14] proposed to precompute shadow information for a voxelized scene representation in projective light-space, which is efficiently encoded in a 2-bit Sparse Voxel Octree [LK10]. The octree is further compressed by subtree merging using a Directed Acyclic Graph (DAG) [KSA13]. The initial compression and construction performance was improved and resulted in the current state-of-the-art compression method for precomputed shadows [KSA15]. Unfortunately, the voxelization process leads to depth quantization and, although, the information is encoded hierarchically, it is not a multiresolution representation and fast filtering requires additional memory, almost doubling the size of the octree. Furthermore, the extension to shadow map sets is difficult since the compression is only efficient in the projected space of the light source.

Tree Compression A large body of research exists for efficient tree-based encoding of multiresolution hierarchies (e.g., [Woo84, Sam85, LH07]). Unfortunately, our tree must support random access and exhibits certain uncommon characteristics, making it difficult to apply most previous techniques. However, to address the overhead introduced by storing topology information, we utilize the pointer compression technique proposed by Lefebvre et al. [LH07] and efficiently encode tree pointers using 16-bits. We do not employ any vector quantization [GG91, CCG96, KE02] or other optimizations to the stored depth values themselves. Our results demonstrates that even without such data changes, our approach outperforms previous compression approaches, while maintaining full 32-bit depth precision.

3. Compressed Multiresolution Hierarchies

Multiresolution decompositions of images (e.g., wavelets [Mal89] or quadtree images [Sam84]) split features into components of different scales, typically storing homogeneous parts at coarser levels and details at finer levels. In consequence, finer levels (which contain more coefficients) are usually sparse, and coefficients are small if they are encoded as differentials to previous levels. Lossy compression exploits this characteristic and removes small coefficients assuming their influence to the composed image is low. However, for lossless compression all coefficients, independent of their magnitude, have to be considered. This results in decreased sparsity and diminished compression (Fig. 2, left).

Our key idea is to compress an alternative representation of the shadow map that is more homogeneous, but still conservative. Our

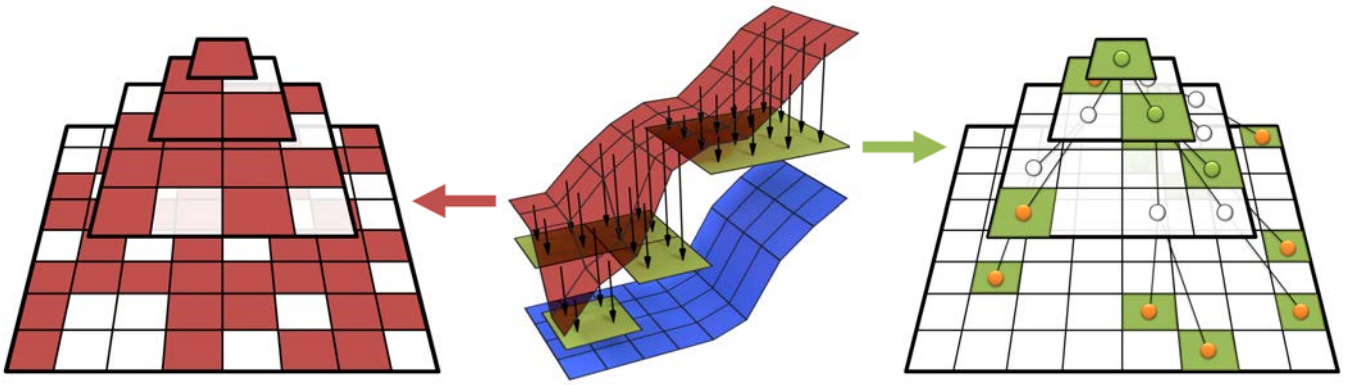


Figure 2: **Left:** A multiresolution decomposition of a shadow map requires many coefficients (red) at finer levels in varying regions and is typically not sparse. **Middle:** Using dual shadow mapping, an intermediate surface (green) can be found between the shadow map (red) and the auxillary second-surface shadow map (blue). Here, the intermediate surface represents a linear, conservative approximation of the shadow map by a set of axis-aligned planes. Choosing these planes to represent common depth values for many pixels results in a more homogeneous occlusion surface. **Right:** A significantly sparser multiresolution decomposition encoding the set of axis-aligned planes. The overlayed quadtree shows the encoding of coefficients. Inner nodes are represented by green circles, while leaf nodes are marked as yellow. Note the empty inner nodes (white circles) which are required to encode the topology information, but do not store any depth values.

goal is to find new depth representatives for each pixel in order to increase the sparsity of the hierarchy, but such that a conservative depth test remains possible. This is achieved by choosing values inside the boundaries defined by the first entry and exit surface points. To compute these bounds, we employ the concept of *dual shadow mapping* (Fig. 2, middle). As a whole, the procedure can alternatively be interpreted as the compression of an image with a spatially-varying error bound defined by an interval that must be met to maintain lossless compression. For fast random-access during run-time, we encode the sparse decomposition using a compressed quadtree (Fig. 2, right).

For non-watertight or one-sided objects, the upper and lower bounds of the depth interval need to be set to the depth value of the entry surface to ensure a conservative depth test. Although this reduces compression capability, our technique still handles these cases correctly and no artifacts are introduced.

In the following, we will first propose two greedy construction methods for finding sparse decompositions from conservative depth bounds. Then, we cover efficient encoding and traversal of the sparse representation using a compressed quadtree. Finally, we discuss shadow map filtering and propose an optimized traversal technique to significantly reduce filtering costs. While this section focuses on single shadow maps only, we will demonstrate in Sec. 4 how to extend our approach to a set of coherent shadow maps using a compressed octree.

3.1. Construction

Our first task is to define the allowable depth interval for each pixel. While the lower depth bound is defined by the original shadow map, the upper bound is determined using the second layer obtained via depth peeling [Eve01].

If the scene contains intersecting watertight objects, we can even further exploit the compression potential by ignoring surfaces inside of another objects. To exemplify this point, one can imagine each shadow map pixel corresponding to a ray cast from the light source. For each ray, one can track the encountered surfaces with a counter while advancing in the scene. The counter is incremented for each front-facing surface and decremented when a back-facing surface is encountered. The first intersection corresponds to the minimum bound of the allowable depth interval. After that, when the counter reaches zero, the ray has exited all objects and that point depth corresponds to the maximum of the depth interval. This procedure can be efficiently carried out for all pixels simultaneously via a depth-peeling algorithm. In the case of one-sided surfaces, they can be accounted for as coinciding front and back faces.

Having the per-pixel depth intervals of the shadow map, we then find a simplified surface inside the depth bounds which allows for a sparse decomposition. Interestingly, the task of finding an intermediate surface inside a given envelope is a common problem in mesh simplification, and for the 3D case it is known to be NP-hard [AS94]. We propose two greedy approaches, which perform a sparse decomposition and tree construction at the same time. The first one is a top-down approach which tries to globally minimize the number of distinct depth values, while the second one operates in a bottom-up manner and inspects only local pixels from the next finer level. The depth-value hierarchy will then be compressed using a quadtree structure.

Top-down construction The top-down construction starts at the coarsest level, which represents the entire image domain, and greedily selects the depth value which covers the largest number of depth intervals. It then marks all pixels that can be represented by this value as covered. These covered pixels will inherit the depth value from the coarsest level and only the remaining uncovered pixels need to store a separate depth value. The approach then proceeds to the next finer level by decomposing the domain into four quadrants. For each quadrant that contains at least one non-covered value, the algorithm is launched recursively. If all pixels in a quadrant are covered or the finest level is reached, the algorithm stops.

Algorithm 1 Pseudo-code for top-down hierarchy creation

```
function createHierarchy(intervals) :
    sortedIvals ← sort(intervals)
    createNode(rootNode, sortedIvals)
function createNode(node, sortedIvals) :
    bestIval ← 0
    numIvals ← 0
    bestNum ← 0
    for each ival ∈ sortedIvals do
      if isMin(ival) then
            numIvals++
      else
            numIvals- -
      end if
      if numIvals > bestNum then
            bestNum ← numIvals
            bestIval ← ival
      end if
    end for
    sortedIvals ← extractUncovered(sortedIvals, bestIval)
    for each child do
      childIvals ← extractChildIvals(child, sortedIvals)
      createNode(child, childIvals)
    end for
```

Consequently, homogeneous areas result in an early termination of the process.

To find the depth value covering most intervals, a direct approach would be to discretize the depth, create a histogram, and find the bin with the largest number of overlapping intervals. To avoid discretization, we propose an analytic sweep-based algorithm instead. We start by sorting all interval-bound depths (min and max) in ascending order. Then, we sweep through the sorted list and keep track of the number of overlapping intervals by incrementing a counter each time an interval minimum is encountered (we enter an interval) and by decrementing it when a maximum is encountered (we exit an interval). The highest detected count during the sweep leads to the depth representative which covers the maximum amount of intervals possible (Fig. 3). The pseudo-code for the top-down decomposition is shown in Alg. 1.

The algorithm requires a single sorting in the beginning, which can be performed in $O(n \log n)$ with n being the number of intervals (shadow map pixels). Once the initial list is sorted, all subsequent levels only require an $O(n)$ extraction step to retrieve the sorted list of uncovered-pixel intervals for the corresponding quadrant. Since the extraction has to be performed for each level, the overall runtime remains $O(n \log n)$.

Bottom-up construction An alternative is a bottom-up construction, which only considers the subjacent pixels of the next finer level during creation. This approach is better suited for parallel execution and, for all our test scenes, it performs competitively to the top-down construction, while being an order of magnitude faster.

The bottom-up construction is based on the idea of a min-max mipmap creation. Initially, the pixels at the finest level will contain

Algorithm 2 Pseudo-code for bottom-up hierarchy creation

```
childbounds ← [lower,upper]
for level from finestlevel - 1 to coarsestlevel do
  for each pixel in level do
    children ← getChildren(pixel, level+1)
    depthRepresentative ← findBestRepresentative(children)
    bounds(pixel) ← []
    for each child ∈ children do
      if satisfiesBounds(depthRepresentative, child) then
        setNonExistant(child)
        bounds(pixel) ← bounds(pixel) ∩ getBounds(child)
      end if
    end for
  end for
end for
```

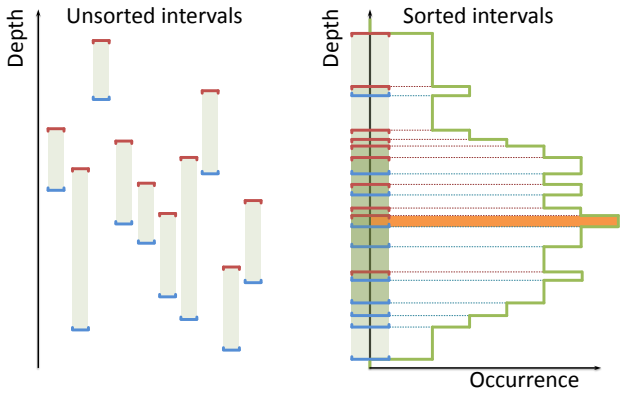


Figure 3: To find a depth value which intersects the maximum number of intervals from a given set (left), we propose a sweep-based mode finding scheme. The interval boundaries are sorted in ascending order first (left). We can then find the mode which corresponds to the best depth value by sweeping through the sorted list and keep track of the number of open intervals.

the lower and upper depth bound. When proceeding one level up, we analyze the four subjacent depth bounds of each pixel P using the same sweeping algorithm as for the top-down approach to find a largest depth interval valid for most of these four pixels. This interval I is then stored in P and all subjacent pixels, whose depth interval contain I are flagged as empty pixels. The algorithm then proceeds upwards to the next level. Once we reach the coarsest level, we will populate the map with actual depth values in every non-empty pixel by storing the average of the interval bounds.

Since only four intervals are treated at a time, the costly sorting step is avoided and the bottom-up construction requires only a constant number of operations per pixel. Hereby, although the complexity stays the same, the algorithm maps better to current GPU architectures, leading to a practical speedup. The pseudo-code is shown in Alg. 2 and Fig. 4 shows an example of the creation of a three level tree. The approach shows similarities to the Mallat-algorithm for wavelet construction [Mal89], but instead of using

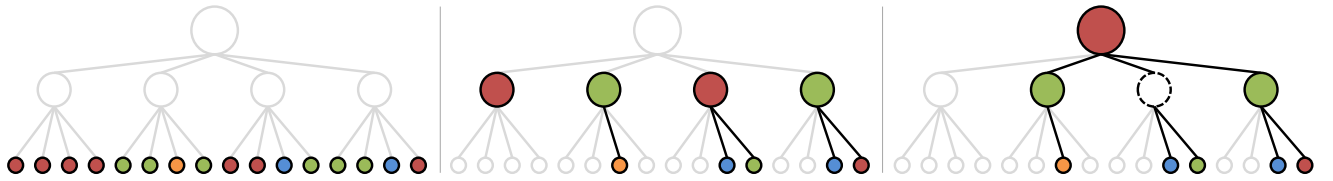


Figure 4: Three steps of the bottom up creation algorithm. **Left:** Initially a
value is pulled up the hierarchy. **Right:** The procedure is repeated for the r

the average as representative we choose the mode from the set of intervals in order to sparsify the representation.

Tiled construction For the extremely large shadow map resolutions encountered in our approach, it is infeasible to keep the full uncompressed data in memory to begin with. Fortunately, our approach is able to perform a tiled construction. First, the shadow map is divided in tiles of manageable size (in our implementation typically $4k \times 4k$ - $8k \times 8k$). For each tile, we compute its uncompressed bounds via depth peeling, compress it using the top-down or bottom-up algorithm, and store the depth bounds of the root node. After all tiles have been compressed, the stored depth bounds from all root nodes form the bounds of a new shadow map, which is again compressed to create the top level structure of the complete tree. Since only the uncompressed data of a single tile is required in memory at once, this construction procedure is both efficient and maintains a small memory footprint.

3.2. Compressed Quadtrees

The previous algorithms lead to a sparse hierarchy of depth values, which subsequently needs to be encoded efficiently while ensuring fast random-access at run-time — which are requirements fulfilled by a quadtree.

Encoding Our quadtree contains three node types: leaves (nodes with no children), inner nodes, and empty nodes. Inner nodes and leaves contain a 32-bit depth value. Empty nodes are only required to encode the quadtree connectivity, but do not contain a value themselves. Unlike other multiresolution decompositions, we do not encode the depth values using parent node differentials. In consequence, only a single value needs to be fetched during tree traversal, which reduces the memory throughput and accelerates lookups.

Inner and empty nodes, contain an 8-bit mask indicating the type of each child node (stored in two bits) and a pointer to the first child. We distinguish four cases for the child type: a) non-existent, b) leaf node, c) inner node, d) empty node. As it is common practice, a single pointer is sufficient, as all present child nodes are stored contiguously in memory, and the location of a specific child can be obtained from examining the mask in the parent node.

We employ the pointer encoding scheme proposed by Lefebvre et al. [LH07] in order to reduce the amount of memory needed to store pointers. Their scheme stores subtrees close together and allows us to encode pointer offsets, whose magnitudes decrease rapidly per level. Using a per-level scaling, we can efficiently encode pointers even for larger resolutions with just 16 bits introducing only a minimal padding overhead for alignment. We exhaustively search for the optimal per-level scaling factor as proposed by Lefebvre et al. Finally, we also pad full and empty nodes with a single byte, resulting in 4-byte aligned nodes (8 bytes for full nodes, and 4 bytes for empty nodes and leaves). The padding increases the memory footprint, but eases fetching the values on current GPU architectures, hence decreasing lookup times. Alternatively, 24-bit pointers could be used, however, we decided to keep the bit count compatible with the compressed octree representation which will be introduced in Sec. 4. In all our test scenes, no significant difference was introduced by using 16-bit pointers instead of 24-bit pointers for the quadtree compression. If memory footprint is overall more critical than traversal performance, the padding can be removed to further improve the compression.

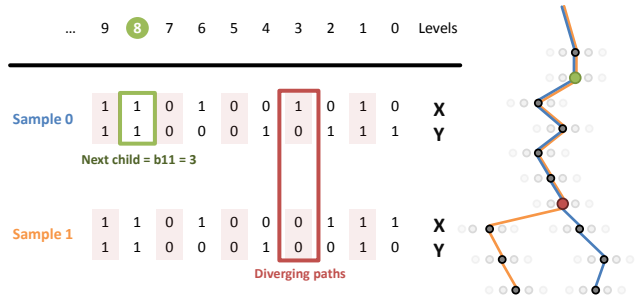


Figure 5: Finding the next child index can be done by inspecting the bits from the x and y position of the query point. The lowest common node of multiple query points can be found by finding the last level where the bits are equal.

Traversal Traversal of our compressed-quadtree encoding follows the same procedure as standard quadtree traversal, but performs lazy fetching of the depth values to account for the presence of empty nodes. The traversal path through the tree is defined by the position of the query point. By keeping track of the current level, we can directly compute the index of the next child using a few bit-operations (see Fig. 5). We start traversal at the root node (which is always a full node) and initialize a pointer which holds the index of the last node containing a depth value. After reading the children mask and pointer, we compute the index of the next child and query the mask to validate the child's existence. We compute the offset of the next child node from the mask and the 16-bit child pointer, to recursively continue the traversal. The pointer to the last position of a depth value is always updated, if we traverse a full node. If a child node does not exist or a leaf node is reached, the depth value is fetched from the last-stored position and the recursion is terminated. We do not query the depth value earlier, as these would

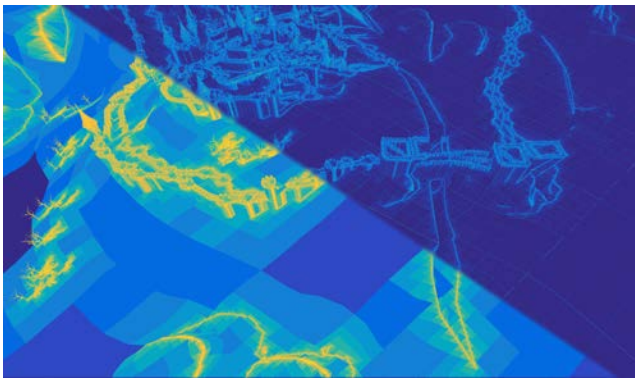


Figure 6: Required traversal steps in the CLOSED CITY scene for a 5x5 PCF filtering using a naive implementation (lower-left triangle of the image) and our optimization finding the lowest common node first (upper-right triangle). Bright colors represent numbers close to the maximum (tree height times number of samples), while dark ones mean that only few levels are traversed.

be unnecessary texture fetches, since we do not store differentials, but absolute values.

Although hierarchical traversal has typically a run-time depending on the tree height, the sparsity of our tree often leads to a termination after only a few levels.

3.3. Filtering

Efficiently filtering shadow lookups is an important aspect for shadow mapping. Percentage-closer filtering (PCF) [RSC87] is a popular technique which performs averaging of several depth-test results in a fixed-size kernel (usually an $r \times r$ box). A naive implementation of PCF using our approach would perform a full tree traversal for each kernel sample. Since our quadtree encodes a multiresolution prediction, we can perform analytic filtering when all samples end up at the same node. While this is not always the case, most samples share at least a common path from the root node until a certain level. This level can be directly computed from the minimum and maximum query points of the filter kernel and a few bit-operations (see Fig. 5). We propose to traverse all kernel samples together through the first few levels until we find the lowest common node where paths divergence. After that, each sample proceeds individually. This easy-to-implement optimization can lead to drastic improvements in PCF lookup time. A visualization of the amount of traversal steps for the naive implementation and our optimization for a 5x5 PCF filtering is shown in Fig. 6. Another possible optimization is to keep a cache of the last queried sample and reuse it if the next sample shares the same path through the tree down to the retrieved value.

Multiresolution anti-aliasing such as *hierarchical PCF* computes the shadow map footprint of a pixel and looks up the depth value for the corresponding resolution level. Since each of our tree levels encodes a full shadow map of the corresponding resolution, hierarchical filtering is natively supported. When performing PCF filtering, the appropriate sampling level of the hierarchy can be chosen to maintain a 1 to 1 correspondence between screen pixels and

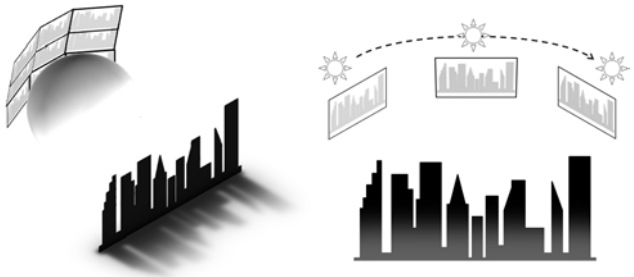


Figure 7: **Left:** Computing soft-shadows from an area light requires to sample multiple close-by points on the light source to apply slight variations to the incoming light direction. The set of shadow maps can be stacked in a 3D image cube for efficient compression. **Right:** Motion of dynamic light source, which is known in advance, can be precomputed by discretly sampling the trajectory, e.g., for simulating high-quality shadows from sun lights.

shadow map texels. This ensures that smooth shadows are present at any view distance regardless of a pixel's projected area in the shadow map. Furthermore, tri-linear filtering can be performed by choosing two consecutive sample levels and interpolating their values in order to create smooth transitions during motion.

4. Shadow map stacks

We can extend our concept of compressed multiresolution hierarchies directly to a set of coherent shadow maps. By stacking shadow maps in a 3D image cube, we can compute a sparse decomposition in the same manner as for a single shadow map and encode it using an octree. This approach is useful for rendering of soft shadows from area lights (Fig. 7, left) or varying light positions (Fig. 7, right). For soft shadows, we sample multiple light positions on an area light using a Hilbert-curve sampler as also used by Ritschel et al. [RGKM07]. The light positions can be jittered in order to avoid banding for smaller sampling rates. For moving light sources, the motion has to be known and the images are simply stacked in the order of discrete sample points along the trajectory. Our hierarchical structure is able to exploit coherency along all 3 dimensions by encoding homogeneous cubic regions in coarser levels of the hierarchy.

The construction and traversal techniques of the previous section can be directly applied for octrees as well. The only major difference, however, is that we now have to consider potentially eight children instead of four, which leads to 16-bit child masks. This conveniently removes the need for padding and makes the octree nodes perfectly aligned to 4-byte boundaries by default.

In the case of light trajectories, it is often only necessary to store a small number of different light positions. In this case, creating an equally-sized cube would restrict the resolution to match the number of images. Our approach allows for a convenient encoding of non-cubic image stacks by generating placeholder nodes with a depth boundary of [0,1]. Having the largest possible interval width, these nodes will be merged up the hierarchy during compression and introduce only little overhead.

	Method	$1K^2$	$2K^2$	$4K^2$	$16K^2$	$64K^2$	$256K^2$	$512K^2$	$1M^2$
Closed City	Uncompressed	4 MB	16 MB	64 MB	1 GB	16 GB	256 GB	1 TB	4 TB
	Arvo et al.	0.092 MB	0.23 MB	0.56 MB	2.83 MB	12.85 MB	54.09 MB	109.8 MB	221.9 MB
	Sintorn et al.	-	-	0.62 MB	3.40 MB	14.89 MB	60.46 MB	-	-
	Ours	**0.067 MB**	**0.17 MB**	**0.41 MB**	**2.10 MB**	**9.26 MB**	**38.43 MB**	**79.63 MB**	**160.5 MB**
	Ours (ratio)	**1.68%**	**1.06%**	**0.64%**	**0.21%**	**0.056%**	**0.015%**	**0.0078%**	**0.0039%**
Cityscape	Uncompressed	4 MB	16 MB	64 MB	1 GB	16 GB	256 GB	1 TB	4 TB
	Arvo et al.	0.15 MB	0.36 MB	0.78 MB	3.42 MB	14.03 MB	56.61 MB	113.5 MB	-
	Sintorn et al.	-	-	0.94 MB	3.94 MB	16.38 MB	63.34 MB	-	-
	Ours	**0.11 MB**	**0.26 MB**	**0.59 MB**	**2.70 MB**	**11.41 MB**	**46.73 MB**	**94.88 MB**	**190.4 MB**
	Ours (ratio)	**2.75%**	**1.625%**	**0.92%**	**0.26%**	**0.069%**	**0.0178%**	**0.0090%**	**0.0045%**
Villa	Uncompressed	4 MB	16 MB	64 MB	1 GB	16 GB	256 GB	1 TB	4 TB
	Arvo et al.	0.14 MB	0.40 MB	1.10 MB	5.89 MB	25.25 MB	103.84 MB	-	-
	Sintorn et al.	-	-	1.78 MB	9.26 MB	39.70 MB	166.47 MB	-	-
	Ours	**0.11 MB**	**0.35 MB**	**0.86 MB**	**5.01 MB**	**23.61 MB**	**101.52 MB**	**205.5 MB**	**414.6 MB**
	Ours (ratio)	**2.75%**	**2.18%**	**1.34%**	**0.48%**	**0.14%**	**0.03%**	**0.02%**	**0.009%**
Ship	Uncompressed	4 MB	16 MB	64 MB	1 GB	16 GB	256 GB	1 TB	4 TB
	Arvo et al.	0.21 MB	0.60 MB	1.44 MB	6.73 MB	28.23 MB	115.34 MB	233.2 MB	-
	Sintorn et al.	-	-	2.01 MB	8.66 MB	35.57 MB	153.67 MB	-	-
	Ours	**0.15 MB**	**0.43 MB**	**1.05 MB**	**5.26 MB**	**22.71 MB**	**94.18 MB**	**191.5 MB**	**392.1 MB**
	Ours (ratio)	**3.75%**	**2.68%**	**1.64%**	**0.51%**	**0.13%**	**0.036%**	**0.018%**	**0.0093%**

Table 1: Compression results for our 2D MH approach comparing to the scanline compression of [AH05] and the voxelized shadows approach of [SKOA14]. Our approach outperforms competing compression approaches consistently while retaining full depth precision.

	Resolution	Total nodes	Rendering	MH creation	Quadtree creation	Serialization	Total time	Kampe et al. time
Closed City	$1K^2$	14944	0.019	0.001	0.003	0.007	0.089	-
	$4K^2$	90775	0.067	0.003	0.004	0.017	0.242	0.098
	$64K^2$	2037131	3.253	0.555	0.584	0.260	5.301	4.878
	$256K^2$	8477953	39.53	9.052	3.653	1.001	55.18	65.23
Cityscape	$1K^2$	25859	0.012	0.001	0.003	0.009	0.089	-
	$4K^2$	130974	0.044	0.004	0.009	0.021	0.221	0.061
	$64K^2$	2508119	1.755	0.551	0.612	0.299	3.888	4.089
	$256K^2$	10215660	22.45	8.975	4.287	0.984	38.85	51.58

Table 2: A detailed inspection of construction timings and tree characteristics for the multiresolution quadtree compression for the Closed City and Cityscape scene. Rendering times dominate construction times, while creation of the sparse decomposition and quadtree encoding constitutes around one third of the total. We also provide a comparison to the method from Kampe et al. [KSA15].

5. Results

In this section, we demonstrate the compression capabilities of our method for five test scenes and evaluate its construction and runtime. The Closed City scene (613K triangles) represents a typical open-world game setting with both large scale and detailed features. The Cityscape scene (11K triangles) is an example of an architectural design model, while the Villa scene (89K triangles) as well as the Ship scene (810K triangles) are examples of scenes containing many fine scale details. The Dragon scene (7.2M triangles) consists of a scanned model with a very high polygon count.

We implemented larger parts of the construction algorithm on the GPU using NVidia CUDA 7.5. The rendering is done using OpenGL 4.3 and deferred shading, and our measurements are reported for the evaluation of the shadows. All experiments were at a resolution of 1920x1080 on Windows 7 using a PC with and Intel i7-5820K CPU with 16GB of system memory, and an NVidia Titan X GPU. We re-implemented the algorithm of Arvo et al. [AH05] for the comparison to scanline compression. For the comparison to DAG-based compression of voxelized shadows [SKOA14], we used the implementation provided by the authors which includes all the improvements from Kampe et al. [KSA15].

Quadtree compression Table 1 presents compression results for single shadow maps using multiresolution quadtree compression. We report memory footprints for the quadtree using the 1-byte padding for inner nodes and a full 32-bit depth precision. In all cases, our algorithm outperforms the previous approaches and is able to compress even large resolutions in the order of hundreds of thousands down to a few hundred megabytes. All results used the bottom-up construction. Note that, in contrast to the previous approaches, our method implicitly encodes a full multiresolution representation of the shadow information.

Table 2 showcases detailed construction times and total node quantity for several test scenes, as well as the construction time

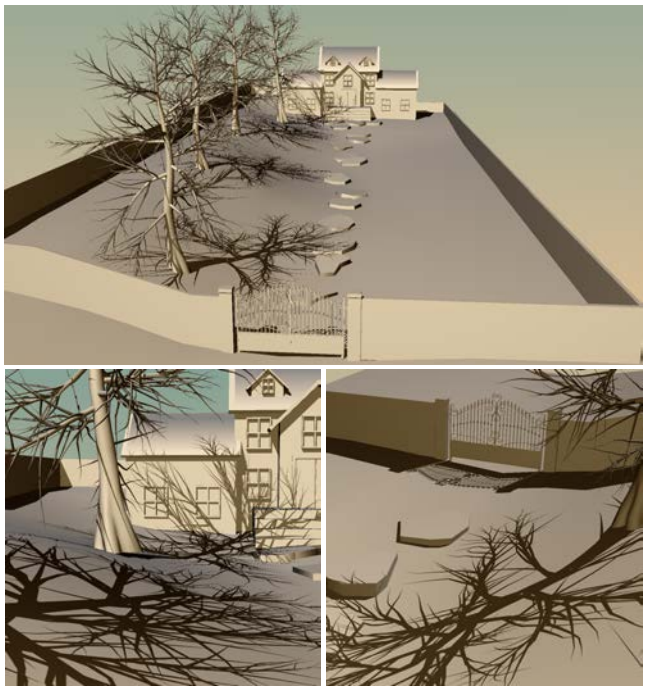


Figure 8: **Top**: An overview of the VILLA scene with an unfiltered $32K^2$ compressed shadow map. **Bottom left:** Close-up of filtered shadows rendered in 2 ms using a 3x3 non-hierarchical PCF kernel. **Bottom right**: Another viewpoint with a 5x5 non-hierarchical PCF filtering kernel rendered in 5 ms.

