Thesis for the degree of Licentiate of Engineering

Towards accessible content creation of real world objects for virtual environments

Sverker Rasmuson

Division of Computer Engineering
Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2019

Towards accessible content creation of real world objects for virtual environments
Sverker Rasmuson

Technical report number 206L,
ISSN 1652-876X
Department of computer Science and Engineering
Research group: Computer Graphics

Department of computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
Phone: +46(0)32 772 1000

Contact information:
Sverker Rasmuson
Department of computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg, Sweden

Phone: +46(0)72 974 48 47
Email: sverkerr@chalmers.se
URL: http://www.cse.chalmers.se/~sverkerr/

Cover:
A piece of cloth 3D reconstructed from photographs with a tool described in this thesis.

Printed in Sweden
Chalmers Reproservice
Göteborg, Sweden 2019

Towards accessible content creation of real world objects for virtual environments

Sverker Rasmuson
Department of Computer Science and Engineering
Chalmers University of Technology

Thesis for the degree of Licentiate of Engineering
a Swedish degree between M.Sc. and Ph.D.

Abstract

3D reconstruction is the general problem of creating 3D models from real world objects. In today's movie and games industry, there is an increasing demand for using real world content as assets in production. In general, however, 3D reconstruction is a challenging problem, and current techniques only allow for production-ready results given a combination of expensive equipment and specific expertise.

This thesis is a collection of three papers that address various aspects of this general problem of 3D reconstruction, with the aim of lowering the bar for making usable real world content.

In **Paper I**, we address the problem of storing and streaming time varying geometry for e.g. free-viewpoint video, which otherwise has too high bandwidth requirements to be streamed efficiently. We use a memory-efficient structure based on compressed voxels to store the data, in which we can send only incremental updates to the geometry in each frame.

In **Paper II**, we implement an end-to-end real-time pipeline for free-viewpoint video communication. The pipeline uses a set of ordinary webcams as input and do all processing on a single desktop computer. Even with these limitations, we show that we can produce free-viewpoint video with agreeable quality in real-time.

Paper III addresses the problem of accessible and accurate modeling of static real-world objects. Given a set of calibrated input images, we have developed an interactive tool that makes 3D reconstruction with multi-view stereo more accessible. This interactive reconstruction has several advantages over automatic 3D scanning, since we obtain correct topology by design as well as information about visibility and foreground segmentation.

Keywords: voxels, compression, volumetric video, 3D reconstruction

Acknowledgements

I want to thank my supervisor Ulf Assarsson for having the courage to try something new and accept me as a Ph.D student, and for his invaluable guidance in the world of academia and beyond. I also want to thank my co-supervisor Erik Sintorn for continuous hands-on support and always being available for discussions and questions.

I also want to thank my present and former colleagues Dan, Viktor, Markus, Ola, Roc, Alexandra, Nadja, Dmitry, Victor, and everyone else at our division, for creating an enjoyable work environment and always providing moral support.

Finally I want to thank my family, and especially my partner Jonna Hellström, for their love and invaluable support.

List of Appended Papers

This thesis is a summary of three papers.
References to the papers will be made with roman numerals.

Paper I - Viktor Kämpe, Sverker Rasmuson, Markus Billeter, Erik Sintorn, Ulf Assarsson,
Exploiting Coherence In Time-Varying Voxel Data,
I3D '16: Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games.

Paper II - Sverker Rasmuson, Erik Sintorn, Ulf Assarsson,
A Low-Cost, Practical Acquisition and Rendering Pipeline for Real-Time Free-Viewpoint Video Communication,
under revision.

Paper III - Sverker Rasmuson, Erik Sintorn, Ulf Assarsson,
User-Guided 3D Reconstruction Using Multi-View Stereo,
submission pending.

Table of Contents

I Summary

II Appended Papers

Part I
Summary

1 Introduction

Asset production for computer generated graphics, i.e. the creation of 3D models, is one of the most costly and time-consuming parts in any game as well as many movie productions. Each production employs up to several hundreds of artists, who carefully model, animate, and create geometry and textures for each asset, no matter its size, seen in each scene.