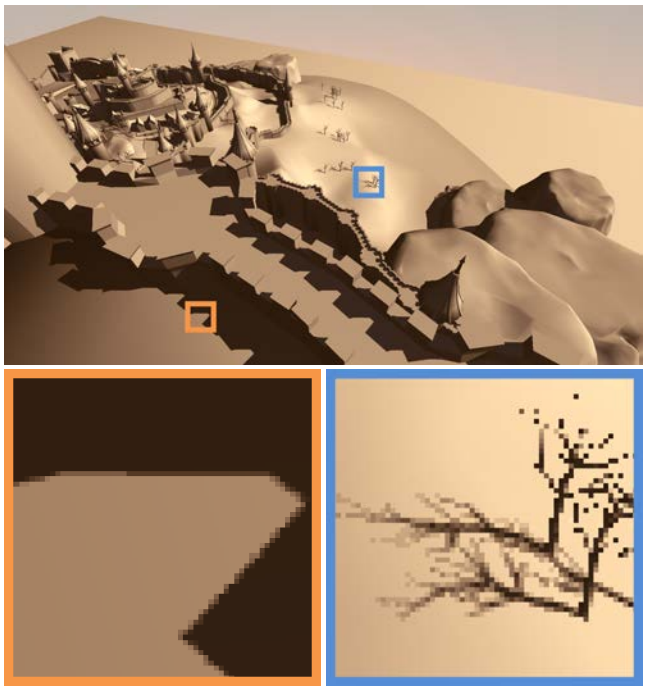


Figure 9: **Top**: An example of hierarchical PCF with a 3×3 kernel in the CLOSED CITY scene. Anti-aliased shadows are present at all distances. **Bottom left**: An inset showing anti-aliased shadows closer to the camera. **Bottom right**: Another inset showing anti-aliased shadows cast by a complex occluder in the distance.

	Method	$4K^2$	$16K^2$	$64K^2$	$256K^2$
Single	Shadow mapping	0.25	0.36	-	-
	Ours	0.495	0.52	0.54	0.71
	Arvo et al.	0.39	0.51	1.04	2.6
	Sintorn et al.	0.61	0.61	0.68	0.72
3×3	Shadow mapping	0.34	0.65	-	-
	Our PCF Naive	3.35	3.7	3.99	4.11
	Our PCF Optimized	1.61	1.72	1.89	2.04
	Arvo et al.	0.85	1.25	4.18	9.7
5×5	Shadow mapping	0.62	1.46	-	-
	Our PCF Naive	7.7	8.49	8.9	9.32
	Our PCF Optimized	3.45	3.95	4.4	4.72
	Arvo et al.	1.4	2.25	5.6	15.9
9×9×9	Sintorn et al.	0.78	0.84	0.93	0.96

Table 3: Traversal time in ms for a single scene (VILLA) for our approach and comparing to standard shadow mapping and the approaches from Arvo et al [AH05] and Kampe et al. [KSA15]. The latter is highly optimized for a cubic 9×9×9 kernel size and for a fair comparison, we only report these numbers.

for the voxelized shadows approach from Kampe et al. [KSA15]. While the preprocessing time is not interactive for larger resolutions, we report numbers in the same order of magnitude as the highly-optimized implementation from Kampe et al. It can be seen that most of the time is spent in the depth peeling in order to obtain the initial depth bounds. The compression itself is mostly dominated by the bottom-up construction that creates the sparse decomposition, and to a lesser extent by the encoding of the quadtree. Finding the optimal per-level scale for 16-bit pointer compression only takes up a small fraction of the overall construction time.

In Table 3 we report timings for single lookup performance and different PCF kernel sizes. We compare our optimized PCF implementation against a naive one, standard shadow mapping (for supported resolutions), and the methods from Arvo et al. [AH05] and Kampe et al. [KSA15] for the VILLA scene. Since it is naturally provided, our implementations perform *hierarchical* PCF. Shared traversal is significantly faster for PCF filtering than a naive implementation (up to 2 ms for a 3x3 kernel, and 4.7 ms for 5x5). The method from Arvo et al. performs well for small PCF kernels at low resolutions, but does not scale well. The voxelized shadow approach is highly optimized for 9x9x9 and 17x17x17 cubic kernels, and achieves almost the same look-up times compared to single lookups. Nevertheless, their filtering is done in 3D space and at reduced precision, which potentially results in artifacts.

In Fig. 8 and Fig. 9, we present visual results for unfiltered, non-hierarchical, and hierarchical PCF filtering using our method. It can be seen that even high-frequency shadows from small features can be faithfully rendered. Additionally, the insets in Fig. 9 show that anti-aliased shadows at any view distance can be achieved by sampling the appropriate level.

As a practical optimization, Sintorn et al. [SKOA14] store the top 6 levels of the hierarchy in a simple dense grid for large resolu-

	Method	512^3	$1K^3$	$2K^3$	$4K^3$
DRAGON	Uncompressed	512 MB	4 GB	32 GB	256 GB
	2D MHSM	11.5 MB	48.4 MB	205.8 MB	863.6 MB
	3D MHSM	6.7 MB	27.8 MB	145 MB	689.2 MB
	3D/2D ratio	57.9%	57.3%	70.4%	78.8%

Table 4: Memory footprint of 3D multiresolution octree-based compression for a set of **N** images with different resolutions. As a comparison we report the memory by naively using our 2D quadtree compression for each image individually.

	Method	**256** $\times$ $2K^2$	**256** $\times$ $4K^2$	**256** $\times$ $8K^2$
CITYSCAPE	Uncompressed	4 GB	16 GB	64 GB
	Compressed size	45.85 MB	136.08 MB	456.32 MB
	Construction time	12.3 s	36.2 s	135.9 s

Table 5: Memory footprint and construction times of 3D multiresolution octree-based compression for a non-cubic data set of 256 images taken a fixed-trajectory moving light source.

tions. This requires a constant 8 MB of memory, which is negligible at higher resolutions. The numbers we report for their approach include this optimization. In our case, this would allow us to remove the upper 11 levels of the quadtree and halve the number of traversed levels on average. If evaluation time is more critical than compression, this could potentially lead to faster lookups.

Octree compression We evaluate our 3D compression for high-quality soft shadows and light motion, and compare it against naively compressing each image separately with our 2D scheme. Table 4 reports memory sizes for our image stack compression algorithm for soft-shadows. In the table, we show the resulting memory footprint of compressing a set of shadow maps from an area light separately using our quadtree structure and compressing them with our octree approach. It can be seen that our 3D compression provides an additional gain and is able to reduce the compression rate down to 57.9% at best compared to 2D compression.

A visual impression of high-quality soft-shadows in the SHIP scene is given in Fig. 11. Please note that the shadow penumbrae generated in this way is geometrically correct and appears more realistic as opposed to PCF filtering. The lookup time for 32 random samples per pixel out of 512 depth maps is 14 ms, whereas evaluating 64 samples takes 30 ms.

Finally, we evaluate our 3D compression scheme for non-cubic image stacks for fixed-trajectory light sources in Table 5. We show different viewpoints for the CITYSCAPE scene in Fig. 10. Since the construction of the octree is based on cubic tiles, which need to be kept small to fit in GPU memory, the viewport size for rendering is restricted, leading to a large amount of render calls. Therefore, rendering makes up most of the octree creation time.

6. Conclusion and Future Work

We presented a novel compression scheme for shadow maps based on multiresolution hierarchies. We demonstrated that our approach creates high-quality shadows for real-time rendering and achieves high compression rates. For example, our method is able to compress a 32-bit shadow map with a resolution of *1.000k x 1.000k* (uncompressed 4 terabytes) down to 0.0045% at best. Another benefit of our approach is that a multiresolution representation is highly beneficial for fast hierarchical filtering. Using a set of coherent shadow maps, we are able to create soft shadows or dynamic lights on a fixed trajectory.

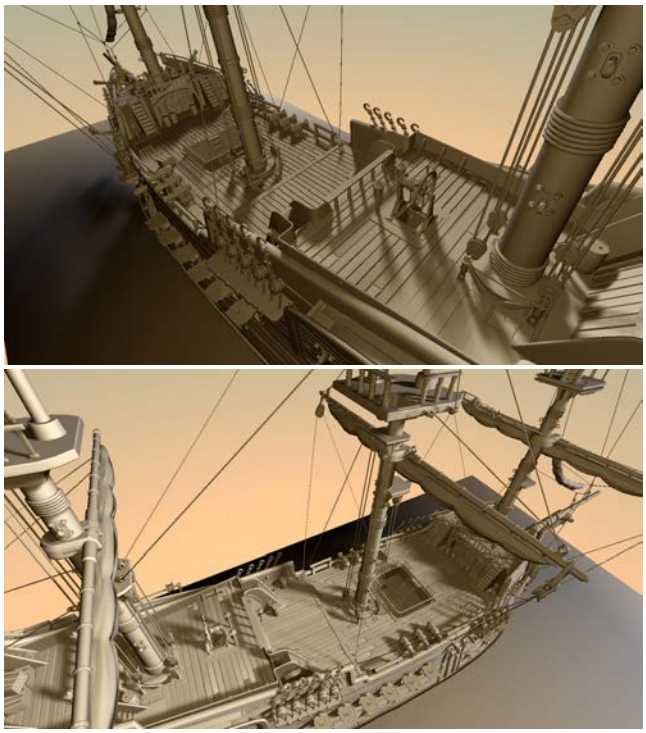


Figure 11: Realistic soft shadows in the SHIP scene generated with an octree from 512 depth maps of $1K^2$ resolution. Overall, the compressed octree is stored in 57 MB. **Top:** A closeup using 32 shadow-map samples per pixel rendered in 14 ms. **Bottom:** Another viewpoint using 64 samples rendered in 30 ms.

While our approach can handle non-closed geometry, these parts as well as very thin objects lead to a reduction of compression performance. This stems from the reduced size of the depth interval, diminishing the possibility for creating homogeneous regions. Nonetheless, this problem is shared by all related compression methods. Another issue is geometry that is viewed at grazing angles due to the representation of intermediate surfaces as strictly axis-aligned planes. In the future, we would like to investigate alternative, non-linear representations to overcome these limitations.

Additionally, we would like to investigate non-regular subdivision schemes (e.g., based on multiresolution kd-trees) to provide more adaptivity to the underlying depth signal. Still, it is not clear how to efficiently construct these non-regular trees and if the overhead of storing subdivision information introduces a too large overhead. Finally, we want to investigate sparse decompositions that avoid storing topological information for empty inner nodes, e.g., matrix trees [AT10].

Figure 10: The CITYSCAPE scene shows shadows from different views taken from a compressed shadow map stack of 256 $4K^2$ images taken on a trajectory above the city. The octree has a compressed size of 136.08 MB (uncompressed 16 gigabytes) and is queried in under 1 ms.

7. Acknowledgements

We would like to thank Erik Sintorn and his team for kindly providing their voxelized shadows implementation for our tests, as well as the CLOSED CITY and VILLA test scenes. The DRAGON model is freely available at the Stanford 3D scanning repository[†]. The SHIP scene was modeled by Greg Zaal and Chris Kuhn, and is available under the creative commons license at Blend Swap[‡].

Ths work was partially funded by the EU FP7-323567 project Harvest4D and the Intel VCI at Saarland University.

References

[AH05] ARVO J., HIRVIKORPI M.: Compressed shadow maps. *Vis. Comput. 21*, 3 (Apr. 2005), 125–138. 2, 7, 8

[AS94] AGARWAL P. K., SURI S.: Surface approximation and geometric partitions. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms* (1994), SODA '94, Society for Industrial and Applied Mathematics, pp. 24–33. 3

[AT10] ANDRYSCO N., TRICOCHE X.: Matrix trees. *Computer Graphics Forum 29*, 3 (2010), 963–972. 9

[CCG96] CHADDHA N., CHOU P. A., GRAY R. M.: Constrained and recursive hierarchical table-lookup vector quantization. In *Data Compression Conference* (1996), IEEE Computer Society, pp. 220–229. 2

[DJ98] D. M., J. B.: Directx 6 texture map compression. *Game Developer* (1998), 42–46. 2

[Eng06] ENGEL W.: Cascaded shadow maps. *ShaderX5: Advanced Rendering Techniques* (2006). 1

[ESAW11] EISEMANN E., SCHWARZ M., ASSARSSON U., WIMMER M.: *Real-Time Shadows*. A.K. Peters, 2011. 2

[Eve01] EVERITT C.: Interactive order-independent transparency, 2001. 3

[FFBG01] FERNANDO R., FERNANDEZ S., BALA K., GREENBERG D. P.: Adaptive shadow maps. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (2001), SIGGRAPH '01, ACM, pp. 387–390. 1

[GG91] GERSHO A., GRAY R. M.: *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1991. 2

[HIN04] HONG Z., IOURCHA K., NAYAK K.: Fixed-rate block-based image compression with inferred pixel values, Aug. 10 2004. US Patent 6,775,417. 2

[IM06] INADA T., MCCOOL M. D.: Compressed Lossless Texture Representation and Caching. In *Graphics Hardware* (2006), The Eurographics Association. 2

[KE02] KRAUS M., ERTL T.: Adaptive texture maps. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (2002), HWWS '02, Eurographics Association, pp. 7–15. 2

[KSA13] KÄMPE V., SINTORN E., ASSARSSON U.: High resolution sparse voxel dags. *ACM Transactions on Graphics 32*, 4 (2013). SIGGRAPH 2013. 2

[KSA15] KÄMPE V., SINTORN E., ASSARSSON U.: Fast, memory-efficient construction of voxelized shadows. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2015), ACM. 2, 7, 8

[LH07] LEFEBVRE S., HOPPE H.: Compressed random-access trees for spatially coherent data. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (2007), EGSR'07, Eurographics Association, pp. 339–349. 2, 5

[LK10] LAINE S., KARRAS T.: *Efficient Sparse Voxel Octrees – Analysis, Extensions, and Implementation*. NVIDIA Technical Report NVR-2010-001, NVIDIA Corporation, Feb. 2010. 2

[Mal89] MALLAT S. G.: A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 11* (1989), 674–693. 2, 4

[RGKM07] RITSCHEL T., GROSCH T., KAUTZ J., MÜELLER S.: Interactive illumination with coherent shadow maps. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (2007), EGSR'07, Eurographics Association, pp. 61–72. 2, 6

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. *ACM Siggraph Computer Graphics 21*, 4 (1987), 283–291. 6

[Sam84] SAMET H.: The quadtree and related hierarchical data structures. *ACM Comput. Surv. 16*, 2 (June 1984), 187–260. 2

[Sam85] SAMET H.: Data structures for quadtree approximation and compression. *Commun. ACM 28*, 9 (Sept. 1985), 973–993. 2

[SKOA14] SINTORN E., KÄMPE V., OLSSON O., ASSARSSON U.: Compact precomputed voxelized shadows. *ACM Trans. Graph. 33*, 4 (July 2014), 150:1–150:8. 2, 7, 8

[WE03] WEISKOPF D., ERTL T.: Shadow mapping based on dual depth layers. *Eurographics 2003 Short Papers* (2003). 2

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph. 12*, 3 (Aug. 1978), 270–274. 1

[Woo84] WOODWARK J. R.: Compressed quad trees. *The Computer Journal 27*, 3 (Aug. 1984), 225–229. 2

[Woo92] WOO A.: The shadow depth map revisited. In *Graphics Gems III*, Kirk D., (Ed.). Academic Press, 1992, pp. 338–342. 2

[WP12] WOO A., POULIN P.: *Shadow Algorithms Data Miner*. A K Peter/CRC Press, June 2012. 2

[ZSXL06] ZHANG F., SUN H., XU L., LUN L. K.: Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications* (2006), pp. 311–318. 1

[†] http://www.graphics.stanford.edu/data/3Dscanrep

[‡] http://www.blendswap.com

Animated Mesh Approximation With Sphere-Meshes

JEAN-MARC THIERY
Delft University of Technology
ÉMILIE GUY and TAMY BOUBEKEUR
LTCI, CNRS, Télécom-ParisTech, Université Paris-Saclay
ELMAR EISEMANN
Delft University of Technology

Performance capture systems are used to acquire high-quality animated 3D surfaces, usually in form of a dense 3D triangle mesh. Extracting a more compact, yet faithful representation is often desirable, but existing solutions for animated sequences are surface-based, which leads to a limited approximation power in the case of extreme simplification. We introduce animated sphere-meshes, which are meshes indexing a set of animated spheres. Our solution is the first to output an animated volumetric structure to approximate animated 3D surfaces and optimizes for the sphere approximation, connectivity, and temporal coherence. As a result, our algorithm produces a multi-resolution structure from which a level of simplification can be selected in real-time, preserving a faithful approximation of the input, even at the coarsest levels. We demonstrate the use of animated sphere-meshes for low-cost approximate collision detection. Additionally, we propose a skinning decomposition, which automatically rigs the input mesh to the chosen level of detail. The resulting set of weights are smooth, compress the animation, and enable easy edits.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Object hierarchies*

General Terms: Algorithms

Additional Key Words and Phrases: animated shape approximation, simplification, abstraction

ACM Reference Format:

Thiery, J.-M., Guy, E., Boubekeur, T. and Eisemann, E. 2016. Animated Mesh Approximation With Sphere-Meshes. ACM Trans. Graph. VV, N, Article XXX (Month 2015), 13 pages.
DOI = 10.1145/XXXXXXX.YYYYYYY
http://doi.acm.org/10.1145/XXXXXXX.YYYYYYY

This work has been partially funded by the European Commission under contract FP7-323567 Harvest4D and the Intel VCI at Saarland University.

DOI 10.1145/XXXXXXX.YYYYYYY
http://doi.acm.org/10.1145/XXXXXXX.YYYYYYY

1. INTRODUCTION

Modern performance capture systems automatically generate high-resolution 3D animated meshes from real-world objects and characters [Vlasic et al. 2008; de Aguiar et al. 2008]. However, the resulting output sequences are mostly targeted for "replay"; each frame is represented independently.

In this context, high-level control mechanisms, based on geometric approximation, are often desirable to perform shape and motion processing, analysis and editing. The underlying structures should capture the global spatial embedding of the animated shape and remain coarse enough to act as intuitive intermediate representations to edit the sequence. A number of such approaches have been proposed to reconstruct control structures having a shape-driven 3D spatial embedding such as an *animation skeleton* or a *deformations cage*. The former offers a natural layout to model articulated objects while the latter is more suited for volume evolution modeling and maps better to non-tubular geometries. Ideally, one would like to benefit from both worlds, using a medial structure, such as the *medial axis transform* [Blum 1967] (MAT), which provides both explicitly an inner skeleton and local thickness models. At a coarse scale, such a solution would be a valid alternative to both cages and skeletons. At finer scales, the volumetric nature of the medial axis is problematic, requiring an untractable amount of values, which describe geometry far away from the location of interest (i. e., the surface). In the case of static meshes, *sphere-meshes* [Thiery et al. 2013] tackle this problem. A sphere-mesh is a multi-resolution mesh structure (edges, faces) indexing a set of spheres, which are optimized to properly approximate the input geometry locally, when linearly interpolated on the sphere-mesh simplices. Such a volume approximation compactly represents the shape as a convolution surface or a (simpler) primitive sum, proving useful for a variety of applications such as surface analysis [Siddiqi and Pizer 2008], shadowing [Wang et al. 2006] or proximity queries [Stolpner et al. 2012].

Unfortunately, volumetric structures and abstractions, such as the MAT or 3D “blobs” [Muraki 1991], exist mostly for static shapes. For example, the MAT cannot be used to represent consistently animated 3D data, since the MAT of a shape varies strongly and in unexpected ways along the animation, even for smooth and isometric transformations of the shape.

In this paper, we approximate animated mesh sequences with *animated* sphere-meshes. Our algorithm outputs a nested hierarchy of simplified animated sphere-meshes that evolve from surface structures at fine scales to more volumetric structures at coarse scales, keeping a consistent connectivity during the entire animated sequence. Our work is the first to provide a time-consistent volumetric approximation of an animated surface mesh at different levels of detail. Existing applications using this representation can

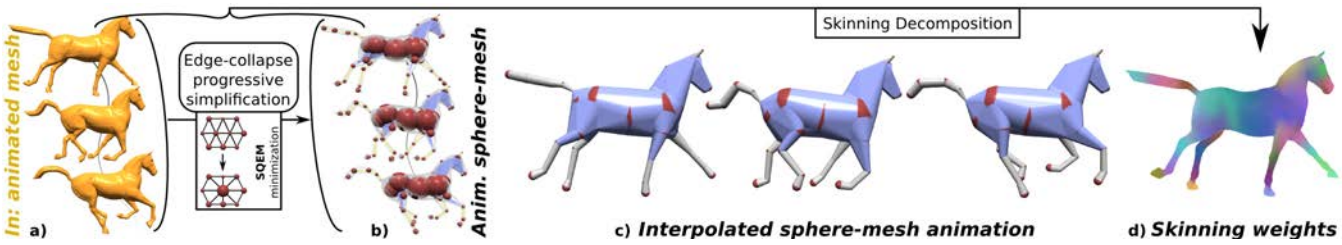


Fig. 1. Our algorithm takes as input an animated surface mesh (a), and outputs an animated sphere-mesh: a non-manifold mesh indexing a set of animated spheres (b): spheres in red, edges in yellow, triangles in blue). The input animation can be approximated efficiently by interpolating the spheres on the simplices of the sphere-mesh (c): interpolated edges in grey, interpolated triangles in blue). We perform a skinning decomposition of the animation (d) using a sphere-mesh to define a level of detail. The resulting weight maps allow us to efficiently compress the data via linear blend skinning and to define new poses.

thus trivially be used directly on our data, when they could previously use static or manually-designed animated convolution surfaces only. We illustrate this point with an application to approximate collision detection. As a second application, we show how to perform automatic rigging of the input animation using our animated sphere-mesh as a rigging structure for the original data. The resulting skinning weights are smooth enough for high-end shape modeling applications and reproduce faithfully the complex deformations learned from the input animation. Additionally, we introduce mixed weights to benefit from geometrically salient structures in regions which lack detailed motions along the animation.

Specifically, we make the following contributions (see Fig. 1):

—an approximation algorithm to build an animated sphere-mesh from the input (Sec. 3);

—a spherical quadric error metric for animated meshes (Sec. 3);

—an error-minimization algorithm (Sec. 4.1.1), which is compatible with temporal-coherence optimization (Sec. 4.2);

—an optional connectivity improvement (Sec. 4.3);

—a skinning decomposition method (Sec. 6.2), based on the chosen simplification level of the animated sphere-mesh, resulting in smooth weights considering either animation or geometric weights, depending on their appropriateness.

We present the results for our approximation (Sec. 5) and skinnning decomposition (Sec. 6.2), before discussing our work (Sec. 7).

2. RELATED WORK

Animated-mesh simplification. Contrary to static-mesh simplification [Talton 2004], few approaches address animated-mesh simplification. Furthermore, the output of existing methods is generally an animated triangle mesh [Mohr and Gleicher 2003; DeCoro and Rusinkiewicz 2005; Kircher and Garland 2005; Landreneau and Schaefer 2009; Zhang et al. 2010], or a set of triangle meshes with minimal frame-to-frame connectivity transformations [Houle and Poulin 2001; Huang et al. 2005; Payan et al. 2007]. Most of these techniques rely on a quadric error metric (QEM) [Garland and Heckbert 1997] to define the cost of collapsing an edge to a 3D vertex by summing the QEMs over all frames. Such a solution exhibits two major advantages. First, edge decimation is fast and efficient, which is crucial for processing large animations; Second, starting from consistent meshes and decimating corresponding edges in all frames simultaneously, maintains the output connectivity along the animation, which simplifies storage and editing. We build upon such a solution but, in contrast to previous approaches, we opt for an animated volumetric approximation, which has been shown to be favorable for coarse simplifications [Thiery et al. 2013].

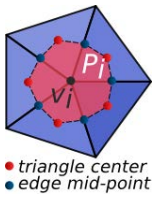
Skinning from animations. For purely geometric animated-mesh compression, most work focuses on algebraic approximations (generally assimilated by non-linear dimensionality reduction), such as skinning decomposition techniques. The latter approximate the input animation via a single mesh (often, a frame of the animation) and additional data in form of skinning weights and a skeleton animation [de Aguiar et al. 2008; Le and Deng 2014], or simply a set of *proxy-bone* transformations [Kavan et al. 2010; James and Twigg 2005; Le and Deng 2012], which can be either rigid or arbitrary. The input is then approximated by linear blend skinning (LBS) [Magnenat-Thalmann et al. 1988; Lewis et al. 2000]. These methods tend to produce a single bone per rigid region, considering the motion of the input mesh rather than its complete animated geometry (e. g., if the head of an animal is globally rigid along the animation, a single bone will be found). Additionally, the skeleton connectivity is usually deduced from an adjacency graph of the rigid parts in the mesh. Our approach makes it possible to select a skeletal domain taking the full animated geometry into account and to choose an appropriate level of detail before deriving a particular skinning decomposition.

Sphere-meshes. In [Thiery et al. 2013], the QEM decimation algorithm [Garland and Heckbert 1997] is modified to approximate the sum of squared distances of the tangent planes in a region to a sphere instead of a point. Iteratively, the edge inducing the lowest error according to this metric is collapsed. The resulting nested hierarchy of sphere-based approximations can then be traversed in real-time, making it possible to progressively navigate from surface structures to volumetric structures by decreasing the number of spheres. Nonetheless, this approach did not consider animation, nor does there exist any technique for building an animated simplified volumetric representation. One important reason could be that most existing techniques for the static case rely on a MAT, which is inconsistent during animation when determined per frame.

Technical background

We briefly present the edge-collapse decimation framework described in [Thiery et al. 2013], which we extend in our system. It takes as input a surface mesh (preferably but not necessarily manifold nor closed, and possibly containing some wire edges), and outputs a nested hierarchy of coarser meshes with a sphere associated with each of the vertices. The input mesh is then approximated at multiple resolutions by linearly interpolating the spheres on each of the triangles and edges of a sphere-mesh in the hierarchy.

Computation from triangle meshes. At first, each vertex v_i is associated with a region of the mesh called its barycentric cell P_i (see inset figure). In each simplification step, an edge is collapsed and the region associated with the newly-created vertex is set as the union of the regions of the two collapsed vertices. The cost associated with each edge collapse is defined as the integral of the squared distance from the tangent planes in the region to a 3D sphere, whose radius and position are optimized to minimize this energy. All edges are collapsed iteratively, and these collapse operations are ordered by increasing cost in a priority queue.



When optimizing the geometry of a sphere approximating a region, its diameter is constrained to be smaller than the *width* [Gärtner and Herrmann 2001] of the region, essentially to avoid impossibly large spheres approximating planar surfaces.

The cost of a partition into K regions $\{I_k, \mathbf{s}_k\}_{k\leq K}$ is

$$\mathfrak{C}(\{I_k, \mathbf{s}_k\}_{k\leq K}) = \sum_{k\leq K} \mathbf{Q}_{I_k}(\mathbf{s}_k)$$

where $\{I_k\}_{k\leq K}$ is a partition of the vertex' barycentric cells $\{P_i\}$, $\mathbf{Q}_{I_k}$ is the sum of the spherical quadric error metrics (SQEMs) of the vertices in the set I_k and $\mathbf{s}_k$ is a sphere associated with the region k. Formally, $\mathbf{Q}_{I_k}(q,r) = \sum_{i\in I_k} \int_{\xi\in P_i} SQEM_{(p_\xi,n_\xi)}(q,r)d\sigma_\xi$, where

$$SQEM_{(p_\xi,n_\xi)}(q,r) := (n_\xi{}^{\mathsf{T}} \cdot (p_\xi - q) - r)^2 \qquad (1)$$

is the squared distance between the plane intersecting $p_\xi \in \mathbb{R}^3$ with normal orientation $n_\xi \in \mathbb{R}^3$ and a sphere $\mathbf{s} := (q,r) \in \mathbb{R}^3 \times \mathbb{R}^+$, consisting of its center q and radius r, and $d\sigma_\xi$ denotes the infinitesimal surface element. The sphere s_k is optimized to best approximate the region I_k (in the sense that it minimizes its associated quadric $\mathbf{Q}_{I_k}$). As explained in detail in [Thiery et al. 2013], the SQEM is sensitive to the normal orientation, and minimizing it while constraining the radius of the solution to be positive prevents approximating concave patches with a single sphere.

Hierarchy traversal. At each decimation step the locations and radii of the collapsed and created spheres are recorded, as well as the edges/triangles that are deleted or created. All these events are stored in an array, representing a sphere-mesh hierarchy. After the decimation process, the user can navigate in real-time in the sphere-mesh hierarchy to find the desired level of detail. Following Hoppe et al. [Hoppe 1996], we note $\mathcal{M}_0$ the input mesh, and $\mathcal{M}_\tau$ the mesh resulting from the τ^{th} recorded edge-collapse operation (i. e., $\mathcal{M}_\tau$ is obtained by collapsing an edge of $\mathcal{M}_{\tau-1}$).

Surface extraction. A surface approximating the input mesh can be extracted from a sphere-mesh as a convolution surface [Bloomenthal and Shoemake 1991] between the base mesh and the spheres. In our work, spheres are interpolated along the edges and faces of the base mesh. An interpolated edge corresponds therefore to a cone cut by orthogonal planes at each edge extremity, and an interpolated triangle corresponds to a triangular prism with 3 faces from the edges extrusion and two triangular faces representing the lower and upper crust. For the sake of simplicity, although convolution-surface extraction algorithms exist [McCormack and Sherstyuk 1998; Zanni et al. 2013], we mesh a sphere-mesh (if needed) by contouring the signed distance to the sphere-mesh using a marching cube algorithm.

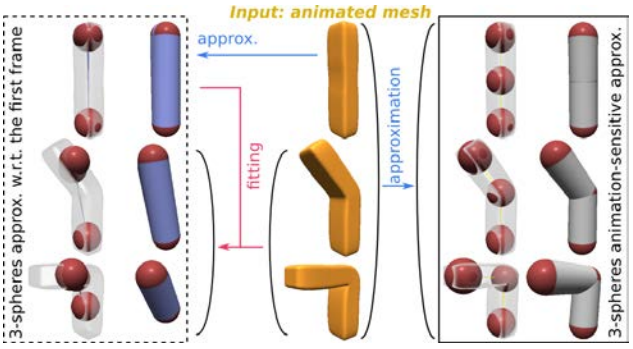


Fig. 2. The input animation is approximated with a sphere-mesh with 3 spheres with time-consistent connectivity. On the left, two spheres on top almost coincide because the sphere-mesh geometry is optimized w.r.t. the first frame, while this choice is not adequate for the complete animation. On the right, the sphere-mesh geometry is optimized w.r.t. all frames simultaneously, resulting in a geometry that is better suited to approximate the input animation.

Rendering. The sphere-mesh primitives are rendered efficiently on the GPU using the geometry shader [de Toledo and Lévy 2008], without any surface extraction.

3. ANIMATED SPHERE-MESHES

Problem statement. We aim at approximating a mesh animation with F frames, (meaning F different triangles meshes having the same connectivity) with an animated sphere-mesh (a mesh where each vertex is associated with a time-varying sphere). The linear interpolation of these spheres across the sphere-mesh triangles and edges results in the animated approximating shape.

Approximation algorithm overview. We cast our shape approximation problem into a partitioning of the barycentric cells P_i^f around each mesh vertex v_i in $\mathcal{M}_0$. The resulting regions of the partitioned animated input mesh are approximated via animated spheres. These spheres, together with the connectivity (deduced from the dual of the partition), results in an animated sphere-mesh.

We initialize the animated sphere-mesh with $\mathcal{M}_0$, and simplify it by iteratively collapsing its edges (equivalently, this means that regions are merged in the animated mesh), while minimizing a cost function. The output is, thus, an animated sphere-mesh hierarchy.

For a given frame $f \in [\![1, F]\!]$, the SQEM Q_i^f for a barycentric cell P_i^f describes the squared distances of the tangent planes in P_i^f:

$$Q_i^f(q,r) := \int_{\xi\in P_i^f} SQEM_{(p_\xi,n_\xi)}(q,r)d\sigma_\xi. \qquad (2)$$

The cost of a partition into K regions $\{I_k, \{\mathbf{s}_k^f\}_{f\leq F}\}_{k\leq K}$ is then

$$\mathfrak{C}(\{I_k, \{\mathbf{s}_k^f\}_{f\leq F}\}_{k\leq K}) := \sum_{k\leq K} \sum_{f\leq F} Q_{I_k}^f(\mathbf{s}_k^f) \qquad (3)$$

where $\{I_k\}_{k\leq K}$ is a partition of barycentric cells and $\mathbf{s}_k^f = (q_f^k, r_f^k)$ is a sphere approximating region I_k in frame f (in the sense that it minimizes $Q_{I_k}^f := \sum_{i\in I_k} Q_i^f$). By combining the cost of all frames, the entire animation is taken into account. This step is crucial (see Fig. 2); a simple three-sphere fit to a capsule in the first frame (dashed box) is less suitable than a three-sphere approximation considering the entire sequence.

Radius constraints. Similarly to [Thiery et al. 2013], we bound each sphere's diameter during the optimization process, to avoid overly large spheres approximating planar regions. This bound, $R(P_{I_k})$ for a region I_k of the barycentric cell partition, is computed based on an analysis of I_k over the animation. Since this step is linked to the spheres optimization, the details are given in Sec. 4. To enforce smoothness in the reconstruction, we also constrain each sphere to have a constant radius over the animation. The motivation is that many animations (e. g., character and animal animations, etc.) are meant to be volume-preserving, in which case the radius should be constant. We discuss this point further in Secs. 4.2 and 5. Formally, it implies:

$$\begin{cases} r(\mathbf{s}_k^f) = r_k \quad \forall\, f \in [\![1, F]\!] \\ 0 \leq\ r_k \leq\ R(P_{I_k}) = \min_{f \in [\![1,F]\!]} R(P_{I_k}^f) \end{cases} \tag{4}$$

Quadrics in $\mathbb{R}^{3F+1}$

To apply the simplification process on animated meshes, we need:

$$Q_{v_i}(\bar{\mathbf{s}}) := \sum_{f=1}^{F} \int_{\xi \in P_i^f} SQEM_{(p_\xi, n_\xi)}(q_f, r) d\sigma_\xi, \tag{5}$$

where $P_i := \{P_i^f\}_{f \leq F}$ is the F-tuple of barycentric cells of vertex v_i and $\bar{\mathbf{s}}$ an animated sphere. In this section, we derive a closed-form expression for this quadric.

Since we enforce that all F instances of the spheres have the same radius, we can denote an *animated* sphere $\bar{\mathbf{s}} \equiv ({q_1}^{\mathrm{T}}, \cdots, {q_F}^{\mathrm{T}}, r)^{\mathrm{T}} \in \mathbb{R}^{3F+1}$, i. e., the concatenation of its F sphere centers $\{q_f\}_{f \leq F}$ and radius r.

Noting that each barycentric cell P_i is contained in the set of triangles $t_j \in T_1(v_i)$, where $T_1(v_i)$ is the set of triangles adjacent to vertex v_i, and that the distance from an oriented point (p_ξ, n_ξ) to a sphere (q_f, r) is constant on any triangle, we can rewrite Eq. 5:

$$Q_{v_i}(\bar{\mathbf{s}}) = \sum_{t_j \in T_1(v_i)} \sum_f w_{ij}^f SQEM_{(p_j^f, n_j^f)}(q_f, r), \tag{6}$$

where p_j^f (resp. n_j^f) denotes the center (resp. normal) of the triangle t_j^f (t_j in frame f), and $w_{ij}^f := \text{area}(t_j^f)/3$.

$Q_{v_i}(\bar{\mathbf{s}})$ is itself an error quadric, as it is the sum of quadrics $Q_{v_i,t_j} := \sum_f w_{ij}^f SQEM_{(p_j^f, n_j^f)}$ for $t_j \cap P_i$. To find an expression for Q_{v_i}, it is thus sufficient to sum up the quadrics Q_{v_i,t_j}, which are given by:

$$Q_{v_i,t_j} =: \frac{1}{2} \bar{\mathbf{s}}^{\mathrm{T}} \cdot \bar{A}_{ij} \cdot \bar{\mathbf{s}} - \bar{b}_{ij}^{\mathrm{T}} \cdot \bar{\mathbf{s}} + \bar{c}_{ij}, \tag{7}$$

with

$$\begin{cases} \bar{A}_{ij} := \left[\begin{array}{c|c|c|c} M_1 & & & N_1 \\ \hline & \ddots & & \vdots \\ \hline & & M_F & N_F \\ \hline {N_1}^{\mathrm{T}} & \cdots & {N_F}^{\mathrm{T}} & W \end{array}\right] & \in \mathcal{S}^{3F+1} \\ \bar{b}_{ij} := \left({b_1}^{\mathrm{T}}, \cdots, {b_F}^{\mathrm{T}}, B\right)^{\mathrm{T}} & \in \mathbb{R}^{3F+1} \\ \bar{c}_{ij} := \sum_f \left({N_f}^{\mathrm{T}} \cdot p_j^f\right)^2 & \in \mathbb{R}, \end{cases} \tag{8}$$

where $\mathcal{S}^n$ denotes the set of symmetric matrices of size $n \times n$ and

$$\begin{cases} M_f := 2 w_{ij}^f n_j^f \cdot {n_j^f}^{\mathrm{T}} & \in \mathcal{S}^3 \\ N_f := 2 w_{ij}^f n_j^f & \in \mathbb{R}^3 \\ W := 2 \sum_f w_{ij}^f & \in \mathbb{R} \end{cases} \quad \begin{cases} b_f := ({N_f}^{\mathrm{T}} \cdot p_j^f) n_j^f & \in \mathbb{R}^3 \\ B := \sum_f ({N_f}^{\mathrm{T}} \cdot p_j^f) & \in \mathbb{R} \end{cases}$$

Importance-driven distribution. Note that it is also possible to introduce a spatially-varying importance function $K^f(\xi)$ in the process (e. g., taking into account the saliency of the mesh at each frame), by setting $w_{ij}^f = \int_{\xi \in P_i^f \cap t_j^f} K^f(\xi) d\sigma_\xi$ (which justifies indexing all terms over both the vertex and the triangle indices in previous equations). Following [Thiery et al. 2013], for all results, we used $K^f(\xi) := 1 + BBD^2({\kappa_{\mathbf{2}}}^f(\xi)^2 + {\kappa_{\mathbf{1}}}^f(\xi)^2)$, where ${\kappa_{\mathbf{2}}}^f(\xi)^2 + {\kappa_{\mathbf{1}}}^f(\xi)^2$ denotes the total curvature at point ξ in frame f and BBD the average over the F frames of the bounding-box diagonal length. This measure prevents very small structures from disappearing too early during the optimization.

4. APPROXIMATION ALGORITHM

The simplification algorithm partitions the original mesh $\mathcal{M}_0$ into regions by successively simplifying a sphere-mesh. The initial sphere-mesh is defined as the input mesh, along with the quadrics Q_i associated with the barycentric cells of each vertex v_i (following Eq. 2). The spheres $\bar{s}_i$ of the sphere-mesh are optimized as $\bar{s}_i = \text{argmin}_{\bar{s}} Q_i(\bar{s})$. The resulting approximation error is then $Q_i(\bar{s}_i)$. The closed-form solution for the minimization process is presented in Sec. 4.1.

The algorithm performs one edge collapse of the sphere-mesh at a time, the one leading to the lowest approximation error. To this extent, each possible edge collapse together with the resulting approximation error is placed in a priority queue $\mathcal{Q}$, i. e., given an edge linking two vertices u and v with corresponding error quadrics Q_u and Q_v, this edge (u, v) is placed in $\mathcal{Q}$ with the approximation error for the quadric $Q_u + Q_v$ as a priority.

When taking the top edge (u, v) from $\mathcal{Q}$, we collapse it in the sphere-mesh to a new vertex w with $Q_w := Q_u + Q_v$. Remaining edge-collapse suggestions involving u and v are removed from $\mathcal{Q}$ while, for all edges from w to a neighbor x, the (w, x) edge-collapse suggestion is added to $\mathcal{Q}$. The algorithm iterates until $\mathcal{Q}$ is empty. Notice that Q_w implicitly encodes the sphere associated with w: the mesh structure mainly encodes connectivity, while position and radius at vertices can be derived via the minimization of the quadric error.

Radius bound computation. Following [Thiery et al. 2013], when determining the minimizing sphere, we bound the maximal radius to avoid flat regions being approximated by overly large spheres, as mentioned in Sec. 3. Specifically, for two vertices u and v with associated regions P_u and P_v, Thiery et al. suggest setting the radius bound to $\mathbf{R} := \mathbf{3/4}\mathcal{W}(\mathbf{P_u} \cup \mathbf{P_v})$, where $\mathcal{W}(\mathcal{X})$ denotes the width of the set $\mathcal{X}$ in $\mathbb{R}^3$.

The width, i. e., the minimum extent over all directions [Gärtner and Herrmann 2001], is computed as follows. At the initialization stage, we compute the extent of the barycentric cell P_i^f of the vertices v_i along a given set of directions $k_j \in \mathbb{R}^3$ ($|k_j| = 1$). The extents are given by $M_j(P_i^f) := \max_{\xi \in P_i^f}(\xi^{\mathrm{T}} \cdot k_j)$ and $m_j(P_i^f) := \min_{\xi \in P_i^f}(\xi^{\mathrm{T}} \cdot k_j)$, and are used to define a width $\mathcal{W}(P_i^f) := \min_j |M_j(P_i^f) - m_j(P_i^f)|$ for frame f. Finally, the width $\mathcal{W}(P_i) := \min_f(\mathcal{W}(P_i^f))$ is defined as the minimum over the F frames. The width of $P_u^f \cup P_v^f$ is computed from the extents of P_u^f and P_v^f, which are obtained via $M_j(P_u^f \cup P_v^f) = \max(M_j(P_u^f), M_j(P_v^f))$ and $m_j(P_u^f \cup P_v^f) = \min(m_j(P_u^f), m_j(P_v^f))$. During a collapse, we update the extents for direction k_j, following these formulae.

Triangle inversion. Additionally, we suggest forbidding collapsing two vertices as soon as it induces triangle inversions. For each vertex v, we check all triangles $t \in T_1(v)$ and detect an inversion when $\sum_f \sqrt{\text{area}(t_f)\text{area}(t'_f)} n(t_f)^{\mathrm{T}} \cdot n(t'_f) < 0$, with t' being the geometry of t after the collapse. In other words, we test if large triangles are inverted for a significant amount of frames.

4.1 Sphere Optimization

Here, we describe the derivation of the minimizer $\text{argmin}_{\bar{s}}\, Q(\bar{s})$, under the radius constraints $0 \leq r \leq R$.

4.1.1 *General case.* For convenience, in the following, we keep the notations introduced in Eq. 8 to describe the block elements of the matrix $\bar{A}$ and the vector $\bar{b}$, which correspond to the quadric we want to minimize (the block structure never changes).

Quadric Mininimization (invertible):.
Minimize $E(\bar{\mathbf{s}}) = \frac{1}{2}\bar{\mathbf{s}}^{\mathrm{T}} \cdot \bar{A} \cdot \bar{\mathbf{s}} - \bar{b}^{\mathrm{T}} \cdot \bar{\mathbf{s}}$, with $\bar{A}, \bar{b}$ of the form given by Eq. 8, subject to $0 \leq r \leq R$.

Assuming all block matrices M_f are invertible, the global minimizer $\bar{\mathbf{s}}^*$ (without inequality constraints) is given by $\bar{A} \cdot\ \bar{\mathbf{s}}^* = \bar{b}$ ($\Leftrightarrow \overrightarrow{\nabla}_{\bar{s}} E(\bar{s}^*) = \vec{0}$), which leads to:

$$\begin{cases} r^* = \frac{B - \sum_f {N_f}^{\mathrm{T}} \cdot M_f^{-1} \cdot b_f}{W - \sum_f {N_f}^{\mathrm{T}} \cdot M_f^{-1} \cdot N_f} \\ q_f^* = M_f^{-1} \cdot (b_f - r^* N_f) \qquad \forall f \in [\![1, F]\!] \end{cases}$$

The solution to the problem with inequality constraints is given by

$$\begin{cases} r = \min(\max(r^*, 0), R) \\ q_f = M_f^{-1} \cdot (b_f - r N_f) \qquad \forall f \in [\![1, F]\!] \end{cases} \tag{9}$$

If the denominator $(W - \sum_f {N_f}^{\mathrm{T}} \cdot M_f^{-1} \cdot N_f)$ in the expression of r^* is null, we set $r = 0$ and $q_f = M_f^{-1} \cdot b_f\ \forall f$. However, this measure is only a sanity check to avoid numerical instabilities and this case rarely occurred in our experiments.

4.1.2 *Degenerate cases.* Care must be taken if some of the matrices M_f are non-invertible (i. e., the quadrics are degenerate). For the sake of simplicity, we assume that $[\![1, D]\!]$ are the indices of the degenerate quadrics.

In the context of edge-collapse simplification, such cases are usually handled by constraining the optimal sphere position q_f to be located on the edge (u_f, v_f) which is about to be collapsed [Garland and Heckbert 1997]: $q_f = u_f + \lambda_f \vec{d_f}$, with $\vec{d_f} := v_f - u_f$. This restriction leads to a quadratic problem of size $D + 3(F - D) + 1$ with $2D + 2$ linear inequality constraints. Solving this problem via active-set methods (see [Lawson and Hanson 1974]) is very costly for high-dimensional problems with a high number of inequalities and, in our experiments, it happened that all F quadrics were degenerate. Therefore, we constrain all variables λ_f to equal a single value λ, i. e., $q_f = u_f + \lambda \vec{d_f}\ \forall f \in [\![1, D]\!]$, leading to the following quadratic problem of size $3(F - D) + 2$:

Quadric Minimization (not invertible):.
Minimize $E(\bar{\mathbf{s}}) = \frac{1}{2}\bar{\mathbf{s}}^{\mathrm{T}} \cdot \bar{A} \cdot \bar{\mathbf{s}} - \bar{b}^{\mathrm{T}} \cdot \bar{\mathbf{s}}$, with $\bar{A}, \bar{b}$ of the form given by Eq. 10, subject to $0 \leq \lambda \leq 1$ and $0 \leq r \leq R$.

$$\begin{cases} \bar{A} = \left[\begin{array}{c|ccc|c} \mu & & & & \nu \\ \hline & M_{D+1} & & & N_{D+1} \\ \hline & & \ddots & & \vdots \\ \hline & & & M_F & N_F \\ \hline \nu & {N_{D+1}}^{\mathrm{T}} & \cdots & {N_F}^{\mathrm{T}} & W \end{array}\right] & \in \mathcal{S}^{3(F-D)+2} \\ \bar{b} = (\beta_1, {b_{D+1}}^{\mathrm{T}}, \cdots, {b_F}^{\mathrm{T}}, \beta_2)^{\mathrm{T}} & \in \mathbb{R}^{3(F-D)+2} \\ \bar{s} = (\lambda, {q_{D+1}}^{\mathrm{T}}, \cdots, {q_F}^{\mathrm{T}}, r)^{\mathrm{T}} & \in \mathbb{R}^{3(F-D)+2} \end{cases} \tag{10}$$

with the various scalars $\mu, \nu, \beta_1, \beta_2$ defined as

$$\begin{cases} \mu := \sum_{f \leq D} \vec{d_f}^{\mathrm{T}} \cdot M_f \cdot \vec{d_f} & \nu := \sum_{f \leq D} {N_f}^{\mathrm{T}} \cdot \vec{d_f} \\ \beta_1 := \sum_{f \leq D} (b_f - M_f \cdot u_f)^{\mathrm{T}} \cdot \vec{d_f} & \beta_2 := B - \sum_{f \leq D} {N_f}^{\mathrm{T}} \cdot u_f \end{cases}$$

For this problem, active set methods are efficient because, despite its size, the number of inequalities is low. Alternatively, we provide a closed-form solution for the minimization, which we used for all examples and results, in Appendix A.

4.2 Temporal coherence

The use of a constant sphere radius implicitly captures deformations and leads to a high consistency. Nonetheless, the position of the spheres being optimized per-frame can lead to loss of temporal *correlation* with the input at very coarse scales, depending on the input. This section describes an algorithm to improve temporal coherence, while only slightly reducing approximation accuracy.

One could enforce a smooth trajectory for each sphere. However, this approach is restrictive because it is sensitive to the time-sampling of the input and not all sequences are smooth animations. Further, enforcing directly null time-derivatives on trajectories results in unnatural and overly smooth animations.

Instead, we add a soft-attach term to the center of the sphere in each frame during the optimization, biasing its position towards a point moving consistently along the animation of the mesh. To this extent, we choose the barycenter of the corresponding region in $\mathcal{M}_0$, to which the vertex in mesh $\mathcal{M}_\tau$ is associated with. Hereby, we avoid assumptions about the sampling of the input animation and mimic the temporal coherence of the input.

Formally, for an edge (u, v) with corresponding error quadrics Q_u and Q_v, the associated quadric becomes $Q_{uv} = Q_u + Q_v + \delta_{\mathbf{TC}} q_{uv}$ instead of $Q_u + Q_v$, with $\delta_{\mathbf{TC}}$ the temporal coherence weight. Let σ_u^f and c_u^f be the area and the mean point of the region P_u corresponding to the vertex u in frame f, we then define q_{uv}:

$$q_{uv}(\bar{\mathbf{s}}) = \sum_{f \leq F} \sigma_{uv}^f ||q_f - c_{uv}^f||^2, \tag{11}$$

where $\sigma_{uv}^f = \sigma_u^f + \sigma_v^f$, $c_{uv}^f = \frac{\sigma_u^f c_u^f + \sigma_v^f c_v^f}{\sigma_u^f + \sigma_v^f}$. In other words, $q_{uv}(\bar{\mathbf{s}})$ defines the sum of the squared distances between the sphere centers and $\{c_{uv}^f\}_{f \leq F}$ over all frames. In practice, following Eq. 8, the following elements need to be added to the quadric $Q_u + Q_v$:

$$\begin{cases} M_f \leftarrow M_f + 2\delta_{\mathbf{TC}} \cdot \sigma_{uv}^f \cdot I_3 & \forall f \leq F \\ b_f \leftarrow b_f + 2\delta_{\mathbf{TC}} \cdot \sigma_{uv}^f \cdot c_{uv}^f & \forall f \leq F \\ \bar{c} \leftarrow \bar{c} + \delta_{\mathbf{TC}} \cdot \sum_{f \leq F} \sigma_{uv}^f ||c_{uv}^f||^2 \end{cases} \tag{12}$$

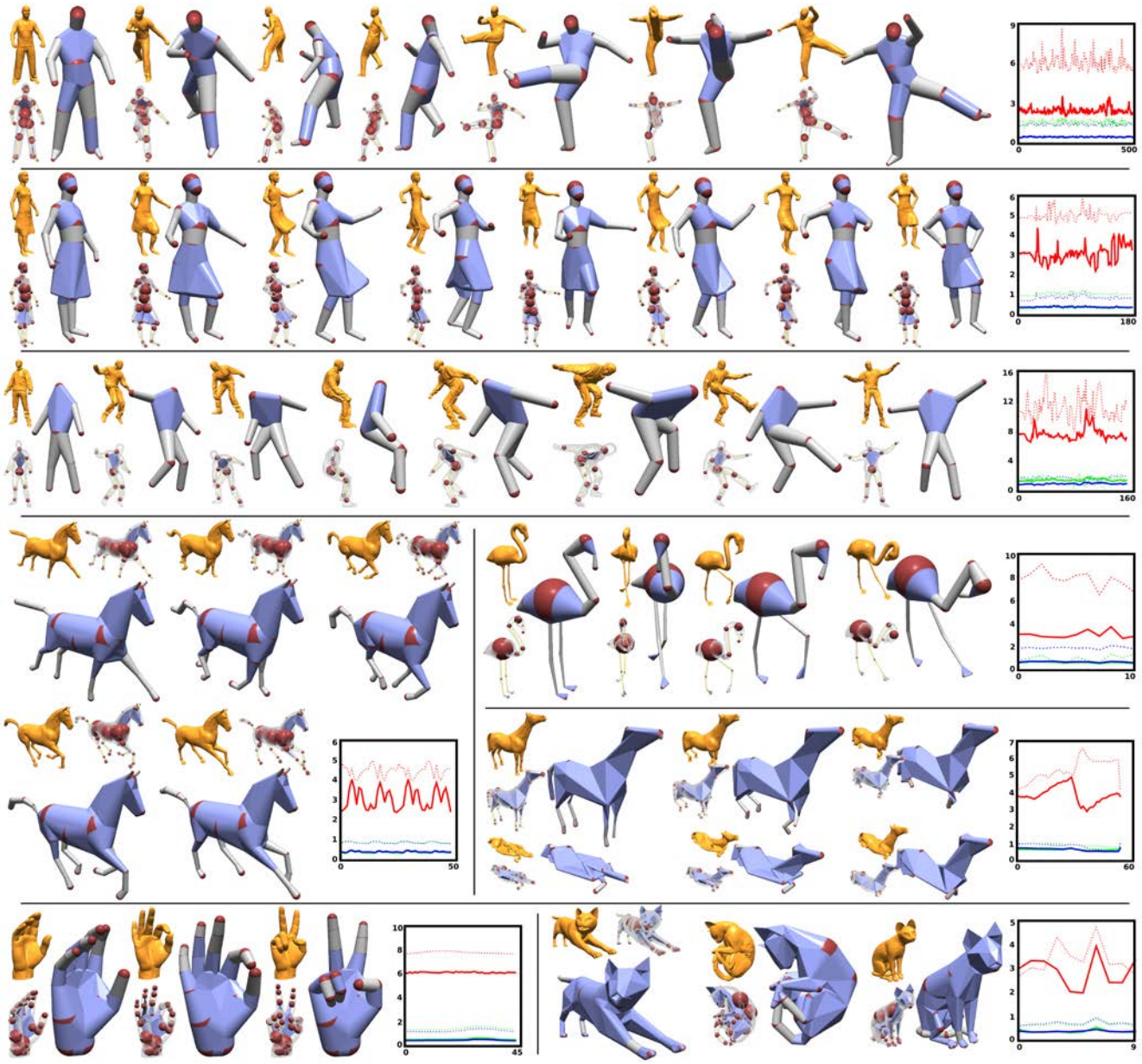


Fig. 3. with $\delta_{\mathbf{TC}} = \mathbf{0}$, **no connectivity optimization**. For each animation, we show the input mesh (gold), the constructed sphere-mesh with transparent input, and the interpolated sphere-mesh. Solid (resp. dashed) curves plot distances between our (resp. QEM) approximation and the input. Red curves: Hausdorff distance (H), blue curves: mean distance from the approximation to the original model (M21), green curves: mean distance from the original model to the approximation (M12) (distances in percentages of the bounding-box diagonal).

4.3 Connectivity optimization

Most edge collapses, especially for the first levels, maintain a good connectivity. A verification after each edge collapse is thus not justified due to the involved costs. However, at coarse scales, undesired artifacts in the form of large and thin triangles crossing the original surface can occur, which is a typical pitfall of edge-decimation-based simplification algorithms. These artifacts are more likely to occur whenever the associated regions in $\mathcal{M}_0$ differ significantly from the Voronoï regions of the vertices (in our context, spheres) of $\mathcal{M}_\tau$. This last observation motivated the use of (centroïdal-) Voronoï tesselation algorithms, which are considered the most effective for 2D surfaces and 3D volumes remeshing. It also inspires ours, which exhibits a similar behaviour.

We propose to optimize the connectivity only after selecting the desired level of simplification τ. We start by associating each vertex v_i of the original mesh $\mathcal{M}_0$ to a set of candidate sphere indices $S(i)$ in $\mathcal{M}_\tau$, which are the sphere to which v_i was collapsed and all its adjacent spheres in $\mathcal{M}_\tau$. This set is relatively small in practice, as it is bounded by the highest vertex degree in $\mathcal{M}_\tau$ plus one. For each

Table I. Performance and timings for ou animated sphere-mesh approximation algorithm. The initialization time excludes the animation parsing. #S / #E / #T: number of spheres, wire edges, and triangles in the output sphere-mesh. We compare our approximation with QEM simplification for the same number of primitives (smallest error in **bold**). H, M21 ,M12 (see Fig. 3) are averaged over the animation.

Input Anim.	(#V / #T / #F)	Init. (ms)	Decim. (ms)	Animated Sphere-Mesh (#S / #E / #T)	H	M12	M21	QEM Simplification (#V / #T)	H	M12	M21
Capoeira	(19988 / 39972 / 499)	65338	186482	20 / 9 / 12	**2.45**	**0.46**	**0.47**	20 / 31	6.05	1.58	1.38
Samba	(9971 / 19938 / 175)	8037	16302	40 / 12 / 46	**3.05**	**0.37**	**0.39**	40 / 60	4.97	1.05	0.86
Jump	(10002 / 20000 / 150)	7221	15488	10 / 6 / 4	**7.62**	1.53	**1.02**	10 / 10	11.11	**1.47**	1.76
Flamingo poses	(26394 / 52895 / 11)	2252	5628	20 / 8 / 10	**3.03**	**0.62**	**0.67**	20 / 21	7.79	0.96	1.89
Horse-gallop	(8431 / 16843 / 48)	2338	5394	46 / 17 / 50	**3.06**	**0.41**	**0.43**	46 / 76	4.47	0.89	0.89
Horse-collapse	(8431 / 16843 / 54)	2554	4915	46 / 9 / 61	**3.84**	**0.65**	**0.66**	46 / 76	5.29	0.93	0.85
Hand	(7929 / 15855 / 44)	2120	4412	34 / 8 / 44	**6.11**	**0.55**	**0.47**	34 / 62	7.77	1.44	1.26
Cat poses	(7207 / 14410 / 10)	489	1283	85 / 5 / 142	**2.86**	**0.38**	**0.37**	85 / 166	3.34	0.66	0.70

vertex v_i an index s_i among $S(i)$ is chosen as the one minimizing the sum of distances along the animation:

$$s_i = \underset{(\{q_f\}_{f\leq F},r)\in S(i)}{\operatorname{argmin}} \sum_{f\leq F} \Big(||v_i^f - q_f|| - r \Big)$$

The connectivity of $\mathcal{M}_\tau$ is redefined entirely based on the dual of the sphere-index labeling in $\mathcal{M}_0$: **(i)** for each triangle (v_a, v_b, v_c) of $\mathcal{M}_0$ whose vertices are labelled with three different spheres s_a, s_b and s_c, the corresponding triangle (s_a, s_b, s_c) and edges are added, unless they already exist; **(ii)** for each triangle or edge of $\mathcal{M}_0$ whose vertices have two different sphere labels, a corresponding edge is added to $\mathcal{M}_\tau$, unless it already exists.

5. RESULTS

We implemented our shape approximation method in C++ and report its performance on an Intel Core2 Duo running at 2.5 GHz with 4GB of main memory. The algorithm is automatic and computes the full hierarchy. Afterwards, the user controls one parameter (the target number of spheres). In Fig. 3, we show results on performance-capture, as well as on synthetic animations.

We report timings and distances to the input (computed with *Metro* [Cignoni et al. 1998]) in Table I, which serve as numerical comparisons to QEM simplification. For a fair comparison, we allow triangles to degenerate to wire edges for QEM. However, it results in invalid shapes, since QEM does not model volumes.

Sphere-meshes tend to better approximate shapes than traditional surface meshes for low number of primitives. We roughly halve reconstruction errors for the Capoeira, Samba, Jumping and Flamingo sequences, using various numbers of spheres ranging from 10 to 42. In contrast, if many primitives are required (with respect to the geometric complexity of the shape), sphere-meshes tend to have smaller reconstruction-error ratios (e. g., roughly 1.2 for the Hand and Cat sequences, when approximating with 32 and 85 spheres respectively). Sphere-meshes and traditional QEM-driven triangle meshes start with the same animated input mesh. Hence, they are basically equivalent at the first levels of simplification but, with a decreasing number of primitives, sphere-meshes evolve progressively from surface to volumetric structures.

Interestingly, when comparing the *Horse-gallop* and *Horse-collapse* sequences, which both contain 46 spheres, we observe that animated sphere-meshes tend to become volumetric if the input mesh deformations are volume preserving. Consequently, the sphere-mesh nicely adapts and behaves either more like a skeleton or like rigs depending on the type of deformation; e. g., a skeleton could animate efficiently the *Horse-gallop* sequence, whereas a set of surface rigs would be more efficient at animating the *Horse-collapse* sequence.