The detail, in terms of the number of assets used in a given scene, has increased rapidly over time, and this trend shows no signs of declining. More and more resources are thus required modeling these assets for each new production when wanting to stay in the forefront regarding visual detail.

In many genres, e.g. sports, racing or war, a high level of detail is used for increased realism. It is desirable to have a high number of assets that convincingly mimic the look and behavior of real-world objects. Realistic assets can be modeled by hand. However, an increasing effort has been made to try to reconstruct assets directly from the real world by using e.g. 3D-scanning, motion capture and actual photographs for textures.

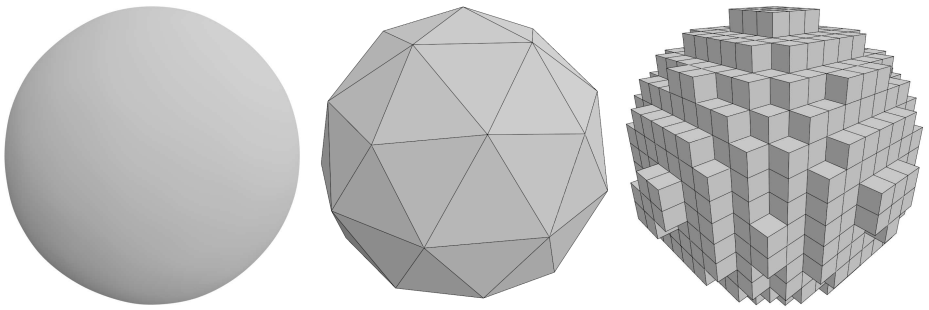
Such techniques have reached a varying degree of maturity, with certain types of motion capture almost being ubiquitous in today's productions, while 3D-scanning only is used in specific cases. In general, however, what they have in common is that they require much expertise, specialized equipment and controlled environments to be used successfully.

In this research, we explore various ways of lowering the bar for creating 3D content of real-world objects. Our aim is to make techniques for this purpose more accessible and making them available for a wider audience. We will start by going through the background of some techniques that has been relevant for the work in this thesis.

1.1 Object representation

Objects in 3D-graphics are commonly modeled as meshes of triangles that represent the surfaces of said objects (see Figure 1). This has a number of advantages, among others that graphics hardware has special functional units to rasterize triangles very fast, which forms the basis of modern real-time graphics [2].

Another way of modeling objects, that is closely related to triangles, is to use quadliterals (quads), i.e. a primitive made up of four vertices instead of three. This is a common primitive used when modeling objects, since it makes it easier to preserve important lines and to later apply subdivision algorithms to the mesh. Quads also allows for easier animation and for selecting loops of primitives (see Figure 2).



(a) Original sphere **(b)** Sphere represented by triangles **(c)** Sphere represented by voxels

Figure 1

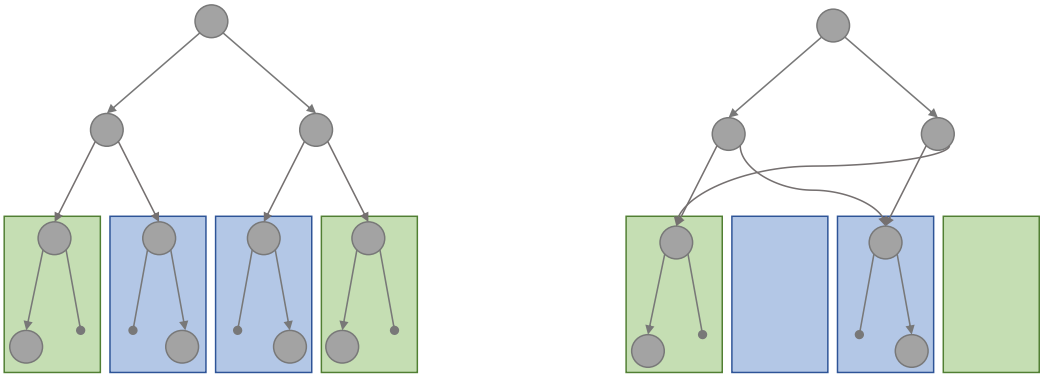
(a) **(b)** **(c)**

Figure 2: A model of cloth represented by quads with increasing levels of tessellation (left to right).