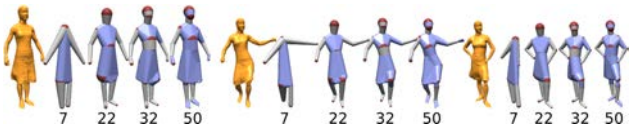


Fig. 4. Samba model animation approximated with animated sphere-meshes with 7, 22, 32 and 50 spheres.

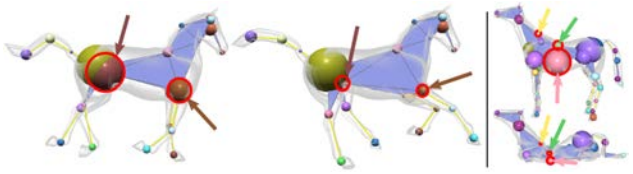


Fig. 5. Results obtained with a time-varying sphere radius.

Fig. 4 illustrates a similarly challenging example because the *Samba* model contains large tubular parts (legs, arms, etc.), which can be approximated by single edges, while the dress model is better represented using surface structures i. e., it is highly deformed in a non-rigid way during the animation. Even with a low number, such as 22 spheres, our sphere-meshes recover the fine details of the dress as well as the simpler arm and leg structures. When using only 7 spheres, an abstract structure taking a plausible skeleton form emerges. Note that the extracted surface remains manifold, even in presence of wire-edges in the sphere-mesh.

Analysis. Fig. 5 shows results where each sphere is allowed to have a time-varying radius, which is bounded per-frame only. The left side shows how strongly the resulting sphere-mesh varies, even for "volume-constant" animations. The right side shows that the resulting sphere-mesh no longer exhibits a proper structure in the collapsed parts, which leads to an unnatural connectivity as well (see Fig. 3 for a comparison to our approach). Our solution results in an improved sphere placement and behavior during animation, which makes it applicable for skinning decompositions (see next Section). The latter would be impossible when using varying radii.

Fig. 6 illustrates the effect of our temporal coherence method. Connectivity and preserved features might change for the same number of spheres and produce more natural simplifications for input animations exhibiting strong temporal coherence. In particular, as emphasized within the figure, spheres remain in the vicinity of the same input vertices and avoid sliding over the geometry. In this

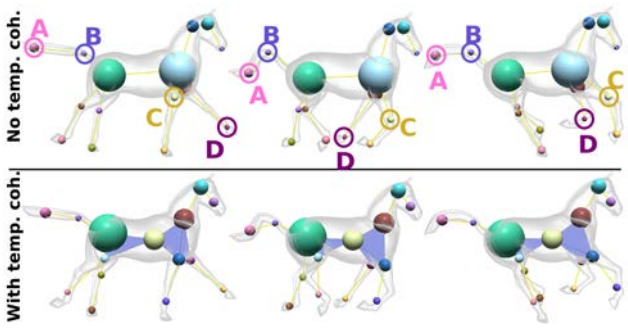


Fig. 6. Comparison of results obtained without (top row) and with temporal coherence (bottom row, $\delta_{\mathbf{TC}} = \mathbf{10^5}$) using 15 spheres. Notice how, without our temporal coherence method, the spheres (A,B,C,D) slide along the mesh to best fit the input mesh.

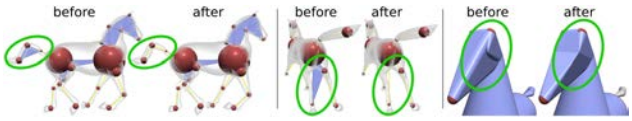


Fig. 7. Comparison of results obtained without and with connectivity improvement for various settings.

example, it is also visible that our solution is less dependent on the time sampling of the input sequence. Our temporal constraints are based on the coherence of the original input, which makes sure that no overly smooth motion is enforced where it is not present. Consequently, the use of the temporal coherence constraint also leads to more constant-length edges in the simplified animated sphere-mesh over the course of the animation, at the cost of a slighty less optimal geometric approximation.

Fig. 7 illustrates the impact of the connectivity optimization step. While the Hausdorff errors might already be acceptable before this optimization, and are not necessarily reduced significantly by a modification of the sphere-mesh connectivity, some elements might be undesirable. An example is the presence of primitives, such as elongated triangles, which cross the input surface or lead to locally inverse connectivity. Fig. 7 illustrates several such cases. In the first example (left), the tail of the horse, defined as a long triangle, crosses the surface due to a non-convex partitioning of the input mesh $\mathcal{M}_0$. The second example (middle) depicts a triangle which introduces an unwanted link, with the ankle of the left leg being connected to the hip of the right leg. Finally, the third example (right) shows that, due to inadequate local connections, complex geometry inversions are introduced. All these connectivity problems are solved by our non-parametric optimization step, as illustrated next to the circled issue.

6. APPLICATIONS

Our algorithm is the first to approximate automatically animated surfaces using an animated convolution surface. Existing applications using this representation can thus trivially be used directly with our data, when they could previously use static or manually-designed animated convolution surfaces only. The first application we showcase, approximate collision detection with animated meshes (Sec. 6.1), is a simple illustration of this. The second application we present, skinning decomposition using our sphere-mesh as decomposition domain (Sec. 6.2), is more involved, and relies on the fact that our approximation method outputs geometries which

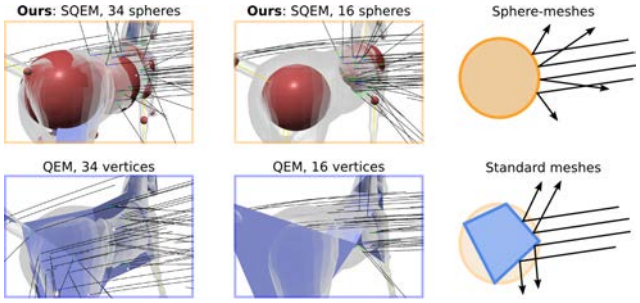


Fig. 8. We compare the use of an animated sphere-mesh (a,b,c) against a standard mesh (d,e,f) for approximate collision detection with particles. The normal field on large tubular structures is smooth, and so are the resulting bounces on the sphere-mesh (compare SQEM with QEM for the same number of primitives). When using 16 vertices (e), a large number of particles go through the input shape and fail to bounce, whereas our 16-sphere approximation (b) provides acceptable results.

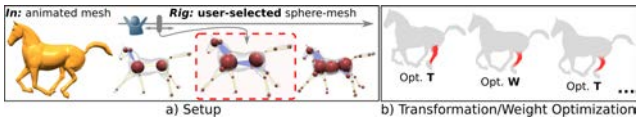


Fig. 9. **a)** The input for the skinning decomposition is a **user-selected** LoD of the animated sphere-mesh hierarchy, which is traversable in real-time. **b)** A few iterations of successive optimizations of the bones' transformations (**T**) and the vertex' weights (**W**).

are compatible with the specific nature of the input animations (as already discussed when comparing the *Horse-gallop* and *Horse-collapse* sequences in Sec. 5).

6.1 Approximate collision detection

Our animated sphere-mesh can be used as a direct low-budget substitute for collision detection with the animated object. By testing the distance from a point to the primitives of the animated sphere-mesh directly, approximate collision detection can be achieved at high frame-rates without the building of any complex structure.

Fig. 8 illustrates the advantages of such a solution for approximate collision detection. An animated sphere-mesh with few spheres presents enough geometric details to be a cheap, yet plausible, substitute to the complete animated mesh. Compared to low detail surface meshes (see Fig. 8), the normal field of low detail sphere-meshes is smooth enough to avoid directional artifacts after the bounces of the particles. In Appendix B, we give formulas for the construction of the sphere-mesh primitives, as well as for intersecting them with particles (represented as spheres on segments).

6.2 Skinning decomposition with rigid bones

We use a strategy similar to [Le and Deng 2012] and decompose the animation on our sphere-mesh via linear-blend skinning (LBS). The bones for LBS correspond to the spheres of the sphere-mesh, which is provided by the user as input for our decomposition scheme. The user can therefore chose any level of complexity he desires to obtain for the decomposition. The output of the decomposition is a set of weights $\{w_{ij}\}$, which rig the rest pose to the sphere-mesh, and bone transformations $\{R_j^f, T_j^f\}$, which, together with the weights, allow for reconstruction of the input animation. Further, the artist can use the derived weights with the sphere-mesh

to create new animations exhibiting the input animation's partial motion.

Notations: v_i^f is the position of vertex i in frame f, μ_i^f is the area of its barycentric cell in frame f, $B(i)$ is the set of bones that have an influence over vertex i, w_{ij} is the weight of bone j at vertex i, p_i is the rest pose vector of vertex i in $\mathbb{R}^3$ (we initialize the rest pose with the first frame), and R_j^f and T_j^f are the rotation and the translation of the bone j in frame f. Please note the slight abuse of notation, allowing us to consider a bone j as an index.

Constraints: $\{w_{ij}\}$ are optimized under a *non-negativity constraint* ($w_{ij} \geq 0 \ \ \forall i,j$), an *affinity constraint* ($\sum_j w_{ij} = 1 \ \ \forall i$), and a *sparsity constraint*,i. e., a small set $B(i) := \{j | w_{ij} \neq 0 \ \forall j\}$, which is the set of bones having an influence on v_i. While the sparsity mostly serves acceleration purposes, the other two constraints are important for stability during editing processes.

In contrast to [Le and Deng 2012], we fix $B(i)$ once and for all based on the animated sphere-mesh, which serves as the skeletal domain for the decomposition. For a given vertex v_i in the first frame, we define $B(i)$ as the set of those sphere-mesh spheres whose Voronoï cells on the input mesh are adjacent to the cell in which v_i is located. In other words, all adjacent spheres have a potential influence, but none of the others.

Fig. 9 illustrates the optimization process. At first, the user picks a level-of-detail from the animated sphere-mesh hierarchy, which is an intuitive way to give control over the degrees of freedom desired in the output skinning decomposition. At each step, we optimize the vertex weights, while fixing the bones' transformations R_j^f and T_j^f. Next, we optimize these bone transformations bone-by-bone and frame-by-frame, while fixing the vertex weights. Finally, we also optimize for the rest pose after a few (typically 5) iterations.

Transformations optimization. Similarly to [Le and Deng 2012], we optimize the transformations of the bones one after the other. When optimizing the one of bone j, we minimize

$$E_{R_j^f, T_j^f} = \sum_{i \in I(j)} \mu_i^f ||w_{ij}(R_j^f \cdot p_i + T_j^f) - q_{ij}^f||^2 \,,$$

for all frames f, where q_{ij}^f is the resulting point without the contribution of bone j: $q_{ij}^f := v_i^f - \sum_{k \in B(i), \neq j} w_{ik}(R_k^f \cdot p_i + T_k^f)$, with R_j^f and T_j^f as variables, $I(j)$ denoting the vertices that are influenced by the bone j (i. e., $I(j) = \{i | j \in B(i)\}$), under the only constraint that R_j^f is a rotation. Details for this optimization can be found in [Le and Deng 2012], up to the fact that the reader should introduce the various weights $\{\mu_i^f\}$ in the formulas.

Minor bones As explained in the work of Le and Deng [2012], such a transformation-optimization process can introduce instabilities with bones that have minor influence over vertices ($\{j | \sum_i w_{ij}^2 < \epsilon\}$). We detect them using the same criterion (bone j is unstable if $\sum_i w_{ij}^2 < 3$), and solve the issue by setting the transformation of the bone to an average of the transformations of its adjacent bones (following the sphere-mesh connectivity).

Vertex rest-pose optimization. We optimize the rest pose p_i of each vertex v_i, by minimizing for each vertex v_i independently

$$E_{p_i} = \sum_f \mu_i^f ||(\sum_{j \in B(i)} w_{ij} R_j^f) \cdot p_i - (v_i^f - \sum_{j \in B(i)} w_{ij} T_j^f)||^2 \,,$$

which boils down to solving a simple quadratic system in $\mathbb{R}^3$.

Mixed Weights optimization. Previous work (e. g., [Le and Deng 2012]) optimize weights with the sole purpose of reconstructing the input animation. The control structure (i. e., the bones) and the weights, which result from this process, exhibit a level of complexity that is directly driven by the complexity of the input animation (i. e., a completely rigid animation will result in a single bone with all mesh vertices having a weight of 1 w.r.t. this bone, rendering this decomposition useless for artistic editing, for example).

In contrast, we provide the control structure as input to our skinning decomposition scheme, and we *enforce* the desired level of complexity of the weight maps. We do so by initializing our weights with *geometric weights* (we use [Baran and Popović 2007], but our method makes no assumptions on the input geometric method, and other geometric skinning weights could be used instead (e. g., [Jacobson et al. 2011]), and progressively refine our weights as to better fit the input animation.

The core idea is that geometric weights should be favored in regions of the mesh where they are able to reproduce the input animation, as they allow for more general transformations. In other regions of the mesh, where these are not able to reproduce the input animation well, the weights should be learned from the animation.

We optimize the weights $\{w_{i\cdot}\}^{t+1}$ of vertex i at step $t+1$ by minimizing the following energy:

$$\begin{aligned} E_{w_i}^{t+1} =& \alpha_i^{t+1} \Big(\sum_f \mu_i^f || \sum_{j \in B(i)} w_{ij}^{t+1} (R_j^f \cdot p_i + T_j^f) - v_i^f ||^2 \\ &+ \gamma \sum_f \mu_i^f \sum_{j \in B(i)} (w_{ij}^{t+1} - \tilde{w}_{ij}^t)^2 \Big) \\ &+ (1 - \alpha_i^{t+1}) \gamma \sum_f \mu_i^f \sum_{j \in B(i)} (w_{ij}^{t+1} - w_{ij}^t)^2, \end{aligned}$$

where $\gamma := 10^{-4} BBD^2$ is a normalization constant that compensates for the differences in the range between reconstruction and weight errors based on the bounding box diagonal, $\tilde{w}_{ij}^t := \sum_{v_k \in B(v_i; 3\sigma)} h_\sigma(v_i, v_k) w_{kj}^t$ is a filtered version of $\{w_{ij}^t\}$ (we use a Gaussian kernel over a geodesic disk centered in v_i, with σ equaling 3% of the bounding box diagonal), and α_i^{t+1} is the blending factor between animation-driven weights and the geometric alternative. α_i^{t+1} should be close to one if information needs be extracted from the input animation. Otherwise, α_i^{t+1} should be low, leading to mostly geometric weights being used.

Our strategy to determine α_i^{t+1} is as follows. We first compute the weights $\hat{w}_{ij}^{t+1}$ for all vertices v_i with $\alpha_i = 1$. Next, we compute the reconstruction errors of the vertices v_i when considering the newly-found weights $\hat{w}_{ij}^{t+1}$ and the weights w_{ij}^t of the previous iteration:

$$\begin{cases} e_i^{\text{before}} = \sum_f \mu_i^f || \sum_j w_{ij}^t (R_j^f \cdot p_i + T_j^f) - v_i^f ||^2 \\ e_i^{\text{after}} = \sum_f \mu_i^f || \sum_j \hat{w}_{ij}^{t+1} (R_j^f \cdot p_i + T_j^f) - v_i^f ||^2 \end{cases}$$

Finally, we set a large value to α_i for the vertices v_i where the reconstruction error decreases significantly when compared to the differences in the weights $\delta_i = ||\{\hat{w}_{ij}^{t+1}\}_j - \{w_{ij}^t\}_j||^2$. Specifically, noting $\gamma_i = (e_i^{\text{before}} - e_i^{\text{after}})_+$ the decrease of the reconstruction error of vertex v_i, $\gamma = \max_i \gamma_i$ the maximum decrease and $\delta = \max_i \delta_i$ the maximum weights change, we set $\alpha_i^{t+1} = \frac{0.8 \gamma_i / \gamma}{\gamma_i / \gamma + \delta_i / \delta}$.

Results. Fig. 10 shows the skinning reconstruction of the *Horse-gallop*, *Horse-collapse* and *Samba* inputs decomposed onto sphere-meshes with 46 spheres. Errors (blue curves) are computed as

$$MSE(f) = \frac{\sum_i \mu_i^f || \sum_j w_{ij} (R_j^f \cdot p_i + T_j^f) - v_i^f ||^2}{BBD^2 \sum_i \mu_i^f} \qquad (13)$$

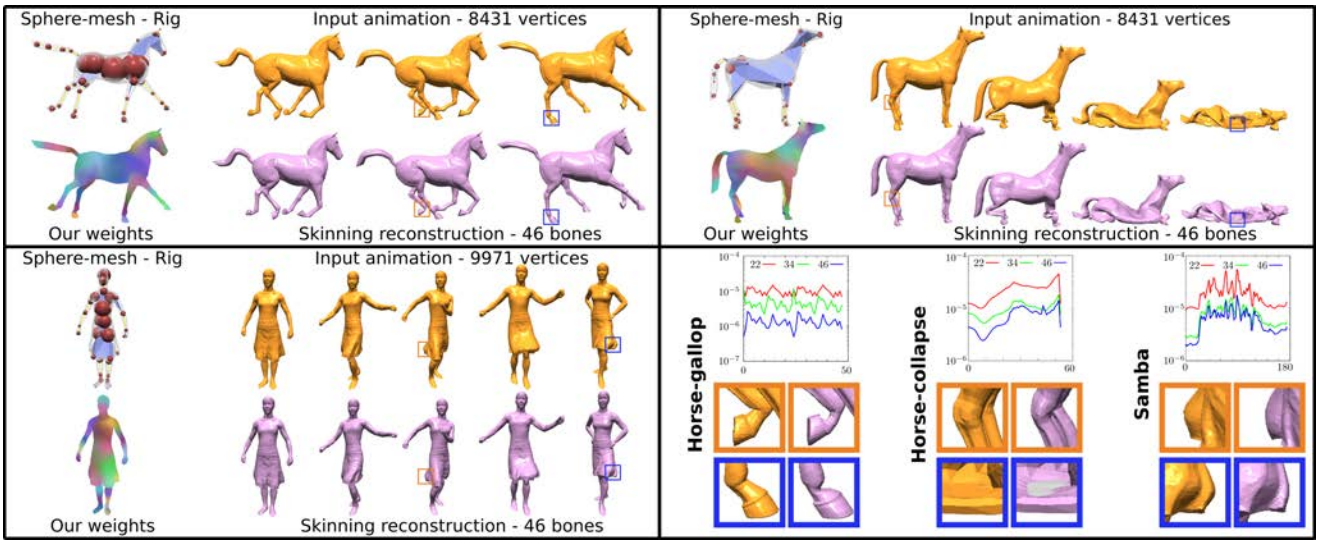


Fig. 10. Skinning reconstructions of three sequences with 46 bones. Close-ups show artifacts that are essentially loss of volume or presence of self-intersections. **Bottom right:** mean squared reconstruction error for 46 (blue), 34 (green) and 22 (red) bones. y axis is logarithmic.

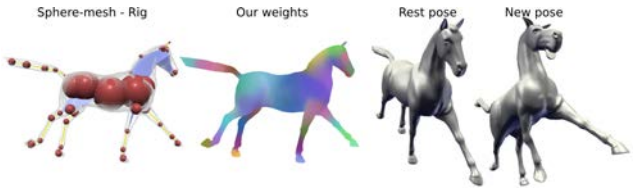


Fig. 11. **Pose editing using mixed weights:** skinning weights learned solely from the *Horse* motion provide a faithful decomposition for the body and the legs, but lack a proper layout for the head, while our mixed weights also account for geometry and provide the necessary degrees of freedom to alter the grouped geometric features, even if they collectively undergo the same motion.

We also plot the errors for skinning decompositions with 34 (green curves) and 22 (red curves) spheres. As pointed out in Sec. 5, the two horse sequences strongly differ in shape, local density and exhibit differences locally in their type of structures (volumetric/surface), even though both input animations share the same mesh topology and connectivity. These examples show the strength of our approach: when converting the mesh animations into linear blend skinning, our animated sphere-mesh adapts automatically and appears as a flexible rig structure that naturally takes the deformations of the input mesh into account, resulting in either a more volumetric or a more surface-oriented representation.

Using the animated sphere-mesh as a prior to derive the bones impacting the reconstruction of each original vertex leads to a fast algorithm: all our skinning decompositions were performed in a couple of minutes. Moreover, by initializing our iterative optimization scheme with smooth geometric weights, we output weight maps that are efficiently learned from the animation wherever it is pertinent, while falling back to the geometric weights elsewhere.

Fig.11 shows an example of a new pose generated using our mixed weights. While traditional skinning decomposition methods can learn weights on the body of the model based on its motion in the raw sequence, they fail at providing the user with deformation handles in the parts of the shape that are deformed rigidly, even if they exhibit interesting structures (e. g., the head). On the contrary, our mixed weights account for both details in the motion and in the geometry, leading to an optimized mix of both properties.

Table II. Our method takes a geometry (GEO) as input parameters for the decomposition, whereas others take a number of bones (NOB). It is the only one to output mixed weights. Our method uses free handles (FH: the spheres of the sphere-mesh), on the contrary to [Le and Deng 2014] which outputs rigid bones.

	OURS	[Le and Deng 2012]	[Le and Deng 2014]
INPUT PARAM.	GEO	NOB	NOB
RAPIDITY	++	+	–
MIXED WEIGHTS	Yes	No	No
HANDLES	FH	FH	BONES

7. DISCUSSION

Skinning decomposition. Table II summarizes the differences with recent previous work.
(i) Our method is the first to take a geometry (i. e., the animated sphere-mesh) as input, whereas other methods require a desired number of bones. Hereby, a user can easily determine the parameters to obtain the expected precision and wanted level of detail. This is critical, especially if the skinning decomposition method requires several minutes to compute e. g., Le and Deng [2014] report an execution time of 384 minutes for the decomposition of the *Samba* sequence on a commodity laptop similar to ours.
(ii) The comparably high performance of our method results from the use of a coarse geometry (our animated sphere-mesh), which is optimized to respect the geometric details that are present in the input animation. It is even several orders of magnitude faster than the method of Le and Deng [2014] (all our decompositions were performed in less than three minutes, even for the above-mentioned *Samba* sequence — a speedup of two orders of magnitude).
(iii) Our method is the only existing one to output mixed weights, which leads to important additional handles for editing applications, which is one of the main goals of previous work.

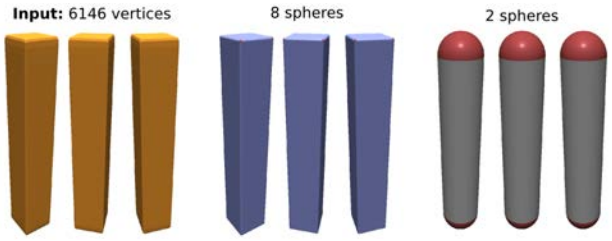


Fig. 12. The input capsule is undergoing rotation around its dominant axis. The motion is still visible when using a 8-sphere approximation, whereas 2 spheres are not sufficient to capture it.

(iv) On the downside, similarly to the method of Le and Deng [2012], our method outputs weights w.r.t. the spheres in our examples, which are free handles and are not constrained to respect any rigidity during the animation; this differs from the explicit edges of Le and Deng [Le and Deng 2014], which have fixed length over the animation and their extremities connected by joints. However, not any animation can be approximated by a rigid skeleton (e. g., sequences with large stretching).

Finally, just like other skinning decomposition techniques, relying on the input connectivity for the geometric construction of the skeletal domain can lead to poor animation reproduction. Currently, an input sequence containing a lot of disconnected parts leads to a disconnected animated sphere-mesh.

Memory complexity. Presently, the input animation has to reside in memory, preventing us from approximating very large animations. We thus cannot immediately generalize our approach to a streaming scenario where a sphere-mesh would be refined on demand. A possible solution could be to rely only on an analysis of the most significant (e. g., least redundant) poses to compute the animated sphere-mesh. This one could then be fit to the other poses.

Surface extraction. As in [Thiery et al. 2013], a surface reconstruction based on the animated sphere-mesh can be seen as a Minkowski sum of an oriented mesh and a sphere with spatially-varying radius (some triangles may be double sided though, and wire edges describe complex orientations, as the degeneracy of a cylinder over themselves). Yet, we extract the outer surface of the complete interpolation of the spheres over the sphere-mesh, which is equivalent to the Minkowski sum only when all triangles are double-sided and there is no boundary in the sphere-mesh.

Noise. Filtering topological and geometric noise from the input sequence prior to our approximation scheme, is clearly a promising direction for future research. While the former may involve highly non-trivial non-homotopic mappings, the latter could be adapted from existing filtering techniques for static meshes.

Abstraction limitations. Finally, extreme simplifications of the input mesh can fail to depict the motion of the input sequence. Fig. 12 illustrates this point: when an 8-sphere approximation of the capsule is used, the motion is still visible, whereas it becomes invisible when only 2 spheres are used. As future work, one could think of attaching a local frame to each sphere, and encode an *interpolating function* on each edge and triangle so as to encode a local coordinate system to the animated sphere-mesh. This could be a step towards parameterizing the animated sphere-mesh, e. g., for texturing.

Conclusion

We proposed a shape approximation algorithm efficiently converting an animated mesh sequence into an *animated sphere-mesh*, which is a mesh indexing a set of animated spheres. To do so, we introduced a new optimization scheme tailoring the animated spheres robustly to capture the animated shape at a given level of detail. Additionally, we showed how connectivity and temporal coherence can be optimized. The resulting animated sphere-mesh models the animated mesh sequence from a fine resolution surface representation to a coarse volumetric one, based on a single user-defined scale value, which still captures the dominant motions and geometric entities in the raw data, even at the coarsest levels. In contrast to skeleton-based or cage-based performance-capture reverse-engineering systems, our alternative can locally model tubular structures and provide a convincing volumetric approximation for all other components. We demonstrated its effectiveness on a collection of non-trivial examples and compared it to purely surface-based approximation methods.

Based on the resulting animated sphere-mesh, we showed how to rig a single mesh of the original sequence with it, reproducing faithfully the full animated sequence. The underlying linear-blend skinning map is smooth and accounts for both the animation and the geometry of the original sequence. Hereby, regions with poor motion, but salient structures, as well as simple geometry with singular motions are rendered editable.

Our work is the first to output an animated volumetric structure to approximate animated 3D surfaces. Volumetric structures of static geometry have already many applications. Hand-designed animated generalized cylinders are used for tasks such as collision detection in modern games [Sambavaram 2007]. We showed that our approach leads to high-quality skinning decompositions, and offers flexibility to the artists in this context. Our work has potential to serve as an enabler for future work, as advanced computer graphics frameworks may build upon our representation to address new challenges, and we believe it is a step toward a unified framework for volumetric shape and motion modeling and analysis.

APPENDIX

A. QUADRIC MINIMIZATION (NOT INVERTIBLE):

Minimize $E(\bar{\mathbf{s}}) = \frac{1}{2}\bar{\mathbf{s}}^{\mathrm{T}} \cdot \bar{A} \cdot \bar{\mathbf{s}} - \bar{b}^{\mathrm{T}} \cdot \bar{\mathbf{s}}$, with $\bar{A}$, $\bar{b}$ of the form given by Eq. 10, subject to $0 \leq \lambda \leq 1$ and $0 \leq r \leq R$.

The global minimizer $\bar{\mathbf{s}}^*$ (without inequality constraints) is given by $\bar{A} \cdot \bar{\mathbf{s}}^* = \bar{b}$, leading to:

$$\begin{cases} \mu\lambda + \nu r = \beta_1 & (E1) \\ M_f \cdot q_f + r N_f = b_f \ \ \forall f > D & (E2_f) \\ \nu\lambda + \sum\limits_{f>D} {N_f}^{\mathrm{T}} \cdot q_f + W r = \beta_2 & (E3)\ , \end{cases}$$

$$\Leftrightarrow \begin{cases} \mu\lambda + \nu r = \beta_1 & (E1 := E1) \\ \nu\lambda + W_2 r = \beta_3 & (E2 := E3 - \sum\limits_{f>D} {N_f}^{\mathrm{T}} {M_f}^{-1} E2_f) \\ q_f = M_f^{-1} \cdot (b_f - r N_f) \ \ \forall f > D & (E3_f := M_f^{-1} \cdot (E2_f - r N_f)) \end{cases}$$

with $W_2 := W - \sum\limits_{f>D} {N_f}^{\mathrm{T}} M_f^{-1} N_f$ and $\beta_3 := \beta_2 - \sum\limits_{f>D} {N_f}^{\mathrm{T}} M_f^{-1} b_f$.

The solution without inequality constraints is therefore given by

$$\begin{cases} \begin{pmatrix} \lambda^* \\ r^* \end{pmatrix} = \operatorname{argmin} \left| \begin{bmatrix} \mu & \nu \\ \nu & W_2 \end{bmatrix} \cdot \begin{pmatrix} \lambda \\ r \end{pmatrix} - \begin{pmatrix} \beta_1 \\ \beta_3 \end{pmatrix} \right|^2 & \\ q_f^* = u_f + \lambda^* \vec{d}_f & \forall f \leq D \\ q_f^* = M_f^{-1} \cdot (b_f - r^* N_f) & \forall f > D \end{cases}$$

We note $C := \begin{bmatrix} \mu & \nu \\ \nu & W_2 \end{bmatrix}$ and $\mathcal{D}_R := [0,1] \times [0,R]$. Note that $\begin{pmatrix} \beta_1 \\ \beta_3 \end{pmatrix} \in Im(C)$, otherwise the gradient of the quadric could never be null. Noting $C^\dagger$ the pseudo-inverse of C, several cases need to be considered based on the dimension of C's kernel $\dim(Ker(C))$:

(1) If $\dim(Ker(C)) = 0$ ($\Leftrightarrow Ker(C) = \emptyset$):
—If $\begin{pmatrix} \lambda^* \\ r^* \end{pmatrix} = C^{-1} \cdot \begin{pmatrix} \beta_1 \\ \beta_3 \end{pmatrix} \in \mathcal{D}_R$:
$$\begin{cases} \lambda = \lambda^* \quad r = r^* \\ q_f = u_f + \lambda \vec{d_f} & \forall f \leq D \\ q_f = M_f^{-1} \cdot (b_f - r N_f) & \forall f > D \end{cases}$$
—otherwise: optimize on $\partial\mathcal{D}_R$ (see next paragraph)

(2) if $\dim(Ker(C)) = 1$:
—If $\{C^\dagger \cdot \begin{pmatrix} \beta_1 \\ \beta_3 \end{pmatrix} + Ker(C)\} \cap \mathcal{D}_R \neq \emptyset$, then choose $\begin{pmatrix} \lambda \\ r \end{pmatrix}$ in $C^\dagger \cdot \begin{pmatrix} \beta_1 \\ \beta_3 \end{pmatrix} + Ker(C)$ with smallest radius, and
$$\begin{cases} q_f = u_f + \lambda \vec{d_f} & \forall f \leq D \\ q_f = M_f^{-1} \cdot (b_f - r N_f) & \forall f > D \end{cases}$$
—otherwise: optimize on $\partial\mathcal{D}_R$ (see next paragraph)

(3) otherwise: the space of solutions of $\overrightarrow{\nabla}_{\vec{s}} E = \vec{0}$ is a two dimensional space, and we choose the solution
$$\begin{cases} \lambda = 1/2 \quad r = 0 \\ q_f = u_f + 1/2 \vec{d_f} & \forall f \leq D \\ q_f = M_f^{-1} \cdot b_f & \forall f > D \end{cases}$$

Optimization on $\partial\mathcal{D}_R$. $\partial\mathcal{D}_R$ is composed of four segments: $\partial\mathcal{D}_R = \{0,1\} \times [0,R] \cup [0,1] \times \{0,R\}$. The minimizer on $\partial\mathcal{D}_R$ is then the solution found with minimal cost over these segments.

Fixing λ to $\hat{\lambda} \in \{0,1\}$: The minimizer on $\{\hat{\lambda}\} \times \mathbb{R}$ is given by
$$\begin{cases} r^* = \frac{\beta_4 - \sum_{f>D} {N_f}^{\mathrm{T}} \cdot M_f^{-1} \cdot b_f}{W - \sum_{f>D} {N_f}^{\mathrm{T}} \cdot M_f^{-1} \cdot N_f} \\ q_f^* = M_f^{-1} \cdot (b_f - r^* N_f) & \forall f > D \end{cases}$$
with $\beta_4 := \beta_2 - \hat{\lambda}\nu$.

If the denominator of r^* is null, we ignore this step and fix r as well to 0 and R and keep the solution with minimal cost. Otherwise:

—If $0 \leq r^* \leq R$: the minimizer on $\{\hat{\lambda}\} \times [0,R]$ is given by
$$\begin{cases} r = r^* \\ q_f = u_f + \hat{\lambda} \vec{d_f} & \forall f \leq D\,. \\ q_f = M_f^{-1} \cdot (b_f - r N_f) & \forall f > D \end{cases}$$
—Otherwise we fix r as well to 0 or R and keep the solution with minimal cost.

Fixing r to $\hat{r} \in \{0,R\}$: The minimizer on $\mathbb{R} \times \{\hat{r}\}$ is given by
$$\begin{cases} \lambda^* = (\beta_1 - \nu \hat{r})/\mu \\ q_f^* = M_f^{-1} \cdot (b_f - \hat{r} N_f) & \forall f > D \end{cases}$$
If the denominator μ is null, we ignore this step and fix λ as well to 0 and 1 and keep the solution with minimal cost. Otherwise:

—If $0 \leq \lambda^* \leq 1$: the minimizer on $[0,1] \times \{\hat{r}\}$ is given by
$$\begin{cases} r = \hat{r} \\ q_f = u_f + \lambda^* \vec{d_f} & \forall f \leq D\,. \\ q_f = M_f^{-1} \cdot (b_f - r N_f) & \forall f > D \end{cases}$$

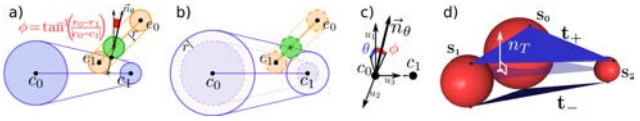


Fig. 13. Intersecting a capsule (blue) and a sphere-segment (orange) (**a**) is equivalent to intersecting a thicker capsule and a segment (**b**). **c**): Cone's normal parameterization. **d**): Interpolation of three spheres on a (transparent blue) triangle.

—Otherwise we fix λ as well to 0 or 1 and keep the solution with minimal cost.

Fixing (λ, r) to $(\hat{\lambda}, \hat{r}) \in \{0,1\} \times \{0,R\}$: The minimizer of the energy when fixing r and λ is simply given by
$$\begin{cases} r = \hat{r} \\ q_f = u_f + \hat{\lambda} \vec{d_f} & \forall f \leq D \\ q_f = M_f^{-1} \cdot (b_f - r N_f) & \forall f > D \end{cases}$$

B. INTERSECTION WITH SPHERE-SEGMENTS

We aim at detecting the intersection between a sphere-segment $[(e_0;r),(e_1;r)]$ (a sphere with radius r travelling continuously from point e_0 to point e_1) and a sphere-mesh. This can be achieved by testing the intersections between the segment and the various geometric primitives of the sphere-mesh, and selecting the closest valid one. In the following, we describe the cases for intersecting the segment with spheres, capsules, and thick triangles. We note $e_\lambda := e_0 + \lambda \overrightarrow{e_0 e_1}$ a parameterization of (e_0, e_1).

Against spheres: There exists an intersection with a sphere $(C;R)$ if the following can be satisfied:
$$\exists \lambda \in [0,1] \quad | \quad ||e_\lambda - C||^2 = (R+r)^2, \tag{14}$$
which boils down to solving a simple quadratic polynomial in λ.

Against capsules: Detecting the intersection between the segment and a capsule defined as the interpolation of two spheres $(c_0;r_0)$ and $(c_1;r_1)$ (with $r_0 \geq r_1$, is equivalent to detecting the intersection between the segment $[e_0,e_1]$ and the capsule thickened by r (see Fig.13a,b). We will therefore focus on the second problem. We note $c_\mu := c_0 + \mu \overrightarrow{c_0 c_1}$ a parameterization of the line (c_0,c_1), and $\vec{n}_\theta$ one of the cone's normal ($\vec{n}_\theta := \cos(\theta)\cos(\phi)u_1 + \sin(\theta)\cos(\phi)u_2 + \sin(\theta)u_3$, see Fig.13b).

There exists an intersection if the following can be satisfied:
$$\exists (\lambda, \mu, \theta) \quad | \quad e_\lambda = c_\mu + (r_0 + r + \mu(r_1 - r_0))\vec{n}_\theta. \tag{15}$$

Computing $(\text{Eq.15})^{\mathrm{T}} u_3$ and $((\text{Eq.15})^{\mathrm{T}} u_1)^2 + ((\text{Eq.15})^{\mathrm{T}} u_2)^2$ gives

$$\lambda \overrightarrow{e_0 e_1}^{\mathrm{T}} \cdot u_3 - \mu(\overrightarrow{c_0 c_1}^{\mathrm{T}} \cdot u_3 + (r_1 - r_0)\sin(\phi)) = \overrightarrow{e_0 c_0}^{\mathrm{T}} \cdot u_3 + r_0 \sin(\phi)$$
$$(\overrightarrow{c_0 e_0}^{\mathrm{T}} \cdot u_1 + \lambda \overrightarrow{e_0 e_1}^{\mathrm{T}} \cdot u_1)^2 + (\overrightarrow{c_0 e_0}^{\mathrm{T}} \cdot u_2 + \lambda \overrightarrow{e_0 e_1}^{\mathrm{T}} \cdot u_2)^2 = (r_0 + r + \mu(r_1 - r_0))^2 \cos(\phi)^2$$

Combining these last equations leads to a simple second order polynomial in (λ, μ). Solving this quadric provides the solution, assuming it respects: $0 \leq \lambda, \mu \leq 1$.

Against thick triangles: The intersection between the segment and a thick triangle defined as the interpolation of the three spheres $(c_0;r_0)$, $(c_1;r_1)$ and $(c_2;r_2)$ with normal n_T, can be found by checking the intersections against the capsules (edges of the thick

triangle) and two additional triangles $t_+ = (p_0^+, p_1^+, p_2^+)$ and $t_- = (p_0^-, p_1^-, p_2^-)$ (see Fig.13c), which are constructed as follows:

(1) compute $\phi_{ij} = \tan^{-1}((r_i - r_j)/||c_j - c_i||),\ \forall 0 \leq i < j \leq 2$
(2) compute p_i as the intersection of the three planes:
—$\{\overrightarrow{c_i p_i}^{\mathrm{T}} \cdot n_T = 0\}$,
—$\{\overrightarrow{c_i p_i}^{\mathrm{T}} \cdot \overrightarrow{c_i c_j} = r_i ||\overrightarrow{c_i c_j}|| \sin(\phi_{ij})\}\ \forall j \neq i, 0 \leq j \leq 2$
(3) compute $p_i^{+/-}$ as $p_i^{+/-} = p_i +/- \sqrt{r_i^2 - ||\overrightarrow{c_i p_i}||^2} n_T$

Note that t_+ and t_- do not always exist, e. g., if the three sphere centers are coplanar.

ACKNOWLEDGMENTS

The captured performance data were provided courtesy of the research group 3D Video and Vision-based Graphics of the Max-Planck-Center for Visual Computing and Communication (MPI Informatik / Stanford).

REFERENCES

BARAN, I. AND POPOVIĆ, J. 2007. Automatic rigging and animation of 3d characters. In *ACM Transactions on Graphics (TOG)*. Vol. 26. ACM, 72.

BLOOMENTHAL, J. AND SHOEMAKE, K. 1991. Convolution surfaces. In *ACM SIGGRAPH Computer Graphics*. Vol. 25. ACM, 251–256.

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

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

DE AGUIAR, E., STOLL, C., THEOBALT, C., AHMED, N., AND SEIDEL, H.-P. 2008. Performance capture from sparse multi-view video. *27*.

DE AGUIAR, E., THEOBALT, C., THRUN, S., AND SEIDEL, H.-P. 2008. Automatic conversion of mesh animations into skeleton-based animations. *27*.

DE TOLEDO, R. AND LÉVY, B. 2008. Visualization of industrial structures with implicit gpu primitives. In *ISVC, International Symposium on Visual Computing*.

DECORO, C. AND RUSINKIEWICZ, S. 2005. Pose-independent simplification of articulated meshes. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*. ACM, 17–24.

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

GÄRTNER, B. AND HERRMANN, T. 2001. Computing the width of a point set in 3-space. *Abstracts for the Thirteenth Canadian Conference on Computational Geometry*, 101–103.

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

HOULE, J. AND POULIN, P. 2001. Simplification and real-time smooth transitions of articulated meshes. In *Graphics Interface 2001*. 55–60.

HUANG, F.-C., CHEN, B.-Y., CHUANG, Y.-Y., AND OUHYOUNG, M. 2005. Animation model simplifications. In *Computer Graphics Workshop*.

JACOBSON, A., BARAN, I., POPOVIC, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph. 30,* 4, 78.

JAMES, D. L. AND TWIGG, C. D. 2005. Skinning mesh animations. In *ACM Transactions on Graphics (TOG)*. Vol. 24. ACM, 399–407.

KAVAN, L., SLOAN, P.-P., AND O'SULLIVAN, C. 2010. Fast and efficient skinning of animated meshes. In *Computer Graphics Forum*. Vol. 29. Wiley Online Library, 327–336.

KIRCHER, S. AND GARLAND, M. 2005. Progressive multiresolution meshes for deforming surfaces. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM, 191–200.

LANDRENEAU, E. AND SCHAEFER, S. 2009. Simplification of articulated meshes. In *Computer Graphics Forum*. Vol. 28. Wiley Online Library, 347–353.

LAWSON, C. L. AND HANSON, R. J. 1974. *Solving least squares problems*. Vol. 161. SIAM.

LE, B. H. AND DENG, Z. 2012. Smooth skinning decomposition with rigid bones. *ACM Transactions on Graphics (TOG) 31,* 6, 199.

LE, B. H. AND DENG, Z. 2014. Robust and accurate skeletal rigging from mesh sequences. *ACM Transactions on Graphics (TOG) 33,* 4, 84.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 165–172.

MAGNENAT-THALMANN, N., LAPERRIRE, R., THALMANN, D., ET AL. 1988. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics interface*. Citeseer.

MCCORMACK, J. AND SHERSTYUK, A. 1998. Creating and rendering convolution surfaces. In *Computer Graphics Forum*. Vol. 17. Wiley Online Library, 113–120.

MOHR, A. AND GLEICHER, M. 2003. Deformation sensitive decimation. *Technical Report*.

MURAKI, S. 1991. Volumetric shape description of range data using "blobby model". *SIGGRAPH Comput. Graph. 25,* 4 (July), 227–235.

PAYAN, F., HAHMANN, S., AND BONNEAU, G.-P. 2007. Deforming surface simplification based on dynamic geometry sampling. In *Shape Modeling and Applications, 2007. SMI'07. IEEE International Conference on*. IEEE, 71–80.

SAMBAVARAM, R. 2007. Cylinder collision. Insomniac Games Tech Team Presentation.

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

STOLPNER, S., KRY, P., AND SIDDIQI, K. 2012. Medial spheres for shape approximation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 34,* 6 (June), 1234–1240.

TALTON, J. O. 2004. A short survey of mesh simplification algorithms. *University of Illinois at Urbana-Champaign*.

THIERY, J.-M., GUY, É., AND BOUBEKEUR, T. 2013. Sphere-meshes: shape approximation using spherical quadric error metrics. *ACM Transactions on Graphics (TOG) 32,* 6, 178.

VLASIC, D., BARAN, I., MATUSIK, W., AND POPOVIĆ, J. 2008. Articulated mesh animation from multi-view silhouettes. *ACM Transactions on Graphics (TOG) 27,* 3, 97.

WANG, R., ZHOU, K., SNYDER, J., LIU, X., BAO, H., PENG, Q., AND GUO, B. 2006. Variational sphere set approximation for solid objects. *The Visual Computer 22,* 9, 612–621.

ZANNI, C., BERNHARDT, A., QUIBLIER, M., AND CANI, M.-P. 2013. Scale-invariant integral surfaces. In *Computer Graphics Forum*. Vol. 32. Wiley Online Library, 219–232.

ZHANG, S., ZHAO, J., AND WANG, B. 2010. A local feature based simplification method for animated mesh sequence. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*. Vol. 1. IEEE, V1–681.

Fast decompression for web-based view-dependent 3D rendering

Federico Ponchio*
Visual Computing Lab, ISTI-CNR, Pisa

Matteo Dellepiane†
Visual Computing Lab, ISTI-CNR, Pisa

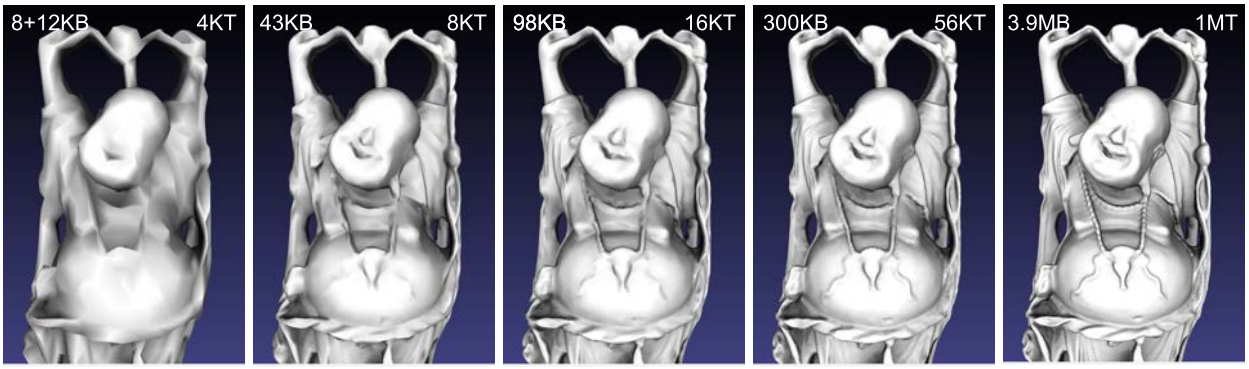


Figure 1: *Progressive refinement of the Happy Buddha: on the upper left corner the size downloaded, on the upper right corner the number of triangles in the refined model. The header and index amount to 8KB*

Abstract

Efficient transmission of 3D data to Web clients and mobile applications remains a challenge due to limited bandwidth. Most of the research focus in the context of mesh compression has been on improving compression ratio. However, in this context the use of Javascript on the Web and low power CPUS in mobile applications led to critical computational costs. Progressive decoding improves the user experience by providing a simplified version of the model that refines with time, and it's able to mask latency. Current approaches do so at very poor compression rates or at additional computational cost. The need for better performing algorithms is especially evident with this class of methods where Limper [Limper et al. 2013b] demonstrated how decoding time becomes a limiting factor even at moderately low bandwidths. In this paper we present a novel multi-resolution WebGL based rendering algorithm which combines progressive loading, view-dependent resolution and mesh compression, providing high frame rates and a decoding speed of million of triangles per second in Javascript. This method is parallelizable, robust to non-manifold meshes, and scalable to very large models.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Visible line/surface algorithms;

Keywords: Multi-resolution, web visualization, 3D compression

1 Introduction

Limited bandwidth and increasing model sizes pose a challenge in the transmission of 3D data to Web clients and mobile applications. Mesh compression is a viable approach to minimize transmission time, and most research focus in this field has been on optimizing compression ratio.

Unfortunately, limited bandwidth often pairs with limited computational power, either because of Javascript environment or low CPU power mobile devices, to the point that for most algorithms decoding time becomes the bottleneck even at moderately low bandwidth. Acceptable rates can be regained reducing compression ratio (for example forfeiting connectivity compression) or using less sophisticate entropy compression algorithms.

A different approach makes use of progressive reconstruction algorithms, which improve the user experience by providing a simplified version of the model that refines while the remaining part of the model is being downloaded. The model converges very quickly at the beginning of the download, and only the details require the full model. However this class of algorithms performs even worse in terms of decoding time (as shown in Limper [Limper et al. 2013b]) or in terms of compression ratio.

Another desirable feature, especially for very large models, is view-dependent resolution: this allows to prioritize the download, decode

*e-mail:federico.ponchio@isti.cnr.it
†e-mail:matteo.dellepiane@isti.cnr.it

a specific part of the model and vary resolution of the rendered geometry to maintain a constant screen resolution. This is obtained by maximizing quality at a given frame rate.

In this paper we present a novel multi-resolution WebGL based rendering algorithm which combines progressive loading, view-dependent resolution and a mesh compression providing good rates and a decoding speed of million of triangles per second in Javascript. This method is can handle non-manifold meshes, and it is also scalable to deal with very large models.

The method is based on a class of multiresolution structures [Cignoni et al. 2004; Cignoni et al. 2005] where the "primitive" of the multiresolution becamesis a patch made of thousands of triangles. The processing required to traverse this structure becomes a fraction of triangle based multiresolution algorithms, and allows "batch" operation on the patches: moving data from disk or network to GPU RAM, rendering, and decompression.

In section 3 we describe the improvement made on the multiresolution structure and the how the compression algorithm was designed to optimize decoding time while maintaining a good compression ratio. In section 4 we compare it with existing web solutions for mesh compression and progressive visualization. It represents a solid alternative to current methods, providing a practical mean to handle 3D models on the web.

2 Related Work

This paper is related to several topics in the field of Computer Graphics. Among them, the main are: web-based 3D rendering, progressive and multiresolution rendering approaches, and fast decompression methods for 3D models.
While a complete overview of all these subjects goes well beyond the scope of the paper, in the next subsections we provide a short description of the state of the art, trying to focus on the aspects which are more related to the proposed approach.

2.1 Web-based 3D rendering

Three-dimensional content has always been considered as part of the multimedia family. Nevertheless, especially when talking about web visualization, its role with respect to images and videos has always been a minor one. Visualization of 3D components was initially devoted to external components, such as Java applets or ActiveX controls [Mic 2013].

After some initial efforts for standardization [Raggett 1995; Don Brutzmann 2007], the proposal of WebGL standard [Khronos Group 2009b], which is a mapping of OpenGL|ES 2.0 [Khronos Group 2009a] specifications in JavaScript, brought a major change. Several actions related to the use of advanced 3D graphics has been proposed since then. For a general survey, please refer to the survey by Evans [Evans et al. 2014b]. Since the use of OpenGL commands needs advanced programming skills, there have been several actions to provide an "interface" between them and the creation of web pages. We could subdivide the proposed systems between *declarative* approaches [Jankowski et al. 2013], like X3DOM [Behr et al. 2009] or XML3D [Sons et al. 2010], and *imperative* approaches, like Three.js [Dirksen 2013], SpiderGL [Di Benedetto et al. 2010] and WebGLU [DeLillo 2009]. The main difference between the groups is that the first ones rely on the concept of *scenegraph*, hence a scene has to be defined in all its elements, while the second ones provide a more direct interface with the basic commands. Other systems provide a sort of hybrid approach, where a very simplified scene has to be defined.
Evans [Evans et al. 2014b] points out in his survey that declarative approaches had a major impact in the research community, while imperative approaches were mainly used in the programming community.
More in general, given the fact that the amount of data that needs to be sent to the webpage can be quite big, several efforts about a better organization of generic streamable formats [Limper et al. 2014a; Sutter et al. 2014] has been proposed. Nevertheless, when complex 3D data have to be streamed, these structures are not flexible enough to handle them.
In order to face this problem, in the last three years some progressive compression methods ad hoc for 3D streaming have been developed. Gobbetti et al. [Gobbetti et al. 2012] proposed a quad-based multi-resolution format. Behr et al. [Limper et al. 2013a] transmit different quantization levels of the geometry using a set of nested GPU-friendly buffers. Lavouè et al. [Lavoué et al. 2013] proposed an adaptation for the Web (reduced decompression time at the cost of a low compression ratio) of a previous progressive algorithm [Lee et al. 2012]. Other research has been also conducted to handle other types of data, like point clouds [Evans et al. 2014a], which may present different types of issues to face with. Please refer to next subsections for a more in-depth analysis.

2.2 Progressive and Multi-resolution methods

An important feature for user experience when rendering over slow connections or compressed models is progressiveness: the possibility to temporarily display an approximated version of the model and to refine it while downloading or processing the rest of the data.

The simplest (and widely used) strategy is to use a a discrete set of increasing resolution models (usually known as Level Of Detail, LOD). The main drawback with this approach is the abrupt change in detail each time a model is replaced.

A change of paradigm was brought by progressive meshes, introduced by Hoppe [Hoppe 1996]. These meshes encode the sequence of operations of a edge collapse simplification algorithm. This sequence is traversed in reverse, so that each collapse becomes a split, and the mesh is refined until the original resolution. An advantage of progressive techniques is the much more smooth transition resolution changes, and the possibility to combine it with selective refining or view-dependent multiresolution, but this high granularity was achieved at the cost of low compression rates: about 37 bpv with 10 bit vertex quantization.

A large number of progressive techniques were later developed, but as noted in [Limper et al. 2013b], Table 1, the research focus, however, was on rate-distortion performances and speed was mostly neglected. Latest algorithms still run below 200KTs in CPU.

Mobile and web application would be really too slow using these methods. As a compromise, pop buffers [Limper et al. 2013a] propose a method to progressively transmit geometry and connectivity, while completely avoiding compression.

Another desirable feature, especially for large models, is view-dependent loading and visualization. Most multiresolution algorithms were made obsolete by the increased relative performances of GPU over CPU around the first years of 2000. It simply became inefficient to operate on the mesh at the level of the single triangle. Several works [Yoon et al. 2004; Sander and Mitchell 2005; Cignoni et al. 2004; Cignoni et al. 2003] achieved much better performances by increasing the granularity of the multiresolution to a few thousand triangles.

The main problem when increasing the granularity is ensuring boundary consistency between patches at different resolution: Yoon

[Yoon et al. 2004] and Sander [Sander and Mitchell 2005] both employ a hierarchical spatial subdivision, but while the first simply disallow simplification of most boundary edges, which results in scalability problems, the second relies relies on global, spatial GPU geomorphing to ensure that progressive meshes patch simplification is consistent between adjacent blocks. The works by Cignoni [Cignoni et al. 2003; Cignoni et al. 2004] rely instead on a non hierarchical volumetric subdivision and a boundary preserving patch simplification strategy that guarantees coherence between different resolutions while at the same time ensures no boundary persists for more than one level. While not progressive in a strict sense, given current rendering speed, the density of triangles on screen is so high that popping effects are not noticeable.

Compression comes as a natural extension to this family of multiresolution algorithms: each patch can be compressed independently from the others as long as the boundary still matches with neighboring patches. a wavelet based compression was developed in [Gobbetti et al. 2006] for terrains, a 1D Haar wavelet version in [Rodríguez et al. 2013] for generic meshes on a mobile application. A comprehensive account of compression algorithms and the convergence with view-dependent rendering of large datasets can be found on a recent survey from Maglo et al.[Maglo et al. 2015].

2.3 Fast Decompression of 3D models

Given that decompression speed is a key factor in order to be able to use compressed mesh, there's been some effort by the community to provide solutions.
Gumhold and Straßer [Gumhold and Straßer 1998] developed a connectivity only compression algorithm that was able to decompress at 800KTs in 1998. Pajarola and Rossignac in [Pajarola and Rossignac 2000], in 2000, reported 26KTs for a progressive compression algorithm, and developed a high-performance Huffman decoding identifying entropy compression as a possible bottleneck. Finally, Isenburg and Gumhold in 2003 developed a streaming approach to compression of gigantic meshes reaching an impressive decompression speed of 2MTs.

3 Method

Our multiresolution algorithm builds upon the methods described on [Cignoni et al. 2004; Cignoni et al. 2005], which is recapped in section 3.1 for completeness. In our solution we adopt a improved partition strategy (see section 3.2), and, more importantly, a novel compression scheme (section 3.3) tailored around the need for decompression speed.

3.1 Batched Multiresolution

The model is split into a set of small meshes at different resolutions that can be assembled to create a seamless mesh simply traversing a tree which encodes the dependencies between each patch, using the estimated screen error to select the resolution needed in each part of the model. To build this collection of patches we need a sequence of non-hierarchical volume partitions (V-partition) of the the model; non hierarchical means essentially that no boundary is preserved between partitions at different levels of the hierarchy.

The data structure is composed of a fixed size header describing the attributes of the models, an small index which contains the tree structure of the patches and the position of each patch in the file, and the patches themselves. We use HTTP Range requests to download header and index, ArrayBuffers to parse this structures into Javascript; the patches are then download prioritizing highest screen error. Figure 2 shows an example of a model before and after view-dependent refinement.

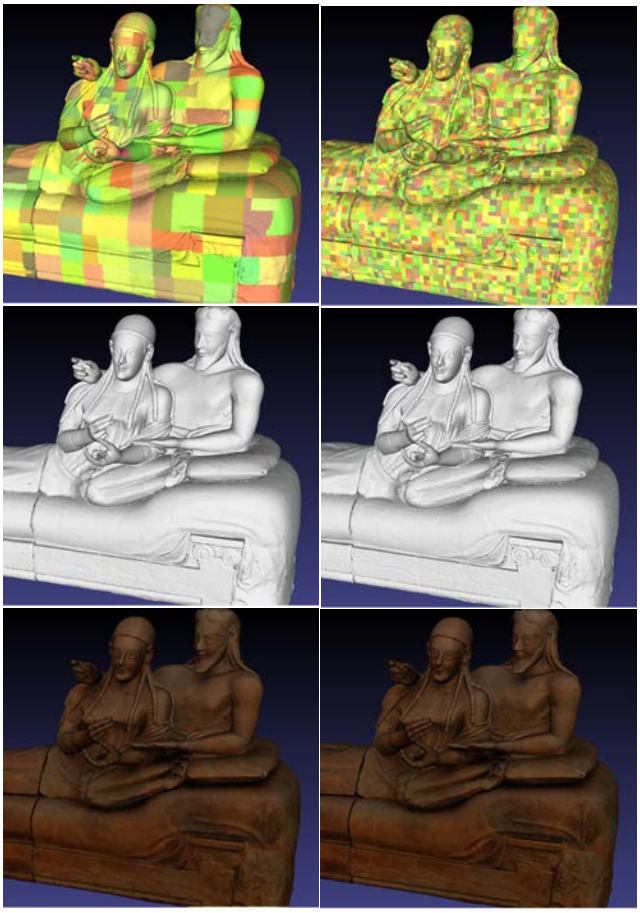


Figure 2: *First column: before refinement. Second column: after refinement. From top to bottom: a visual representation of the geometric patches representing the model, the model with pure geometry, the model with color information.*

The rendering requires the traversal of the patch tree, which is usually quite small since each patch is in the range of 16-32K vertices, computing the approximated screen space error in pixel from the bounding sphere and the quadric error (or any other error metric) during simplification. The traversal is stopped whenever our triangle budget is reached, the error target is met or the required patches are still not available.

Since the rendering can start when the first patch is downloaded and the model is refined as soon as some patch is available, this is effectively a progressive visualization albeit with higher granularity. On the other hand, this structure is view dependent and thus able to cope with very large models, on the order of hundreds of millions of triangles.

3.2 Partition

Cignoni et al [Cignoni et al. 2005] showed that any non-hierarchical sequence of volume partitions can be the base of a patch based multiresolution structure. Good partition strategy minimize boundaries thus generating compact cells. In addition, it allows streaming construction and generates well balanced trees even when the distribution of the model triangles is very irregular. The Voronoi structure,

while optimal for boundary minimization and balance, is not suitable for streaming leading to long processing times. On the other hand the regular spatial subdivision used in [Cignoni et al. 2004] might generate unbalanced trees for very irregular models. This may impact on adaptivity.

In our solution each volume partition is defined by the leaves of a KD-tree built on the triangles of the model; to ensure the non hierarchical condition, the split ratio in the KD-tree alternates between 0.4 and 0.6 instead of the usual 0.5. This choice allows for streaming processing of the model and good adaptivity. As a bonus, the very regular shape of the patches (see figure 2) may be useful when adding texture support.

3.3 Mesh Compression

Our multiresolution algorithm imposes a set of constrains to mesh compression:

- each patch needs to be encoded independently from the other, so the method must be efficient and fast even on small meshes
- boundary vertices, replicated on neighboring patches, need to remain consistent through compression
- non manifold models must be supported

It would be possible to exploit the redundancy of the data due to the fact that the same surface is present in patches at different levels of resolution. We choose not to do so in order to keep the compression stage independent of the simplification algorithm used and to simplify parallel decompression of the patches (we would have to keep track of and enforce dependencies otherwise).