A different approach is to use a volume element instead of a surface element. These elements are commonly named voxels, an abbreviation for volume element. As the name implies, similarly how grids of pixels are used to represent images, 3D-grids of voxels can be used to represent geometry. In its most naive form, a dense grid with a resolution of $W \times D \times H$ encapsulates the object or scene. Each voxel then stores either a 1 or 0, where 1 indicates geometry and 0 indicates empty space. For high resolutions, the memory consumption of this representation soon grows unwieldy. For example, a $1024 \times 1024 \times 1024$ cube and 1 bit per voxel would require 128MB of memory.

A more feasible approach is to use some kind of sparse structure that effectively only store detailed information where there actually is geometry. With this approach, only 1:s along the surfaces of objects are stored. A

common data structure for this is a Sparse Voxel Octree (SVO) [13]. It hierarchically divides each cube into eight smaller cubes, facilitating simple rules for storing and traversing nodes. The structure is sparse, meaning that only sub-nodes that actually have any geometry are stored (see Figure 3).



(a) Sparse Voxel Octree **(b)** Directed Acyclic Graph

Figure 3: The Sparse Voxel Octree in a) is a hierarchical structure that only stores voxels where there is geometry. The Directed Acyclic Graph in b) is a compact version of this structure, where identical subtrees are merged and replaced by a pointer to its first occurrence.

A way of storing even less information, is to compress this SVO into a Sparse Voxel Directed Acyclic Graph (DAG) [11]. This works by identifying identical subtrees of the SVO and then replacing all duplicates of that subtree by pointing to just one of the instances and removing the others (see Figure 3). This can achieve very high compression ratios, sometimes with as low memory requirements as for instance 0.08 bits per non-empty voxel, compared to more than 1 bit per voxel for SVOs [11].

1.2 Depth Cameras

One popular way of acquiring 3D data from the real world is to use a *depth camera*. This is a video camera in the sense that it records images of a scene at interactive to real-time frame rates. However, not regular RGB images, but instead depth maps, where each pixel represent the depth to the closest surface in front of the camera.

Since the arrival of the Microsoft Kinect in 2010 [21], which originally was designed as a motion sensing device for Xbox 360 games, sensors have become a lot more affordable and accessible for the wider research community. Many other devices have appeared on the market since then, typically with a price tag around 100 euros. As a consequence, several impactful papers have

been published using depth cameras, e.g. the Kinect Fusion papers [10][14], which compute very convincing watertight representation of objects using a volumetric representation of signed distance fields.

There are two main technologies devices use: structured light and time-of-flight. Examples are for instance the original Kinect, which used structured light [21], and the KinectV2, which used a time-of-flight sensor [17].

Structured light sensors work by projecting a known IR-pattern on a scene and then computing the depth by analyzing how this pattern is deformed by surfaces. This technique uses IR-cameras for reconstruction and software for computing the actual depth. These sensors can give accurate results, albeit for limited resolution and frame rates.

Time-of-flight uses a more specialized type of hardware and work by emitting a light pulse with a fixed frequency and then measuring how long time it takes (the time-of-flight) for the light to reflect back towards the camera, for each pixel [8]. Time-of-flight cameras typically have high frame rates but suffer from problems of *multi-path interference*, which means that reflected light rays have had more than one bounce, leading to the sensor computing an incorrect depth at for example corners.

Using depth cameras is in many ways very compelling (and nowadays affordable), but each technique comes with its own set of limitations as just described, and the depth maps are typically noisy. These techniques are also non-trivial to scale up to multiple devices, for example to cover a scene from multiple view angles. Structured light cameras have problems with interference between the projected pattern from different devices. Time-of-flight cameras have a similar problems with interference between devices when the same frequency of the light pulse is used, and special care needs to be taken to make them cooperate nicely with each other.

1.3 Camera Calibration

To be able to use ordinary RGB cameras for 3D reconstruction, as described in subsection 1.4, the cameras first need to be *calibrated*. Camera calibration is often divided into two parts, intrinsic calibration and extrinsic calibration.