3.3.1 Connectivity compression

We modified the algorithm presented in [Floriani et al. 1998], to support non manifold meshes and surfaces with handles or holes.

We need face-face topology for compression and this is computed as follows: we create an array containing three edges for each triangle, and sort it so that edges sharing the same vertices will be consecutive (independently of the order of the edges). The edges are then paired taking orientation into account, and all non paired edges are marked as boundary. Non manifold meshes will simply force the creation of some artificial boundaries.

The encoding process starts with a triangle and expands iteratively adding triangles. The processed region is always homeomorphic to a disk and if the region meets already considered triangles, we consider the common vertices as duplicated. The boundary of the already processed (encoded or decoded) region is stored as a doubly linked list of oriented edges (*active edges*), The list is actually implemented as an array for performances reasons. A queue keeps track and prioritize the *active edges*.

The first triangle adds three active edges to the list; iteratively an edge is extracted from the queue and, if not marked as processed, the following codes are emitted (see Figure 3):

SKIP if the edge is a boundary edge, or the adjacent triangle has already been encoded; the edge is marked as processed.

LEFT or **RIGHT** if the adjacent triangle shares two edges with the boundary; The two edges are marked as processed, a new edge added to the queue and its boundary adjacencies adjusted.

VERTEX if the adjacent triangle shares only one edge with the boundary, in this case the edges is marked as processed and two new edges added to the queue. If vertex of the new triangle opposing the edge was never encountered before its position is estimated using parallelogram prediction and the difference encoded, otherwise its index is encoded (in literature this case is often referred as a "split"). This is a key difference with [Floriani et al. 1998], where in the second case a SKIP code would be emitted, to keep the encoded region simple.

If the mesh is composed of several connected components, the process is restarted for each component.

The order in which the active edges are processed is important as we would like to minimize the number of VERTEX split operations, and generate a vertex-cache-friendly triangle order. To do so, we simply prioritize the right edges in the VERTEX operation, so that the encoding proceeds in 'spirals'. If the mesh is not homeomorphic to a disk, some split operations are required. This strategy reduces the number of splits to less than 1% in our examples, incurring in an average of 0.2 bpv cost.

This algorithm is certainly not optimal in term of bitrate, but it is extremely simple, linear in the number of triangles and robust to non-manifold meshes; as we will see in the results, speed is more important than bitrate.

3.3.2 Geometry and vertex attribute compression

To ensure consistency between boundary vertices of adjacent patches, we adopt a global quantization grid for coordinates, normals and colors. The global grid step for vertex position quantization is chosen automatically based on the quadric errors during the simplification step in construction.

Geometry and vertex attributes are encoded as differences to a predicted value. The distribution of these values exhibit a bias which we can exploit to minimize the number of bits necessary to encode them. Our strategy is based on the assumption that most of the bias is concentrated on the position of highest bit (the log_2 of the value) of these value while the subsequent bits are mostly random. We simply store in an array, which is later entropy coded, the number of bits necessary to encode the value; the subsequent bits are stored in an uncompressed bitstream. In this way we need to decode a single symbol, from a limited alphabet, and read a few bits from a bitstream to decode a difference.

Each new vertex position, result of a VERTEX code, is estimated using a simple parallelogram predictor, and the differences with the actual position encoded as above. Color information is first converted into YCbCr color space and quantized, we encode the difference with one of the corner of the edge processed when emitting the VERTEX code. Normals vector are estimated using the decode mesh position and connectivity, and differences encoded as usual.

3.4 Entropy coding

We have shown how to convert connectivity, geometry and attributes into a stream of symbols and bits. It is worth compressing the symbol stream due to the biased probability distribution of the symbols.

Entropy decoding is the speed bottleneck in many mesh decompression methods, often due to the main goal of minimizing bit per vertex. Pajarola and Rossignac [Pajarola and Rossignac 2000] developed a high-performance Huffman decoding algorithm in order to overcome this problem. The main advantage of this method is that it reduces the decoding phase to a couple of table lookups. Arithmetic coding, for example, outperforms Huffman in term of compression rate, but exhibits lower speed. A problem with this approach is the initialization time required to create the, possibly very large, decoding tables. It is then not suitable for decoding

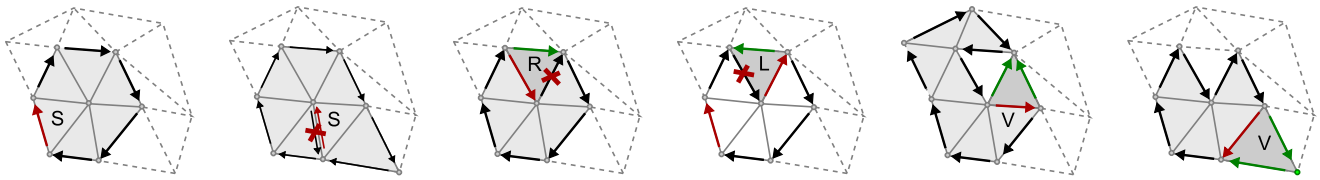


Figure 3: *The four decompression codes: black arrows represent the front, the red arrow the current edge, in green the new edges added to the front.*

small meshes where the construction time would dominate over the decoding time.

Unlike Huffman and other variable-length codes, Tunstall code [Tunstall 1967] maps a variable number of source symbols to a fixed number of bits. Since in decompression the input blocks consists of a fixed number of bits and the output is a variable number of symbols, Tunstall is slightly less efficient than Huffman, especially where the bit size of the input block is small. The decoding step is very similar to the high-performance Huffman algorithm, as it consists in a lookup table and a sequence of symbols for each entry, but the table size is only determined by the word size, and a fast method to generate it described in [Baer 2009].

Given an entropic source of M symbols, to generate an optimal encoding table for a word size of N bits, we need to generate 2^N symbol sequences that have a frequency as close as possible to 2^{-N}, allows to encode every possible input (it is complete) and no sequence is a prefix of any other sequence (it is proper).

Tunstall optimal strategy starts with the M symbols as initial sequences, removes the most frequent sequence A and replaces it with M sequences concatenating A with every symbol until we reach 2^N sequences. The algorithm most time consuming step is to find the most probable sequence.

If we use a matrix where the first column contains the sorted symbol in order of probability, and at each step we replace the sequence with highest probability with M sequences adding a new column, we can observe that this table is sorted both in columns and rows (see Figure 4). This allows to select the next sequence by keeping each row in a queue and using a priority queue to keep track of which queue has the highest front element.

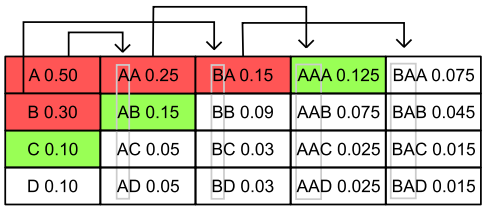


Figure 4: *First four steps in construction of a Tunstall code with four symbols, the sequences A, B, AA, BA are replaced with a new column, beside each sequence, its probability is shown. In green the candidates for the next expansion.*

To initialize the decoding table the symbol frequencies needs to be transmitted in advance.

Finally, an important advantage of variable-to-fixed coding is that the compressed stream is random accessible: decoding can start at any block. This makes it especially suited for parallel decompression in particular GPU decompression. Unfortunately, current limitations in the capabilities of WebGL do not allow for such an implementation.

4 Results

The C++ and Javascript implementation is freely available at http://vcg.isti.cnr.it/nexus under GPL licence.

Our implementation has been successfully tested on major browsers on a variety of platform, from desktop machines to low end cell phones. The results we report here were measured on an iCore5 3.1Gh, using Chrome 41. Timings taken other browsers (e.g.Firefox) where comparable.

The multiresolution model construction is a preprocessing operation, and the bottleneck is the quadric simplification algorithm that runs at about 60K triangles per second per core. Compression time is negligible at about 1M triangles per second.

4.1 Entropy Compression: Comparison

We tested, both in C++ and Javascript, compression rates and decompression speed of:

- our implementation of Tunstall coding (T)
- Huffman coding (H), in the high-performance version of Pajarola [Pajarola and Rossignac 2000] (our implementation, C++ only)
- available implementations of LZMA
  in C++: http://www.7-zip.org/sdk.html
  and Javascript: https://code.google.com/p/js-lzma/
- lz-string, a LZW based Javascript implementation
  http://pieroxy.net/blog/pages/lz-string/index.html

	C++			Javascript		
symbols	T	H	LZMA	T	LZMA	LZW
4	1058	520	1066	201	19	55
9	369	212	170	145	10	23
13	423	168	95	150	6	20
17	359	136	77	163	6	19
22	332	98	67	180	6	17

Table 1: *Decompression speed in million of output symbols per second for Poisson distribution of 32K sequences*

The results are presented in Table 1, the lenght of 32K has been chosen since it is typical in our application.

Huffman and Tunstall are very similar in term of decompression speed, the difference is mainly in the time required to generate the decoding tables which are much larger for Huffman, especially when increasing the number of symbols. We tested also other probability distributions and found little difference in terms of speed.

LZMA and LZW avoid this startup cost, however their more complex and adaptive dictionary management allows them to outperform Huffman and Tunstall in term of decompression speed only for very small runs (and very small dictionaries). In terms of compression ratio, Huffman and LZMA performed quite close to the theoretical minimum, while Tunstall was about 10% worse.

We did not implement Huffman in Javascript, as we are confident the result would be very similar. On the other hand the numbers for LZMA change dramatically. Lz-string serves as a comparison, as a better library, optimized for Javascript. The poor LZMA performances in Javascript help explain the relatively slow performances of CTM in Limper [Limper et al. 2013b].

4.2 Mesh Compression: Comparison

We used the Happy Budda model (in Figure 1), to compare compression ratio and decompression speed with OpenCTM (CTM) [Geelnard] Pop buffers (POP)[Limper et al. 2013b], P3DW [Lavoué et al. 2013], WebGL-loader (CHUN) [Chun 2012]. We compare our multiresolution (OUR) and, to test single resolution performances of our compression approach, a version (FLAT) which loads only the highest resolution level of the model. In each case the model has been quantized at 11 bit for coordinates and 8 bit for normals, and includes colors.

	FLAT	OUR	CTM	CHUN	POP	P3DW
MB	1.9	3.9	3.5	2.8	15	4.5
bpv	28	57	51	41	220	66
full	0.4	0.9	5.3	0.06	0.5	10

Table 2: *Statistics for the Happy Buddha: model size in megabytes, bit per vertex and time in seconds required to fully decompress the model.*

Our decompression Javascript implementation can decode about 1-3 million triangles per second with normals and colors in a single thread, on a desktop machine and 0.5 MT/s on a iPhone Five. Performances are somewhat degraded when the code is run during streaming visualization.

An important comparison is with [Rodríguez et al. 2013], which employs the same multiresolution batched strategy. For their mobile multiresolution application they reports compression rates of 45-50 bpv on large colored meshes (which should be compared to our 28bpv). The difference is probably mostly due to the different connectivity encoding which, in their case, requires 20bpv against our 4 or 5bpv. It is difficult to compare the speed of the two decompression approaches since they run natively in C# on an iPhone4 while we run in Javascript on the same platform. Our implementation speed is still, if a bit faster than their 50KTS [1], at about 60KTs. The difference is probably due their more sophisticate (and slow) arithmetic encoding.

C++ decompression speed is of course faster, reaching 9MTs, including colors and normals, and 16MTs for just position and connectivity. The speed reported in [Floriani et al. 1998] of 35KTs for just the connectivity, as they mention, is due to the dynamic memory allocation in their implementation.

4.3 Streaming and Rendering

Loading the geometry through the Range HTTP request requires an increased number of HTTP calls: one for each patch, or 30-60 calls every million of triangles. This does not really impact over performances: the overhead is quite small (about 400 bytes per call) and pipelining (the process of enqueueing requests and responses between browser and server) ensures full utilization of the available bandwidth. Random access is really necessary only to fully exploit the view-dependent characteristics of the multiresolution structure: the code could be easily modified to load the model with a single call if a higher number of HTTP calls was problematic on certain web hosting architectures.

In the demo page (http://web3d.duckdns.org) it is possible to compare the performances of our method w.r.t. existing solutions in the case of a slow connection. Moreover, very complex geometries are also available for further testing. As an example, in Figure 5 we show our system rendering the Portalada, a 180M triangles model at 30fps. The triangle budget has been fixed at 1M triangles and the streaming requires 2-3 seconds to reach full resolution on a good connection. The original model is 3.6GB, while the compressed multiresolution model is 838MB.

Figure 6 shows two examples where the method deals with non-optimal geometries. On the left side, a model exhibiting strong topological artifacts. On the right side, a model with very unbalanced data density. In both cases, the method is able to deal with the issues and provide an accurate and reliable rendering.

5 Conclusion

The method proposed in this paper provides good compression ratio, progressive visualization, fast decoding and view dependent rendering. It proves effective in a wide range of bandwidth availability, computing power and rendering capabilities. Moreover, it is able to handle models of arbitrary size. This means that also very complex geometries can be now explored in real time with average connections speeds.

Many mesh compression algorithms for mobile and web application do not employ topological connectivity compression often because it is believed to be excessively complex or slow and limited to manifold meshes. We prove that, if implemented correctly, this is not the case, and the choice of the entropy compression algorithm can play a much more important role.

5.1 Future improvements

An important limitation of the current implementation is the lack of texture support. Adding UV coordinates to the multiresolution structure and supporting them in compression is a trivial task, but dealing with simplification and providing multiresolution textures is much more difficult. We plan to tackle this problem in the near future, with an approach similar to Texture Mapping Progressive Meshes [Sander et al. 2001].

Point clouds are currently supported, with a z-index vertex compression strategy, we are working on improving the presentation. While the multiresolution structure is not really needed, using it has the advantage of working on an single framework.

Acknowledgements

The research leading to these results was funded by EU FP7 project ICT Harvest4D (http://www.harvest4d.org/ , GA n. 323567) and EU INFRA Project Ariadne (GA n. 313193, http://www.ariadne-infrastructure.eu/).

[1] The number is extrapolated from the decoding time of a large mesh given in their paper

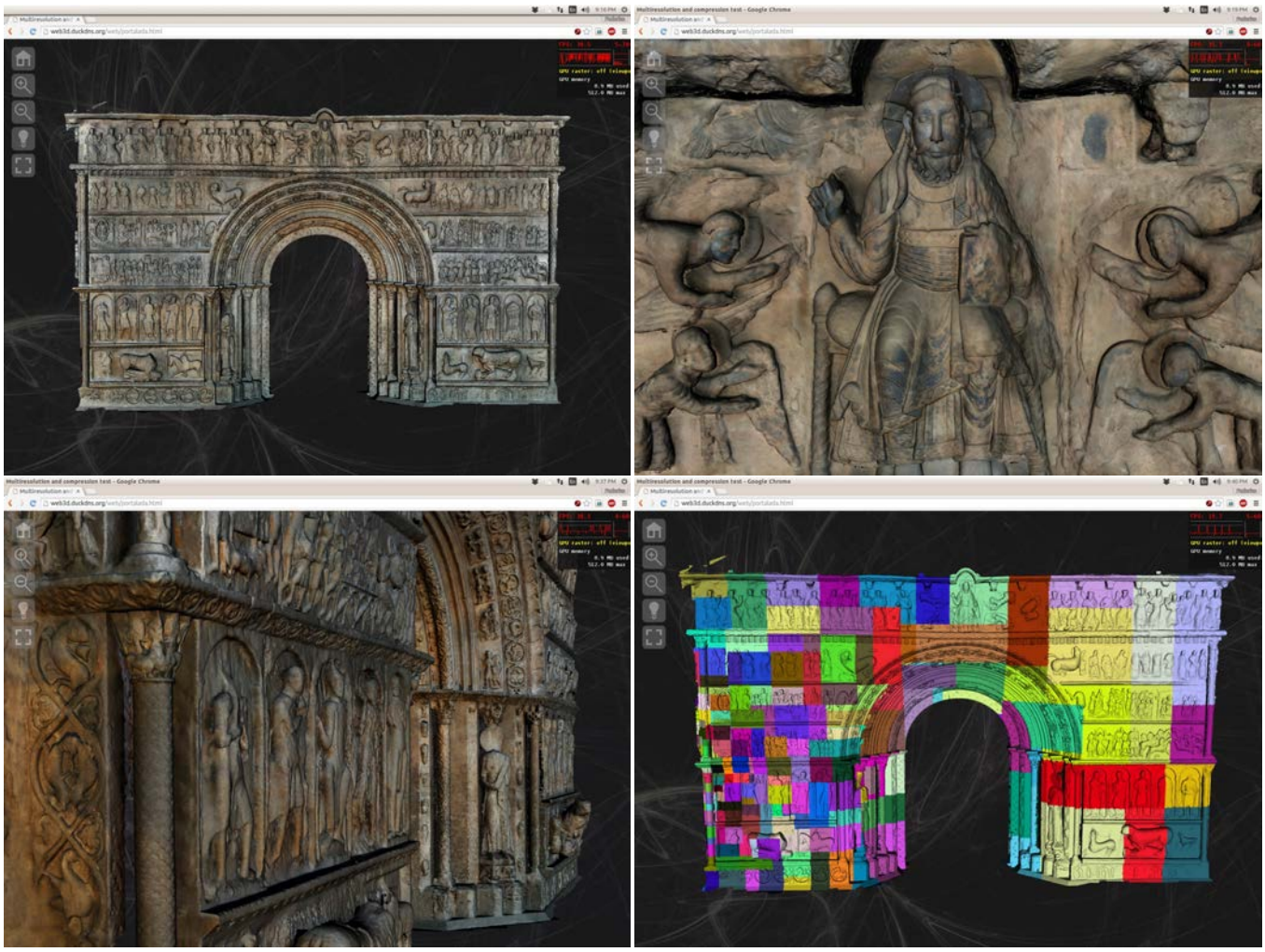


Figure 5: *Portalada rendered in a browser: top left: the full model, top right: a detail of the figure above the arch, middle right: the resolution of the model as seen from the middle left view point (without frustum culling)*

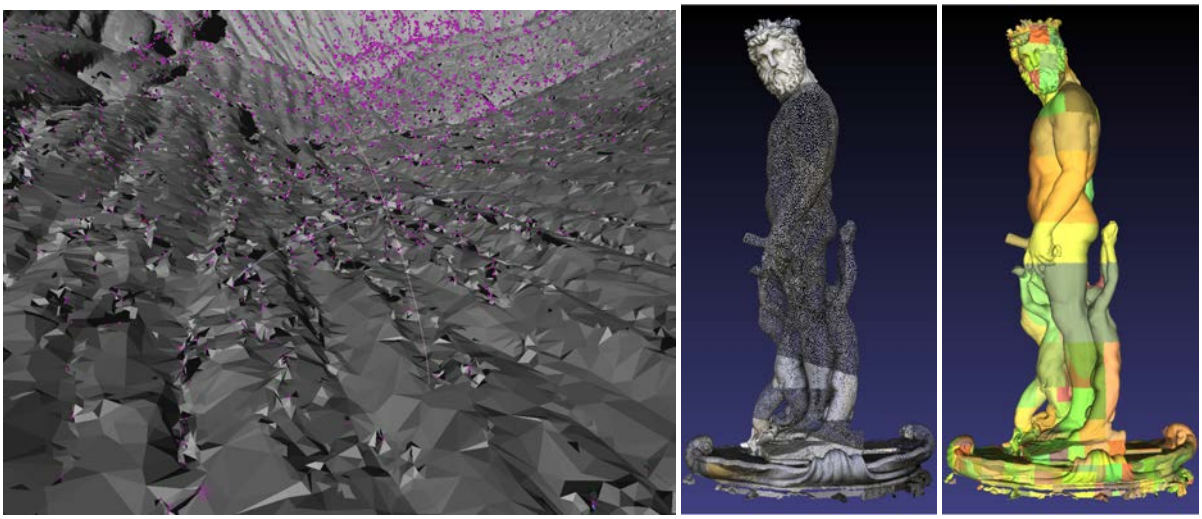


Figure 6: *Left: a model with severe topological issues. Right: a model with very imbalanced vertex distribution*

References

ALLIEZ, P., AND DESBRUN, M. 2001. Progressive compression for lossless transmission of triangle meshes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 195–202.

BAER, M. 2009. Efficient implementation of the generalized tunstall code generation algorithm. In *Proceedings of the 2009 IEEE International Conference on Symposium on Information Theory - Volume 1*, IEEE Press, Piscataway, NJ, USA, ISIT'09, 199–203.

BALSA RODRIGUEZ, M., GOBBETTI, E., MARTON, F., AND TINTI, A. 2013. Compression-domain seamless multiresolution visualization of gigantic meshes on mobile devices. In *Proc. ACM Web3D International Symposium*, New York, NY, USA, ACM Press, 99–107.

BEHR, J., ESCHLER, P., JUNG, Y., AND ZÖLLNER, M. 2009. X3dom: a dom-based html5/x3d integration model. In *Proceedings of the 14th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '09, 127–135.

BEHR, J., JUNG, Y., FRANKE, T., AND STURM, T. 2012. Using images and explicit binary container for efficient and incremental delivery of declarative 3d scenes on the web. In *Proceedings of the 17th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '12, 17–25.

BLUME, A., CHUN, W., KOGAN, D., KOKKEVIS, V., WEBER, N., PETTERSON, R. W., AND ZEIGER, R. 2011. Google body: 3d human anatomy in the browser. In *ACM SIGGRAPH 2011 Talks*, ACM, New York, NY, USA, SIGGRAPH '11, 19:1–19:1.

CHUN, W. 2012. Webgl models: end-to-end. In *OpenGL Insights*, P. Cozzi and C. Riccio, Eds. CRC Press, 431452. `http://www.openglinsights.com/`.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2003. Batching meshes for high performance terrain visualization. In *Second Annual Conference of Eurographics Italian Chapter*, Eurographics Association, Milano (ITALY), Conference Series.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2004. Adaptive tetrapuzzles: Efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Trans. on Graphics (SIGGRAPH 2004) 23*, 3, 796–803.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2005. Batched multi triangulation. In *Proceedings IEEE Visualization*, IEEE Computer Society Press, Conference held in Minneapolis, MI, USA, 207–214.

DELILLO, B., 2009. WebGLU: A utility library for working with WebGL . `http://webglu.sourceforge.org/`.

DI BENEDETTO, M., PONCHIO, F., GANOVELLI, F., AND SCOPIGNO, R. 2010. Spidergl: a javascript 3d graphics library for next-generation www. In *Proceedings of the 15th International Conference on Web 3D Technology*, ACM, New York, NY, USA, Web3D '10, 165–174.

DIRKSEN, J., Ed. 2013. *Learning Three.js: The JavaScript 3D Library for WebGL.* Packt Publishing.

DON BRUTZMANN, L. D. 2007. *X3D: Extensible 3D Graphics for Web Authors.* Morgan Kaufmann.

EVANS, A., AGENJO, J., AND BLAT, J. 2014. Web-based visualisation of on-set point cloud data. In *Proceedings of the 11th European Conference on Visual Media Production*, ACM, New York, NY, USA, CVMP '14.

EVANS, A., ROMEO, M., BAHREHMAND, A., AGENJO, J., AND BLAT, J. 2014. 3d graphics on the web: A survey. *Computers & Graphics 41*, 0, 43 – 61.

FLORIANI, L., PUPPO, E., AND MAGILLO, P. 1997. A formal approach to multiresolution hypersurface modeling. In *Geometric Modeling: Theory and Practice*, W. Strasser, R. Klein, and R. Rau, Eds., Focus on Computer Graphics. Springer Berlin Heidelberg, 302–323.

FLORIANI, L. D., MAGILLO, P., AND PUPPO, E. 1998. Compressing tins. In *In Proceedings of the 6th ACM Symposium on Advances in Geographic Information Systems*, 145–150.

GEELNARD, M. OpenCTM. `http://openctm.sourceforge.net/`.

GOBBETTI, E., MARTON, F., CIGNONI, P., BENEDETTO, M. D., AND GANOVELLI, F. 2006. C-BDAM – compressed batched dynamic adaptive meshes for terrain rendering. *Computer Graphics Forum 25*, 3 (September), 333–342. Proc. Eurographics 2006.

GOBBETTI, E., MARTON, F., RODRIGUEZ, M. B., GANOVELLI, F., AND DI BENEDETTO, M. 2012. Adaptive quad patches: An adaptive regular structure for web distribution and adaptive rendering of 3d models. In *Proceedings of the 17th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '12, 9–16.

GUMHOLD, S., AND STRASSER, W. 1998. Real time compression of triangle mesh connectivity. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '98, 133–140.

HOPPE, H. 1996. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '96, 99–108.

HOPPE, H. 1999. Optimization of mesh locality for transparent vertex caching. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '99, 269–276.

ISENBURG, M., AND GUMHOLD, S. 2003. Out-of-core compression for gigantic polygon meshes. *ACM Trans. Graph. 22*, 3 (July), 935–942.

JANKOWSKI, J., RESSLER, S., SONS, K., JUNG, Y., BEHR, J., AND SLUSALLEK, P. 2013. Declarative integration of interactive 3d graphics into the world-wide web: Principles, current approaches, and research agenda. In *Proceedings of the 18th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '13, 39–45.

KHRONOS GROUP, 2009. OpenGL ES - The Standard for Embedded Accelerated 3D Graphics.

KHRONOS GROUP, 2009. WebGL - OpenGL ES 2.0 for the Web.

KLEIN, S. T., AND SHAPIRA, D. 2011. On improving tunstall codes. *Inf. Process. Manage. 47*, 5 (Sept.), 777–785.

LAVOUÉ, G., CHEVALIER, L., AND DUPONT, F. 2013. Streaming compressed 3d data on the web using javascript and webgl. In

Proceedings of the 18th International Conference on 3D Web Technology, ACM, New York, NY, USA, Web3D '13, 19–27.

LEE, J., CHOE, S., AND LEE, S. 2010. Mesh geometry compression for mobile graphics. In *Proceedings of the 7th IEEE Conference on Consumer Communications and Networking Conference*, IEEE Press, Piscataway, NJ, USA, CCNC'10, 301–305.

LEE, H., LAVOU, G., AND DUPONT, F. 2012. Rate-distortion optimization for progressive compression of 3d mesh with color attributes. *The Visual Computer 28*, 2, 137–153.

LIMPER, M., JUNG, Y., BEHR, J., AND ALEXA, M. 2013. The pop buffer: Rapid progressive clustering by geometry quantization. *Computer Graphics Forum 32*, 7, 197–206.

LIMPER, M., WAGNER, S., STEIN, C., JUNG, Y., AND STORK, A. 2013. Fast delivery of 3d web content: A case study. In *Proceedings of the 18th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '13, 11–17.

LIMPER, M., THÖNER, M., BEHR, J., AND FELLNER, D. W. 2014. SRC - a streamable format for generalized web-based 3d data transmission. In *The 19th International Conference on Web3D Technology, Web3D '14, Vancouver, BC, Canada, August 8-10, 2014*, 35–43.

LIMPER, M., THÖNER, M., BEHR, J., AND FELLNER, D. W. 2014. Src - a streamable format for generalized web-based 3d data transmission. In *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies*, ACM, New York, NY, USA, Web3D '14, 35–43.

LIN, G., AND YU, T. P. Y. 2006. An improved vertex caching scheme for 3d mesh rendering. *IEEE Transactions on Visualization and Computer Graphics 12* (July), 640–648.

MAGLO, A., LAVOUÉ, G., DUPONT, F., AND HUDELOT, C. 2015. 3d mesh compression: Survey, comparisons, and emerging trends. *ACM Comput. Surv. 47*, 3 (feb), 44:1–44:41.

2013. Microsoft ActiveX Controls. `http://msdn.microsoft.com/`.

PAJAROLA, R., AND ROSSIGNAC, J. 2000. Squeeze: Fast and progressive decompression of triangle meshes. In *Proc. Computer Graphics Int'l (CGI 2000*, 173–182.

RAGGETT, D. 1995. Extending WWW to support platform independent virtual reality. *Technical Report*.

RODRÍGUEZ, M. B., GOBBETTI, E., MARTON, F., AND TINTI, A. 2013. Compression-domain seamless multiresolution visualization of gigantic triangle meshes on mobile devices. In *Proceedings of the 18th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '13, 99–107.

SANDER, P. V., AND MITCHELL, J. L. 2005. Progressive buffers: View-dependent geometry and texture lod rendering. In *Proceedings of the Third Eurographics Symposium on Geometry Processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SGP '05.

SANDER, P. V., SNYDER, J., GORTLER, S. J., AND HOPPE, H. 2001. Texture mapping progressive meshes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 409–416.

SANDER, P. V., NEHAB, D., AND BARCZAK, J. 2007. Fast triangle reordering for vertex locality and reduced overdraw. *ACM Trans. Graph. 26* (July).

SONS, K., KLEIN, F., RUBINSTEIN, D., BYELOZYOROV, S., AND SLUSALLEK, P. 2010. Xml3d: Interactive 3d graphics for the web. In *Proceedings of the 15th International Conference on Web 3D Technology*, ACM, New York, NY, USA, Web3D '10, 175–184.

SUTTER, J., SONS, K., AND SLUSALLEK, P. 2014. Blast: A binary large structured transmission format for the web. In *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies*, ACM, New York, NY, USA, Web3D '14, 45–52.

TUNSTALL, B. 1967. *Synthesis of Noiseless Compression Codes*. Georgia Institute of Technology.

YOON, S.-E., SALOMON, B., GAYLE, R., AND MANOCHA, D. 2004. Quick-vdr: interactive view-dependent rendering of massive models. In *Visualization, 2004. IEEE*, 131–138.

Enhanced Visualization of Detected 3D Geometric Differences

Gianpaolo Palma, Manuele Sabbadin, Massimiliamo Corsini and Paolo Cignoni

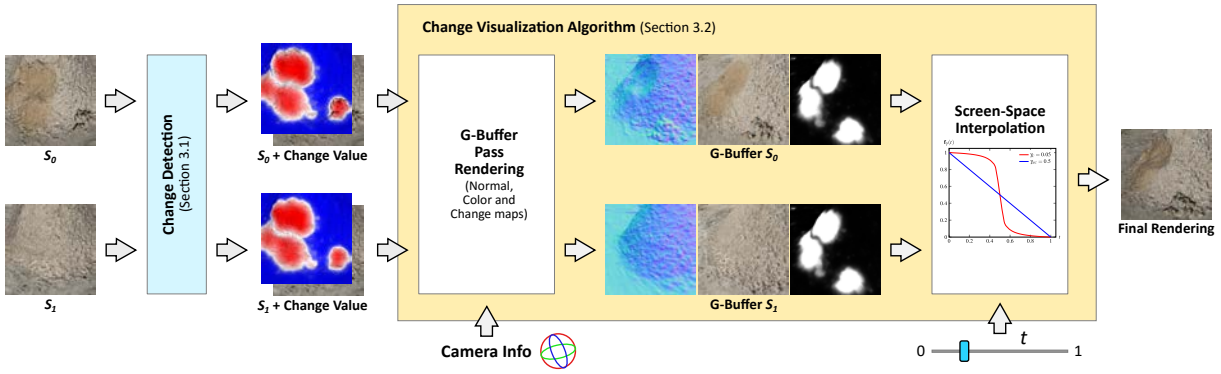


Fig. 1: Algorithm overview. After the computation of a change map for the input meshes S_0 and S_1, the proposed interactive change visualization technique computes the geometry buffers for the two meshes and blends them in screen space using the time t with two different interpolation functions: $\mathbf{f}_{\gamma_C}$ for the change pixels and $\mathbf{f}_{\gamma_{NC}}$ for the no-change pixels, enhancing significant changes and hiding non relevant, yet visible, differences.

Abstract— The wide availability of 3D acquisition devices makes viable their use for shape monitoring. The current techniques for the analysis of time-varying data are able to efficiently detect actual significant changes and rule out differences that are due to irrelevant variations of the data (like sampling, lighting, coverage). On the other hand, the effective visualization of such detected changes can be challenging when we want to preserve the actual appearance of the 3D model. In this paper, we propose a dynamic technique for the effective visualization of detected differences between two 3D scenes. The presented approach, while retaining the original appearance, allows the user to switch between the two models in a way that enhances the geometric differences that have been detected as significant and visually hides the other negligible, yet visibile, variations. The main idea is to use two distinct screen space interpolation functions for the significant 3D differences and the small variations to hide. We have validated the proposed approach in a user study on different types of datasets, proving the objective and subjective effectiveness of the method.

Index Terms—Interactive visualization, 3D differences, temporal change

1 Introduction

The wide availability of acquisition devices like cameras, smartphones and low-cost portable 3D scanners (e.g. Google Tango, iSense, StructureSensor), makes today possible an easy and fast harvesting of 2D and 3D temporal data of the world around us. The correct temporal analysis and interpretation of this data are important for the automatic and robust detection of geometric changes in the scene and the following effective visualization of the temporal evolution of these changes. In the last years several solutions have been proposed to solve the geometric change detection problem for different kind of applications (3D reconstruction [34], urban growth analysis [27] and natural events management [15]) using different input data, like image datasets [22], mixed acquisitions with 3D models and photos [27], or only geometric information [21]. On the other side, the effective visualization of temporal data to highlight and improve the understanding of the changes in the scene was concentrated on time-varying volumetric data [32][13][4] and on videos [1]. Usually, the existing solutions are based on static visualization of a single picture where the changes are encoded in some attribute like the color [20], the color saturation [30], the surface bump [4], glyphs [28][4]. The main problem of these solutions is that they make more difficult the shape understanding. The typical example is the overlay of a color map that encodes the change and no-change areas with different colors, making harder the interpretation of the geometric and color information owned by the model. Additionally, the existing methods are based on a limited or completed absent user interaction allowing for example only the navigation of the volumetric data. Up to now no methods are proposed to interact with the temporal dimension of the visual input in this specific context.

We propose an interactive technique to improve the visualization of the geometric changes between two 3D triangular meshes with color. The two models represent the same environment acquired at different times and can be generated by 3D scanning or by multi-view image reconstruction using digital photos. Our goal is to give the user a visualization tool with three main features: to allow a linear interaction model (slider) to alternate between the two time steps; to make more clear and as understandable as possible what is changing in the scene; to preserve all the original color and geometry attribute of the input models.

Our approach is based on the definition of a better temporal interpolation between the two 3D models that simultaneously tries to maximize the perception and the understanding of the most important and wide changed areas and to hide the nonsignificant, yet visible, differences that can distract the user's attention. These subtle differences are due to several reasons: geometric imperfections or noise due to the 3D reconstruction process, the high-frequency color shift due to the different illumination conditions during the photographic acquisition, the noise of the 3D scanner, actual small nonsignificant changes due

- *Gianpaolo Palma is with Visual Computing Lab - ISTI - CNR. E-mail: gianpaolo.palma@isti.cnr.it.*
- *Manuele Sabbadin is with Visual Computing Lab - ISTI - CNR. E-mail: manuele.sabbadin@isti.cnr.it.*
- *Massimiliano Corsini is with Visual Computing Lab - ISTI - CNR. E-mail: massimiliano.corsini@isti.cnr.it.*
- *Paolo Cignoni is with Visual Computing Lab - ISTI - CNR. E-mail: paolo.cignoni@isti.cnr.it.*

to small scale and fine details.

In particular, we assume the computation of a change probability map using a state-of-the-art method that segments out the change (dynamic) and no-change (static) areas in each 3D model. Our basic idea is to provide a screen-space interpolation technique of two renderings of the 3D models according to the user temporal interaction using different interpolation curves for the two classes of regions. For the choice of the interpolation curves we take into account the insights of the cognitive research on the Change Blindness phenomenon [26], that is the failure of the people to detect large changes in the scene, which normally would be easy to note, during the visual transition from one time to the next. Researchers have developed several different explanations for the occurrence of these phenomena [24], especially when a visual disruption is introduced during the transition between the two images. This disruption can take many forms like an eye movement, a flashed blank screen, a blink, a cut in a motion picture. The most interesting results for our work were presented by Simons et al. [25]. The authors conducted different perceptual experiments where observers viewed a scene throughout the change and they were actively trying to find what was changing. The scene was viewed or with a gradual condition, where the changes were presented as a 12 seconds animation created by dissolving one image of a pair into the other, or with a disruption condition, where one image of a pair was presented for 11250 ms, followed by a blank gray screen for 250 ms, then followed by the second image in the pair. Two different type of changes was tested independently: objects that appear or disappear from the scene replaced by an appropriate scene background; objects or regions of the scene with a color change. The analysis of the experiment results shows two important trends. When changes are sufficiently gradual, the visible change does not seem to draw attention, and large changes can go undetected. In these experiments, the rate of detection with a gradual transition is not better than when the two images are separated by a blank-screen disruption. More interesting are the results of the tests with a color change. In this case, the color changes are detected less often than an addition/deletion change and their detection was better in the disruption condition than in the gradual condition.

To summarize the main contributions of our work are:

- an interactive visualization method that highlights the main temporal geometric changes and, at the same time, minimizes the perception of irrelevant small color and geometry inconsistencies using different temporal interpolation curves;
- an interactive method that preserves all the color and geometry attributes of the input models without the use of overlay visual information, like an additional change color maps, that can make harder and difficult the understanding of the shape and color/geometric features of the underlying 3D models;
- a user study to understand what interpolation curves are more effective and preferred by the user in the visualization of the temporal evolution of a changing scene.

In Section 2 we introduce the state-of-the-art methods for the spatial-temporal comparative visualization of visual input, like images, video, volumetric dataset and 3D models. Then we describe our algorithm (Section 3) and finally we present the results obtained in a user study to evaluate the effectiveness of the proposed method (Section 4).

2 Related Work

The comparative visualization of the temporal data was studied in several application fields but up-to-now the attention was focused mainly on volumetric data. Pagendarm et al. [20] show the benefit of the scientific visualization using overlay layer with color coding to compare flow simulation and experimental data. Gleicher et al. [7] present a survey of visual comparison techniques trying to extract a taxonomy with three categories: juxtaposition to present each object separately; superposition to present multiple objects in the same coordinate system; explicit encoding of the relationships that encodes visual connections between objects.

Several works have been developed for the medical and biological visualization. DeLeeuw et al [14] present a system to visualize time-dependent data captured by a confocal microscopy of live 3D cells taking advantage of animations to improve the understating of some biological processes. Loomis et al. [16] propose a system for growing plant visualization based on a linear interpolation of different images acquired at different times. Tory et al. [28] present three different approaches to highlight changes in medical 4D data: semi-transparent isosurfaces colored by time; direct volume rendering encoding using intensity and change intensity over the time; glyph visualization of the change on the isosurface. The visual comparison of biomechanical motion and 3D data is proposed by Keefe et al. [12] with a mixed solution of overlay and side-by-side views assisted by a graph visualization of the change along a user selected direction. The side-by-side comparison is also used in the system VisTrails [2] to allow the user to create multiple temporal images to analyze.

Kok et al [13] use four different visualization techniques for multi-time CT data: side-by-side, comparison by switching, overlay and a checkerboard approach using tiles from different time images. The last approach is extended by Malik et al. [17] using hexagon cells to show more than two times. The visualization of statistical deformation models for 2D/3D medical images is analyzed by Caban et al. [4] and Hermann et al. [9]. Caban et al. [4] present four visualization algorithms: likelihood volumes to illustrate the probabilistic properties of a group of images; deformable grids to show statistical deformation properties and characterize regions with high variability; spherical glyphs to annotate the variability of different areas; line-based glyphs to illustrate deformation range and morphological variability. Hermann et al. [9] use the theory of stationary velocity fields for the interactive non-linear image interpolation and plausible extrapolation of large deformations.

To understand the spatio-temporal characteristics of time-varying volumetric data, Woodring et al. [32] use a high dimensional direct rendering of a 4D data field. They utilize different integration operators and volume transfer functions to present the spatio-temporal features to the user in an intuitive manner. Caban et al. [3] introduce a texture-based feature tracking technique to detect multiple features over time and find them in the following time volumes. Tracked objects are used to illustrate changes. Wang et al. [30] apply an importance-driven approach to time-varying volume data visualization to enhance the identification and presentation of the essential aspects. After the computation of the importance measure, they show a system to highlight a data cluster selected by the user changing the saturation of the color fragments. Joshi et al. [10] show how to use some techniques inspired from the illustration literature, like speed-lines, flow ribbons and strobe silhouettes, to help the user to see changes in the 3D volume visualization.

Some solutions are also proposed for videos and 3D meshes. Wu et al. [33] propose an algorithm to magnify small changes within the same video while Balakrishnan et al. [1] present a system for the comparison of two videos based on the overlays of the filtered temporal gradient. The color of the gradient edges in the final image is based on a measure of local dissimilarity. Nowell et al. [19] propose a system for temporal change visualization on content collections represented as a 3D landscape. The authors investigated three techniques for drawing attention to changes from one time to the next: 3D morphing; cross-fading; a wire-frame rendering of the emerging contours superimposed on the image.

In general, the existing solutions show some limitations. Some of them are not interactive, like the side-by-side approaches that make harder and time consuming the comparison task for the user. Other ones encode the changes or in some attributes, like the color, the color saturation, the surface bump, or with overlapped glyphs that make difficult the understanding and the interpretation of the geometric and color information of the model.

3 Algorithm

Let us consider two 3D meshes S_0 and S_1 representing the same scene at two different time steps; we assume that the models are aligned and

the geometric differences are already identified and segmented over the meshes.

We want to provide an interactive technique that allows the user to switch in a continuous way between the models to understand and discern the most important and consistent geometric changes. More precisely we assume the input models were preprocessed with a state-of-the-art method of automatic change detection (Section 3.1). This preprocessing step computes a probability *change map* that identifies the regions with a significant amount of geometric differences from the regions with no change or with small irrelevant low-amplitude/high-frequency spatial changes, for example color differences due to varying lighting conditions or geometric inconsistency due to incomplete or noisy data.

The proposed algorithm (Section 3.2) exploits the output of this change detection step and it is based on a novel blending approach of rendered fragment in screen-space using different temporal interpolation curves following the general insights of the studies on the *Change Blindness* phenomenon. In our context, we underline two important observations coming out from the *Change Blindness* experiments [25]. This type of blindness happens also when a gradual transition between the two times is used, making, in some cases, big changes less perceivable. Then the color changes are detected less often that a geometric change (for example an object that was added or removed from the scene) and their detection becomes trivial in the case of a simple instantaneous switch between the two times. These observations suggest us some hints in the design of our visualization algorithm. In particular, a smooth gradual transition for the regions identified as with *no-change* or with irrelevant geometric and/or color differences can help to hide as more as possible their perception at the user. On the other side, the trivial fast switch between the two time steps make simpler for the user to identify the significant changes but, at the same time, it emphasizes all the color differences making harder the visual detection of the geometric changes. This problem is very challenging especially with models by multi-view 3D reconstruction, where it is usual to have high-frequency shading variation due to the input photos acquired in different lighting conditions. Our approach tries to merge these two observations using different interpolation curves for the different regions of the 3D models in order to maximize the perception of most significant change areas and at the same time to hide the small differences in the others.

3.1 Change Detection

To automatically detect the changes between the input meshes we use a slight variant of the method presented in [21]. This method is based on two steps. A first step to detect the changes using the implicit surface defined by the point clouds under a Growing Least Square(GLS) [18] analysis; this kind of comparison produces more robust change classification results than the classical proximity measures. After this classification, a spatial reasoning step is performed to solve critical geometric configurations that are common in man-made environments, like an office or a home interior. The first step computes a multi-scale GLS descriptor on a uniform subsampling of the volume occupied by the point clouds, independently for each point cloud, and then it maps the differences of these descriptors in the time over the original models. In order to be more robust against meshes with large variation of in the local point density, for example meshes generated by a multi-scale data acquisition, we extend the method in [21] with an adaptive octree able to adjust the density of the descriptors with the point density of the meshes. Then since the spatial reasoning step is very sensitive to the noise we avoid this processing for the meshes generated by multi-view 3D reconstruction. The final results are two meshes with a $[0,1]$ per-vertex change field that can be interpreted as a change probability. We assume a small amount of smoothness in the change field so that, as shown in Figure 2, to help reducing rendering artifacts around the transition between change and no-change regions.

3.2 Change Visualization Algorithm

The proposed visualization algorithm computes the screen-space interpolation of the data of the 3D colored meshes S_0 and S_1 using different

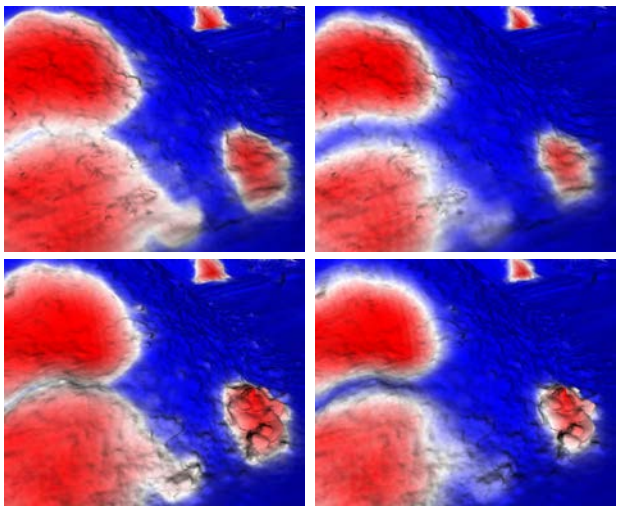


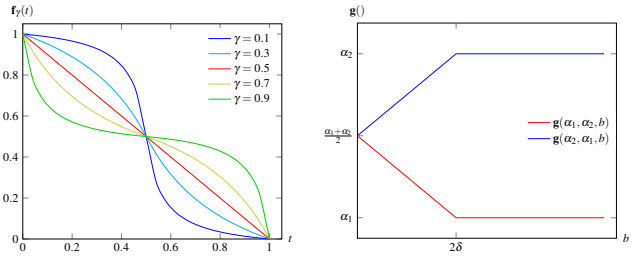
Fig. 2: *Blue-White-Red* color mapping of the $[0..1]$ change field computed with the method in [21]. Left column shows the two meshes S_0(top), S_1 (bottom) where the change threshold is set $d = 0.2$; Right column shows the same two meshes but with a different change threshold ($d = 0.5$).

interpolation curves for the areas with different change values. The segmentation of these areas depends on a segmentation change threshold d. The algorithm is composed of two steps. First, we render each model independently to generate the corresponding geometry buffers where we store for each pixel (x,y) the normal $\vec{\mathbf{n}}(x,y)$, the RGB color $\mathbf{c}(x,y)$ and the change value $\mathbf{q}(x,y)$ generated by barycentric interpolation of the vertex info of each triangle. In the second step, we compute the screen-space interpolation of the two geometry buffers using a time variable $t \in [0,1]$ whose value is controlled interactively by the user, for example with a slider. When the t is zero the tool shows S_0, when it is one the tool shows S_1, while for all the other values the tool shows a rendering obtained by a pixel-wise blending of the geometry buffer data controlled by the output of the function $\mathbf{f}_\gamma(t) : [0,1] \to [0,1]$. The used function $\mathbf{f}_\gamma(t)$ is a parametric smooth-step [23]:

$$\mathbf{f}_\gamma(t) = \begin{cases} 1 - \dfrac{t}{(1/\gamma - 2)(1-2t)+1} & t \leq 0.5 \\ \dfrac{1-t}{(1/\gamma-2)(2t-1)+1} & t > 0.5 \end{cases} \quad (1)$$

where the parameter $\gamma \in (0,1)$ determines its shape (Figure 3a). This function has two interesting properties: for $\gamma = 0.5$ we have the linear function $\mathbf{f}_{0.5}(t) = 1-t$; the functions $\mathbf{f}_\gamma(t)$ and $\mathbf{f}_{1-\gamma}(t)$ are symmetric with respect to the function $1-t$.

The main idea of our technique is to use two different interpolation functions using distinct shape parameters γ: γ_C for the *change* and γ_{NC} for the *no-change* regions. The evaluation of the best pair of functions to use for our purpose was done by means of a user study presented in Section 4. The outline of the algorithm is shown in Algorithm 1. The input are the geometry buffers of the two meshes and the two alpha values $\alpha_C = \mathbf{f}_{\gamma_C}(t)$ and $\alpha_{NC} = \mathbf{f}_{\gamma_{NC}}(t)$ to use in the interpolation of change and no-change pixels. For each pixel (x,y) of the screen we recover the corresponding data from the geometry buffers of each mesh (normals $\vec{\mathbf{n}}_0(x,y)$ and $\vec{\mathbf{n}}_1(x,y)$, colors $\mathbf{c}_0(x,y)$ and $\mathbf{c}_1(x,y)$, change values $\mathbf{q}_0(x,y)$ and $\mathbf{q}_1(x,y)$). This data are used to classify the pixel as change or no-change and to determine the type of interpolation to apply. A screen pixel is classified as no change only if both the meshes project on that pixel a value $\mathbf{q}(x,y)$ below the selected change segmentation threshold d. If at least one mesh projects a value $\mathbf{q}(x,y)$ above d the pixels is classified as change. In all the other case where only one mesh projects geometry on the pixel, due for example to data not acquired in the other time, the pixel is classified as change (in this case



(a) (b)

Fig. 3: (a) Parametric smooth-step function $\mathbf{f}_\gamma$ used for the temporal interpolation with different γ. (a) Function $\mathbf{g}(\alpha_1,\alpha_2,b)$ used for the interpolation of the blending factor near the boundary of binary segmentation obtained with the change threshold d.

the time with no information return a negative value $\mathbf{q}(x,y)$). The next step is the interpolation of the data. For the no-change pixels, we calculate the alpha blending of the two normals and colors independently and then we compute the final Lambertian shading with the interpolate values. Instead for the change pixel, we compute independently the Lambertian shading of each time and then we blend the computed shaded colors. For the pixel with a single valid time, we blend the shaded color with the background color of the screen. The computation of the blending of normals and colors before the shading permits to partially hide negligible variations like high frequency geometric and color differences due to non-relevant factors (see the additional video).

The use of a binary segmentation between the change and no-change areas and the choice of the respective interpolation functions with different parameters γ can produce visible rendering artifacts near the boundary of the segmentation (see Figure 4 for an example). To overcome this problem we interpolate the alpha parameters α_C and α_{NC} in a neighborhood of the change threshold d taking advantage of the smooth radial distribution of the change value around the real changes. In particular, for all the pixels with at least one change value $\mathbf{q}_0$ and $\mathbf{q}_1$ in the range $[d-\delta, d+\delta]$ we interpolate the alpha blending parameters using the following function $\mathbf{g}(\alpha_1,\alpha_2,|\mathbf{q}_0-d|+|\mathbf{q}_1-d|)$ (Figure 3b):

$$\mathbf{g}(\alpha_1,\alpha_2,b) = \begin{cases} \dfrac{\alpha_1+\alpha_2}{2} + \dfrac{(\alpha_1-\alpha_2)}{2}\dfrac{b}{2\delta} & b \le 2\delta \\ \alpha_1 & b > 2\delta \end{cases} \quad (2)$$

The obtained result is a smoother transition near the boundary of the binary segmentation that makes less abrupt and more pleasant the final color interpolation (Figure 4).

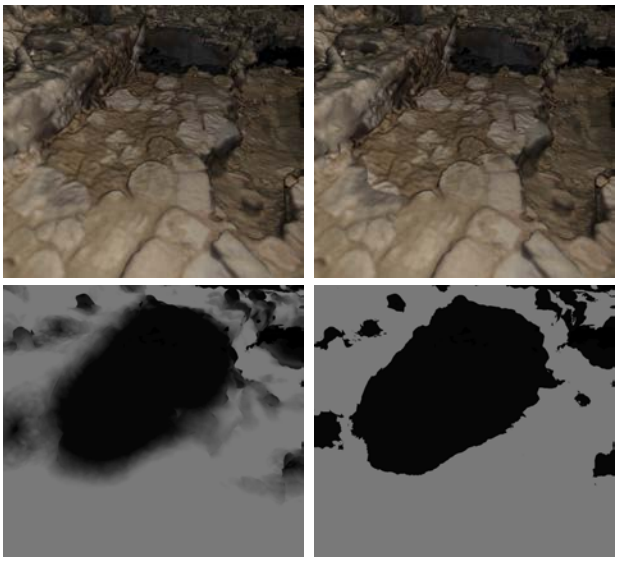


Fig. 4: Results obtained without (left) and with (right) the use of the interpolation of α_C and α_{NC} with the function $\mathbf{g}(\alpha_1,\alpha_2,b)$. The bottom row shows the corresponding weight map used for the final blending of the meshes. ($t = 0.8$, $\alpha_C = \mathbf{f}_{0.05}(t)$ and $\alpha_{NC} = \mathbf{f}_{0.95}(t)$, $d = 0.3$, $\delta = 0.15$).

Procedure 1 Temporal Interpolation Algorithm

Input: Geometry Buffers c,n,q for S_0 and S_1
Input: Time t, change threshold d, and light direction $\vec{l}$
Input: $\alpha_C = \mathbf{f}_{\gamma_C}(t)$ and $\alpha_{NC} = \mathbf{f}_{\gamma_{NC}}(t)$
Output: The pair $\langle$ *color*, *alpha* $\rangle$ of the pixel

for all pixel (x,y) **do**
  $\beta_C \leftarrow \mathbf{g}(\alpha_C,\alpha_{NC},|\mathbf{q}_0-d|+|\mathbf{q}_1-d|)$
  $\beta_{NC} \leftarrow \mathbf{g}(\alpha_{NC},\alpha_C,|\mathbf{q}_0-d|+|\mathbf{q}_1-d|)$
  if $(\mathbf{q}_0 < d \wedge \mathbf{q}_0 \ge 0 \wedge \mathbf{q}_1 < d \wedge \mathbf{q}_1 \ge 0)$ **then** ▷ *no change*
    $\vec{n} \leftarrow \beta_{NC}\,\vec{\mathbf{n}}_0 + (1-\beta_{NC})\,\vec{\mathbf{n}}_1$
    $c \leftarrow \beta_{NC}\,\mathbf{c}_0 + (1-\beta_{NC})\,\mathbf{c}_1$
    return $\langle\ c\ (\vec{n}\cdot\vec{l}\,),\ 1\ \rangle$
  else if $(\mathbf{q}_0 \ge 0 \wedge \mathbf{q}_1 \ge 0)$ **then** ▷ *change*
    $c \leftarrow \beta_C\,\mathbf{c}_0\,(\vec{\mathbf{n}}_0\cdot\vec{l}\,) + (1-\beta_C)\ \mathbf{c}_1\,(\vec{\mathbf{n}}_1\cdot\vec{l}\,)$
    return $\langle\ c,\ 1\ \rangle$
  else if $(\mathbf{q}_0 \ge 0)$ **then** ▷ *only* S_0
    return $\langle\ \mathbf{c}_0\,(\vec{\mathbf{n}}_0\cdot\vec{l}\,),\ \beta_C\ \rangle$
  else if $(\mathbf{q}_1 \ge 0)$ **then** ▷ *only* S_1
    return $\langle\ \mathbf{c}_1\,(\vec{\mathbf{n}}_1\cdot\vec{l}\,),\ 1-\beta_C\ \rangle$
  return $\langle$ *black*, $0\ \rangle$

4 User Study

In this section we present the results of the user study conducted to evaluate the effectiveness of the proposed technique. This user study is subdivided in two sessions. In the first one, we evaluate the effectiveness of the technique from an objective point of view asking to the subjects to perform a visual task. In the second session, we evaluate our technique in a subjective manner asking to the user to score how effective are four different presenting techniques in the task of clearly showing the visual differences in a evolving scene.

In the following we provide a detailed description of the dataset used, the techniques tested and the protocol followed in the two sessions.

4.1 Dataset

We used six datasets with different characteristics and created with different reconstruction techniques. In the following we indicate the original scene acquired at a certain instant time t_0 with S_0 and the same scene acquired at the time $t_1 > t_0$ with S_1. Figure 5 and Figure 6 show the 11 viewpoints used in the user study with the relative rendering parameters: d the threshold for the change/no-change segmentation; δ for the range of change values near the threshold d where to interpolate the blending weights α_C and α_{NC} avoiding abrupt color variations.

Santa Marta The dataset shows an archeological excavation generated from two sets of photos (129 and 98 respectively) acquired with a drone. The photographic acquisition was done in two different years using two different cameras (12Mp and 24Mp). The meshes have been generated using the MVE system [6] and have per-vertex color. The meshes are characterized by high-frequency differences due to the higher photo resolution of the second acquisition that generated denser color and geometry details. We select 3 viewpoints for this dataset (ST.MARTA1, ST.MARTA2 and ST.MARTA3).

Arene de Lutece This dataset shows the Arene de Lutece in Paris. The meshes are generated by two large sets of photos acquired by several people with different cameras and smart-phones in two different days (1500 and 6000 photos). This dataset shows a multi-scale

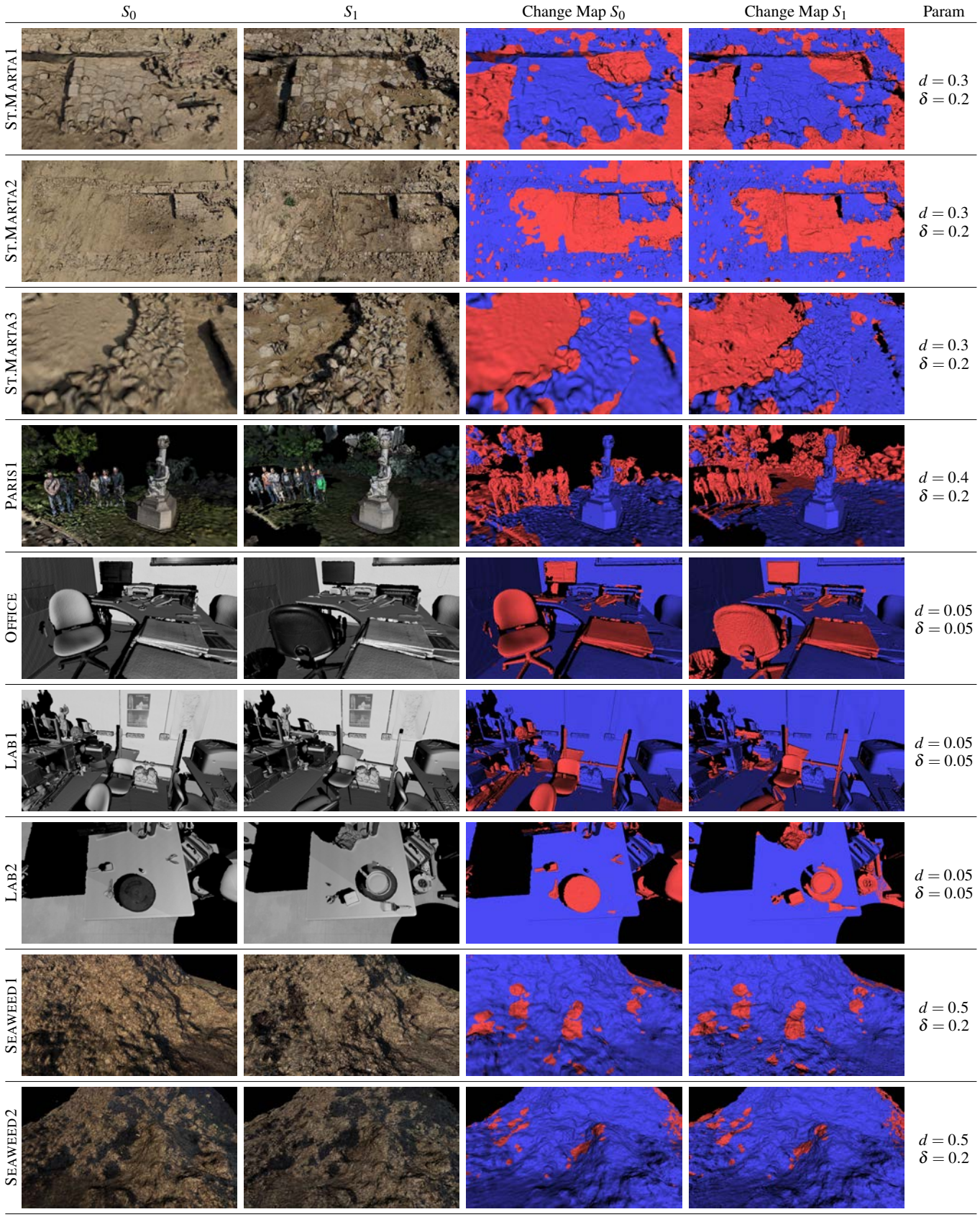


Fig. 5: Viewpoints used in the user study. Each viewpoint shows the two times, the relative change maps computed with [21] (blue = no-change, red = change) and the parameters d for the change/no-change segmentation and δ for the interpolation near the binary segmentation.

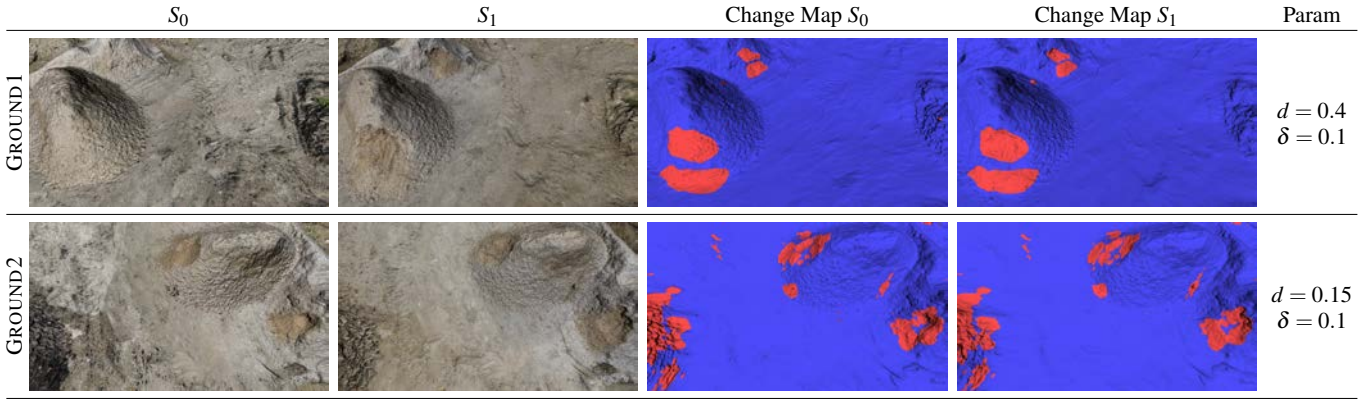


Fig. 6: Viewpoints used in the user study. Each viewpoint shows the two times, the relative change maps computed with [21] (blue = no-change, red = change) and the parameters d for the change/no-change segmentation and δ for the interpolation near the binary segmentation.

acquisition, where the same details are acquired with very different level of details; we used the MVE system [6] and the FSSR algorithm [5] to preserve all the multi-scale acquired geometric details. The meshes have per-vertex color. The meshes show geometric noise due to different type of cameras and high color shading variation due to the different lighting condition during the photographic acquisition. We selected a single viewpoint for this dataset (PARIS).

Office The dataset is composed by two *time-of-flight* scans of an office. The meshes are created using Screened Poisson Surface Reconstruction [11]. The meshes have a per-vertex gray scale color (laser reflectance). The meshes are very accurate and show only large changes (objects that are moved or that appear) with no noise and small inconsistencies. We selected a single viewpoint for this dataset (OFFICE).

Lab The dataset is composed by two *time-of-flight* scans of a laboratory. The meshes are created using Screened Poisson Surface Reconstruction [11]. The meshes have a per-vertex gray scale color (laser reflectance). The meshes are very accurate and show only large changes with no noise and small inconsistencies. We selected two viewpoints for this dataset (LAB1, LAB2).

Seaweed Pile The dataset shows a seaweed pile on the beach acquired with two set of photos in different moments of the day (in the morning and in the afternoon). The two sets contains 76 and 86 photos acquired with a smartphone. The final meshes have been generated using Screened Poisson Surface Reconstruction [11] using as input the dense point cloud computed by Agisoft Photoscan. The geometric changes are very difficult to note due to the color variation of the final meshes. We selected two viewpoints for this dataset (SEAWEED1, SEAWEED2).

Ground Pile The dataset shows a ground pile acquired with three set of photos in different days (55, 79 and 106 respectively). The final meshes have been generated using Screened Poisson Surface Reconstruction [11] using as input the dense point cloud computed by Agisoft Photoscan. The geometric changes are very localized, but the meshes show some inconsistency with high frequency color and geometry variation due to the different lighting condition during the photographic campaign and due to the noise of the input dense point cloud. We selected two viewpoints for this dataset (GROUND1, GROUND2).

4.2 Techniques

In our study we compare four techniques: switch between images (SWITCH), linear blending (LINEAR), and the two variants of the proposed technique (SMOOTHSTEP1 and SMOOTHSTEP2). Figure 7 shows the shape of the interpolation functions $\mathbf{f}_{\gamma_C}(t)$ and $\mathbf{f}_{\gamma_{NC}}(t)$ used by each technique for the change and the no-change pixels. The two methods SWITCH and LINEAR use the same function for both the categories $\mathbf{f}_{\gamma_C}(t) = \mathbf{f}_{\gamma_{NC}}(t)$.

The switch, or instantaneous alternation, between the different scenes is the typical way of comparing different images inside software which use layers. For example, many image processing, image editing and GIS software tools use layers to show different information or different parts of overlapping images. We implement the method with an instantaneous transition from one time to the next at the time $t = 0.5$. The linear blending has been selected to understand if the simple mixing makes able the subject to interpret well the scene changes. The method has been implemented with a function with parameter $\gamma_C = \gamma_{NC} = 0.5$.

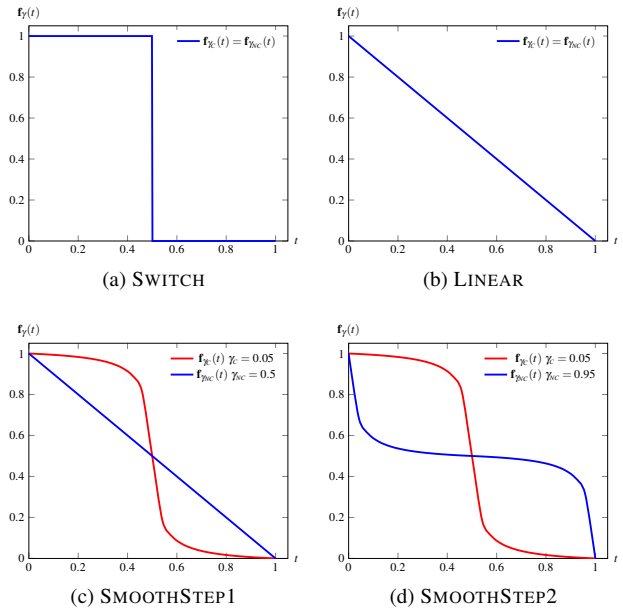
The first variant of the proposed method (SMOOTHSTEP1) uses a linear blending for the no-change regions ($\gamma_{NC} = 0.5$) and a smooth-step interpolation for the change areas ($\gamma_C = 0.05$). The second variant of the method (SMOOTHSTEP2) uses a smooth-step interpolation for both the classes with two different symmetric functions ($\gamma_C = 0.05$ and $\gamma_{NC} = 0.05$). The main difference of the these methods is in the perception of the no-change regions. In the method SMOOTHSTEP1 the no-change regions are interpolated in a linear way avoiding abrupt color changes that make the negligible differences perceivable. At the same time the real changes appear very compressed in the central times around $t = 0.5$. In the method SMOOTHSTEP2 the interpolation of the no-change regions happens in two separated moments near $t = 0$ and $t = 1$ in a fast way while it stays completely still during the interpolation of the change areas (around $t = 0.5$). In this way we ensure no distraction factors when the change is shown.

4.3 Protocol

The user study is subdivided in two sessions with the purpose of evaluating both objectively and subjectively the proposed visualization technique. At the begin of each session we make a training phase where the subject is instructed with the task to perform in the session, listens a description of the GUI and practices a short time with the system. The training aims to guarantee that the subject understands the use of the interface for the specific task.

4.3.1 First Session - Objective evaluation

To evaluate the effectiveness of the approach we measure how well the subjects are able to correctly identify *change/no-change* areas in a number of scenes by tagging squares of a superimposed grid. Each test consists of one of the eleven viewpoints previously listed and shown in Figure 5 and 6. A random technique from the four under investigation, is chosen and used for each user/scene combination. The subject sees this transition automatically animated from S_0 to S_1 and back to S_0. The animation lasts 5 seconds: 250ms on S_0, 2000ms for the transition from S_0 to S_1, 500ms on S_1, 2000ms for the transition back from S_1 to S_0 and 250ms on S_0. The subject can see the animated transition 3 times. The transition duration and the number of repetition was chosen as sufficient to pay attention on the various parts of the scene [31] but



(a) SWITCH (b) LINEAR

(c) SMOOTHSTEP1 (d) SMOOTHSTEP2

Fig. 7: Shape of interpolation functions $\mathbf{f}_{\gamma_C}$ and $\mathbf{f}_{\gamma_{NC}}$ used by the four techniques tested in the user study.

Fig. 8: Example of inputs provided by the subject in two different scenes. Red tiles indicate perceived changes, blue tiles indicate no changes, no input indicates that the user has no a clear choice on that tiles.

yet short enough to make the identification task non trivial. At the end of the third animation, the subject indicates the areas of changes.

The training phase consists in instructing the subject about the kind of changes to indicate and how to enter the input. Concerning the kind of changes, we asked to indicate *significant* geometric difference for each scene and to avoid that non-relevant geometric changes could bias the subject. For example, a wall of the scene can present more or less geometric details depending on the position of the laser scanner in the two acquisition sessions even if a real change has not happen. In this way we avoided confusion and simplified the task requested. To better assess what significant differences are, we mention objects or parts of the scene with appear or disappear and parts of the scene with change its shape significantly. Since we design our technique with the goal to "hide" minor/moderate color changes we deliberately not mentioned anything about color differences.

Finally, the subject is instructed to how indicate the detected visual changes on a 7 grid overdrawn on the scene. The subject indicate with a mouse click the tiles where a change is detected (red tiles), with two clicks the tiles that do not change from S_0 to S_1 (blue tiles). The tiles for which the subject is uncertain or does not remember the status should remain un-tagged and are recorded as "no answer" (Figure 8).

4.3.2 Second Session - Subjective evaluation

The training phase for this session is simpler and shorter then the previous one. The subject is instructed to test the different visualization techniques directly and rate them with a score from 1 to 5 according to his/her preference. In this experiment the subject knows the

Fig. 9: (Top) User graphics interface used in the main monitor. For each technique the subject can interact with the corresponding slider to evaluate its effectiveness in the visualization of the geometric differences. (Bottom) Layout used during the second part of the study. A false color map is shown on a monitor close to the one in front of the subject. On the main screen the subject can interact with the four techniques.

change/no-change segmentation of the scene that is displayed on a side monitor during the interactive session with a false color map (Figure 9). Note that all the techniques are visualized simultaneously so that the subject can better appreciate the different effects produced. We asked to evaluate with '1' the techniques that are not effective to show the changes and with '5' the techniques that are very effective. We asked also to consider in the scoring the effectiveness in hiding which parts of the scene does not change. The subject evaluates the techniques by moving the time slider provided for each technique. The final recommendation of the training phase is to play with all the range of the slider to better evaluate the results that the specific technique produces. For this session we selected five viewpoints from that in Figure 5 and 6 (ST.MARTA1, ST.MARTA2, OFFICE, SEAWEED1, GROUND1).

4.4 Data Analysis and Discussion

Twenty-one volunteers have participated in the first session of the user study and twenty-four in the second session. Most of them (60%) are computer scientists, while others are not. The subjects' age ranges from 30 to 52 and the number of males (75%) is greater than the number of females (25%).

The data of the first session have been aggregated by measuring, for each technique, the tiles correctly identified as "change" (*C*), the tile correctly identified as "no-change" (*NC*) and the "no answered" tiles (*NA*). We compute the weights $w_C = \min(1, \frac{2a}{c})$ for each tile indicated by the subject as change and $w_{NC} = \min(1, \frac{2b}{c})$ for each tile indicated by the subject as no-change, where c is the number of pixels in the tile where at least one mesh projects geometry, a is the number of these pixels classified as change and b is the number of pixel classified as no-change. For the pixel classification we use the same approach in Algorithm 1: a pixel is classified as no-change iff both the models project a change value $\mathbf{q}$ below the segmentation threshold d on the pixel; for all the other cases the pixel is classified as change. The final values C are computed as average of w_C of all the tiles indicated as change for the technique (the same for the values *NC*). Instead, *NA* is simply the percentage of tiles with no answers provided. The results

	Change			No-Change			No-Answer	
	C	#Tiles	Score	*NC*	#Tiles	Score	*NA*	#Tiles
SWITCH	0.680(±0.136)	437	0.382(±0.070)	0.931(±0.043)	421	0.633(±0.058)	0.46	730
LINEAR	0.758(±0.107)	351	0.448(±0.092)	0.923(±0.051)	643	0.695(±0.059)	0.374	595
SMOOTHSTEP1	0.738(±0.109)	478	0.497(±0.060)	0.958(±0.027)	600	0.717(±0.038)	0.319	506
SMOOTHSTEP2	0.729(±0.118)	410	0.420(±0.076)	0.929(±0.049)	507	0.655(±0.055)	0.412	643

Table 1: Results from the objective evaluation session. All the techniques are almost equally efficient for the identification of the scene changes with the exception of SWITCH. The method SMOOTHSTEP1 performs better than the other two methods yielding a lower number of no-answer (*NA*) and a higher number of correct change tiles detected ($0.738 \times 643 = 352.7$ tiles). This is confirmed by the global score for change 0.497 and no-change 0.717. The numbers in parenthesis are the corresponding variances.

	Score
SWITCH	1.68(±0.89)
LINEAR	2.50(±1.40)
SMOOTHSTEP1	4.10(±0.70)
SMOOTHSTEP2	3.90(±1.09)

Table 2: Techniques subjective evaluation. Note the high preference of SMOOTHSTEP1 method and its variant SMOOTHSTEP2 w.r.t to the other techniques tested. The numbers in parenthesis are the variance.

for each technique are summarized in Table 1 (means and variances for *C* and *NC*, the unknown rate *NA* and the absolute number of tile for each categories #Tiles). See the additional material for the data aggregated by scene.

From these results is possible to note that, despite the intrinsic difficulty of the task requested, the subjects are quite good to identify changes between the scenes. The SWITCH technique is the one that performs poorly. Instead, the other three methods have similar performance for the value *C* and *NC* even if the SMOOTHSTEP1 technique is quite better in reducing the number of tiles that received "no answer". This means that the SMOOTHSTEP1 method helps more the subject to understand where the changes are. This is confirmed by the higher number of tiles (column #Tiles) indicated by the subjects as change (478), which is the 36% higher than LINEAR and 16% higher than SMOOTHSTEP2. A better evaluation can be done computing a global score for the detection of change and no change considering the unknown rate *NA*. This score is reported in the columns #Score for the change and the no-change in the Table 1. The value of this score is obtained by computing for each single test two performance parameters: the right change score *C* and the right no-change score *NS* of the test weighted with the percentage of tiles with an answer. This per-test score are aggregated for each technique. In this way the contribution of a subject with a test with few no-answer tiles is higher in the computation of the final score. With this new score, it is more clear that the technique SMOOTHSTEP1 performs better than the others methods. The fact that the three methods are almost equally efficient for the identification of the scene changes can indicate that the task is organized in a fair way with respect to 4 tested techniques.

Concerning the second session we analyzed the scores collected in order to identify and remove scoring bias. To do so, we model the score s_{ij} provided by the subject i for the technique j as:

$$s_{ij} = g_i s_j + b_i + n_{ij} \tag{3}$$

where s_j is the "real" score of the answer j, g_i is a gain factor, b_i is an offset, and n_{ij} is a source of noise sampled from a zero-mean, white Gaussian which models eventual systematic or random errors. In this model, the gain and the offset vary from subject to subject, since any subject provides scores according to an own scale. By aggregating the scores for each technique and performing an analysis-of-variance (ANOVA) we found that a simple mean normalization [8] of the scores is sufficient to remove differences across subjects.

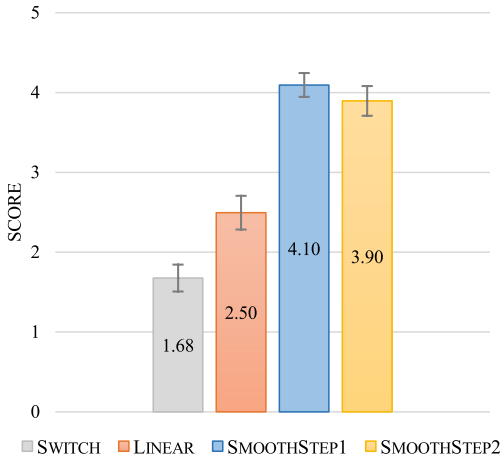


Fig. 10: Average scores and 95% confidence intervals of the four tested techniques without the subjects detected as outliers.

These scores had been also analyzed using a Kurtosis analysis in order to identify a range of values for which a subject can be considered an outlier and then screened. The screening procedure follow the Annex 2 of ITU BT.500 Recommendation [29]. The procedure depends on if the scores distribution can be considered or not a normal distribution. In case of a normal distribution the lower bound is set to be $\mu - 2\sigma$, while the upper bound $\mu + 2\sigma$ (μ is the mean of the scores and σ the standard deviation). This limits change to $\mu - \sqrt{20}\sigma$ and $\mu + \sqrt{20}\sigma$ for a non-normal distribution. Then, indicating with P the number of scores under the lower bound limit and with Q the number of scores over the upper bound limit, a subject is screened if $(P+Q)/N > 0.05$ AND $(\mathrm{P}-\mathrm{Q})/(\mathrm{P}+\mathrm{Q}) < 0.3$. Following this procedure we found that 3 of 24 subjects may be outliers, so they were removed from the final analysis. Table 2 contains the aggregated scores (average and variance) for each techniques after the outliers screening. Figure 10 shows a chart with the average scores and the relative 95% confidence interval. Table 3 and Figure 11 show the scores aggregated for each scene.

The results in Table 2 show that for the user the most effective method for the visualization of the geometric differences is SMOOTHSTEP1 (highest score and lowest variance). The method SMOOTHSTEP2 has a slightly worse rate but is almost as effective as SMOOTHSTEP1. Its lower score is due to the no-change interpolation function that makes clearly noticeable miminum color differences between the meshes at the beginning and at the end of the slider. This features can disturb the user especially in dataset with high-frequency color variation, like SEAWEED1 and GROUND1. On the contrary, the method LINEAR presents a very low score despite its high evaluation in the fist session of the user study. Finally, the method SWITCH confirms the bad performance already emerged in the first session.

Finally using the Kolmogorov-Smirnov (KS) test, we can reject the

	Scenes				
	ST.MARTA1	ST.MARTA2	OFFICE	SEAWEED1	GROUND1
SWITCH	1.47(±0.73)	1.72(±0.87)	2.33(±1.17)	1.29(±0.50)	1.57(±0.53)
LINEAR	2.95(±1.57)	2.57(±1.20)	2.62(±1.38)	1.90(±0.94)	2.43(±1.29)
SMOOTHSTEP1	4.05(±0.81)	3.86(±0.7)	3.86(±0.69)	4.38(±0.52)	4.33(±0.51)
SMOOTHSTEP2	3.86(±1.17)	3.86(±1.46)	3.67(±1.08)	4.00(±0.86)	4.10(±0.75)

Table 3: Subjective evaluation data aggregated by scene. The methods SMOOTHSTEP1 and SMOOTHSTEP2 are the most effective in each tested scene with a slightly higher preference for the method SMOOTHSTEP1. The numbers in parenthesis are the variance of the subject's scores.

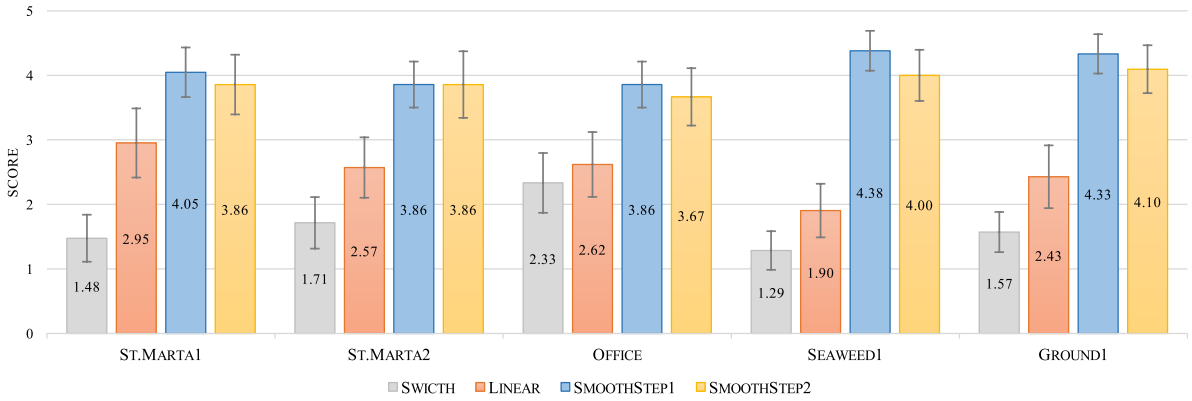


Fig. 11: Average scores and 95% confidence intervals of the four tested techniques aggregated for each scene without the subjects detected as outliers.

null hypotheses that the pairs SWITCH-LINEAR ($p = 6.94 \times 10^{-7}$), SWITCH-SMOOTHSTEP1 ($p = 1.97 \times 10^{-29}$), SWITCH-SMOOTHSTEP2 ($p = 3.25 \times 10^{-23}$), LINEAR-SMOOTHSTEP1 ($p = 1.35 \times 10^{-12}$) and LINEAR-SMOOTHSTEP2 ($p = 1.06 \times 10^{-9}$) are from the same distribution. On the contrary the pair SMOOTHSTEP1-SMOOTHSTEP2 are strictly correlated ($p = 0.78$) as we expect. This correlation is also indirectly confirmed by the opinions of the subjects at the end of the second session where many people declared that the methods SMOOTHSTEP1 and SMOOTHSTEP2 are in general very similar and hard to distinguish.

For sake of documentation, and for the interested reader, we have included as additional material the non-aggregated results of the two sessions of the user study with all the C/NC/NA taggings and the scores provided by the users for the presented viewpoints and techniques.

5 CONCLUSION

We have proposed a new interactive visualization method to better understand the 3D temporal geometric changes of an evolving scene. Starting from a change/no-change segmentation of two input 3D meshes, the method computes a screen-space interpolation of the two geometry buffers using different interpolation curves according to the change/no-change. In particular, exploiting the insights of the cognitive experiments on *Change Blindness*, we propose two different interpolation functions with two goals: to improve the perception of the most significantly changed regions and to hide the negligible, yet visually evident, color and geometric differences due to noise and high-frequency variations (small acquisition imperfections, color shading variation due to different lighting, small non-significant changes). To evaluate the interpolation functions, we conducted a user study to test the two proposed methods SMOOTHSTEP1 and SMOOTHSTEP2 with two other techniques: a simple fast switch (SWITCH) and a linear blending (LINEAR). The user test was organized in two sections to evaluate, in objective and subjective way, the effectiveness of each technique in clearly showing what was marked as changed in the scene. Analyzing the test results we can conclude that the proposed methods are the most effective and the most preferred techniques with a slightly better score for the technique SMOOTHSTEP1. More precisely we showed that the method SMOOTHSTEP1 is the most efficient in helping the subject to understand where is the change with less uncertainty (it has the lowest number of areas with no answer) and similarly, in the subjective evaluation, the method SMOOTHSTEP1 received the highest score with the lowest variance.

Since the proposed approach is very simple to implement, it could be adapted for the change visualization of other input, like 2D images, in order to enhance only the most important changes in noisy data.

REFERENCES

[1] G. Balakrishnan, F. Durand, and J. Guttag. Video diff: Highlighting differences between similar actions in videos. *ACM Trans. Graph.*, 34(6):194:1–194:10, Oct. 2015.

[2] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Vistrails: enabling interactive multiple-view visualizations. In *Visualization, 2005. VIS 05. IEEE*, pages 135–142, Oct 2005.

[3] J. Caban, A. Joshi, and P. Rheingans. Texture-based feature tracking for effective time-varying data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1472–1479, Nov. 2007.

[4] J. J. Caban, P. Rheingans, and T. Yoo. An evaluation of visualization techniques to illustrate statistical deformation models. In *Proc. EuroVis*, EuroVis'11, pages 821–830, Chichester, UK, 2011. The Eurographs Association & John Wiley & Sons, Ltd.

[5] S. Fuhrmann and M. Goesele. Floating scale surface reconstruction. *ACM Trans. Graph.*, 33(4):46:1–46:11, July 2014.

[6] S. Fuhrmann, F. Langguth, N. Moehrle, M. Waechter, and M. Goesele. Mvean image-based reconstruction environment. *Computers & Graphics*, 53, Part A:44 – 53, 2015.

[7] M. Gleicher, D. Albers, R. Walker, I. Jusufi, C. D. Hansen, and J. C. Roberts. Visual comparison for information visualization. *Information Visualization*, 10(4):289–309, Oct 2011.

[8] J. Guilford. *Psychometric methods*. McGraw-Hill series in psychology. McGraw-Hill, 1954.

[9] M. Hermann, A. C. Schunke, T. Schultz, and R. Klein. Accurate interactive visualization of large deformations and variability in biomedi-

cal image ensembles. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):708–717, 2016.

[10] A. Joshi and P. Rheingans. Illustration-inspired techniques for visualizing time-varying data. In *Visualization, 2005. VIS 05. IEEE*, pages 679–686, Oct 2005.

[11] M. Kazhdan and H. Hoppe. Screened poisson surface reconstruction. *ACM Trans. Graph.*, 32(3):29:1–29:13, July 2013.

[12] D. Keefe, M. Ewert, W. Ribarsky, and R. Chang. Interactive coordinated multiple-view visualization of biomechanical motion data. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1383–1390, Nov. 2009.

[13] P. Kok, M. Baiker, E. A. Hendriks, F. H. Post, J. Dijkstra, C. W. G. M. Lowik, B. P. F. Lelieveldt, and C. P. Botha. Articulated planar reformation for change visualization in small animal imaging. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1396–1404, Nov 2010.

[14] W. C. D. Leeuw, R. V. Liere, P. J. Verschure, A. E. Visser, E. M. M. Manders, and R. V. Drielf. Visualization of time dependent confocal microscopy data. In *Proc. Visualization*, pages 473–476, Oct 2000.

[15] Y. Li, X. Fan, N. J. Mitra, D. Chamovitz, D. Cohen-Or, and B. Chen. Analyzing growing plants from 4d point cloud data. *ACM Transaction on Graphics*, 32(6):157:1–157:10, Nov. 2013.

[16] J. J. Loomis, X. Liu, Z. Ding, K. Fujimura, M. L. Evans, and H. Ishikawa. Visualization of plant growth. In *Proc. Visualization*, pages 475–478, Oct 1997.

[17] M. M. Malik, C. Heinzl, and M. E. Groeller. Comparative visualization for parameter studies of dataset series. *IEEE Transactions on Visualization and Computer Graphics*, 16(5):829–840, Sept 2010.

[18] N. Mellado, G. Guennebaud, P. Barla, P. Reuter, and C. Schlick. Growing least squares for the analysis of manifolds in scale-space. *Computer Graphics Forum*, 31(5):1691–1701, Aug. 2012.

[19] L. Nowell, E. Hetzler, and T. Tanasse. Change blindness in information visualization: a case study. In *Proc. INFOVIS*, pages 15–22, 2001.

[20] H. Pagendarm, F. Post, D. U. of Technology, D. U. of Technology. Faculty of Technical Mathematics, and Informatics. *Comparative Visualization: Approaches and Examples*. Reports of the Faculty of Technical Mathematics and Informatics. Delft University of Technology, 1995.

[21] G. Palma, P. Cignoni, T. Boubekeur, and R. Scopigno. Detection of geometric temporal changes in point clouds. *Computer Graphics Forum*, page in Press, 2015.

[22] T. Pollard and J. L. Mundy. Change detection in a 3-d world. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–6, Minneapolis, 2007. IEEE.

[23] C. Schlick. Fast alternatives to perlin's bias and gain functions. In P. S. Heckbert, editor, *Graphics Gems IV*, pages 401–403. Academic Press Professional, Inc., San Diego, CA, USA, 1994.

[24] D. J. Simons. Current approaches to change blindness. *Visual Cognition*, 7(1-3):1–15, 2000.

[25] D. J. Simons, S. L. Franconeri, and R. L. Reimer. Change blindness in the absence of a visual disruption. *Perception*, 29(10):1143–1154, 2000.

[26] D. J. Simons and D. T. Levin. Change blindness. *Trends in cognitive sciences*, 1(7):261–267, 1997.

[27] A. Taneja, L. Ballan, and M. Pollefeys. City-scale change detection in cadastral 3d models using images. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 113–120, Columbus, 2013. IEEE.

[28] M. Tory, N. Röber, T. Möller, A. Celler, and M. S. Atkins. 4d space-time techniques: A medical imaging case study. In *Proc. Visualization*, VIS '01, pages 473–476, Washington, DC, USA, 2001. IEEE Computer Society.

[29] I. T. Union. ITU-T Rec. BT.500-11: Methodology for subjective assessment of the quality of television pictures, 2002.

[30] C. Wang, H. Yu, and K. L. Ma. Importance-driven time-varying data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1547–1554, Nov 2008.

[31] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[32] J. Woodring, C. Wang, and H.-W. Shen. High dimensional direct rendering of time-varying volumetric data. In *Visualization, 2003. VIS 2003. IEEE*, pages 417–424, Oct 2003.

[33] H.-Y. Wu, M. Rubinstein, E. Shih, J. Guttag, F. Durand, and W. Freeman. Eulerian video magnification for revealing subtle changes in the world. *ACM Trans. Graph.*, 31(4):65:1–65:8, July 2012.

[34] F. Yan, A. Sharf, W. Lin, H. Huang, and B. Chen. Proactive 3d scanning of inaccessible parts. *ACM Transaction on Graphics*, 33(4):1–8, July 2014.