Intrinsic calibration refers to the internal parameters of the camera, such as field of view and center of projection. A camera is typically modeled as a simple perspective camera, with zero or more distortion parameters to account for non-linearities and defects in the camera system. Note that these can change given the current configuration of the camera so that e.g. current focus and zoom settings can affect the calibration parameters. The intrinsic calibration is typically represented as a 3x3 matrix with an optional array of distortion parameters, depending to what accuracy the camera is being

modeled.

The extrinsic parameters refer to the 3D position and orientation of the camera in a world-coordinate system. This is typically represented using a 4×4 transformation matrix.

In computer graphics, these two parts can be equivalently represented using a 4x4 projection matrix P (with a given near and far plane) and a 4×4 view matrix V, disregarding the optional distortion parameters. The final transformation from world to clip space is then PV.

Camera calibration is a well understood theoretically but can be challenging to achieve with high quality in practice. This is especially true if consumer or other low-end cameras are used, for which the camera model is probably less accurate.

When designing the calibration strategy, there typically is a conflict between how accurate the camera model needs to be and the practical problem of collecting the amount of data needed to determine said parameters to some degree of precision. With a more complex model, more data is needed to determine the added number of parameters accurately. Therefore, a fairly simple camera model is often used, since the system will probably need recalibration from time to time when something is moved or some internal parameters in the cameras are changed (by for example zooming). With such a model, one can hopefully achieve fairly confident intrinsic and extrinsic parameters for the calibration. On the other hand, if the model cannot approximate the behavior of the lens system sufficiently accurate, it might be impossible to capture the camera's behavior with any precision regardless.

1.4 Stereo Reconstruction

Given two images of an object, known intrinsic and extrinsic camera parameters, and at least some overlap in the visibility of the object between the two images, the 3D-position of the object can be computed with regards to the cameras by using *triangulation* (see Figure 4). Given a pixel in the first camera, and the corresponding pixel or sub-pixel position (that sees the same part of the object) in the second camera, the intersection of two rays shooting through these pixels will produce the correct 3D position [19].

The problem of stereo reconstruction is thus mainly the problem of finding these *pixel correspondences*.

So, under what circumstances is it feasible to find such correspondences, i.e., when will a pixel or region of pixels look the same for two cameras?

One basic assumption to be able to find such correspondences is that the surface of the object is diffuse. This means that the BRDF is roughly constant, i.e. that incoming light to the surface is equally scattered in all

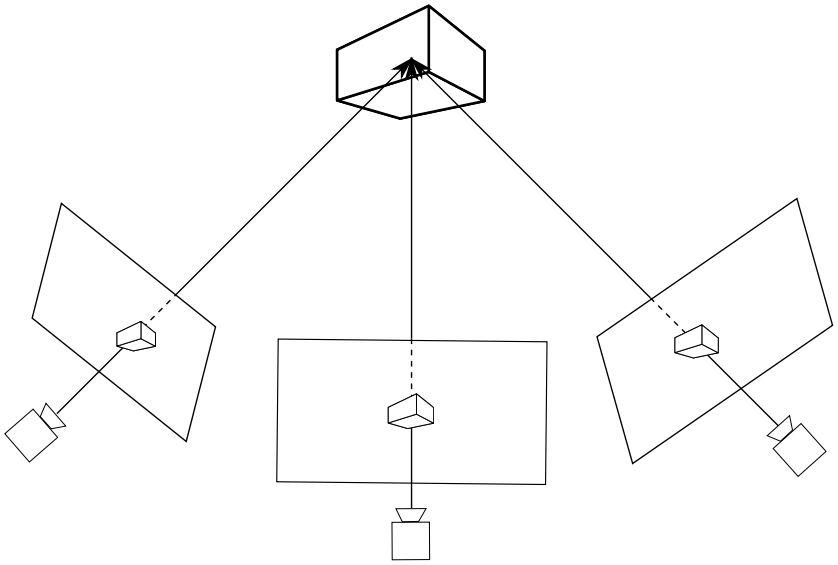


Figure 4: A position in world space can be computed by triangulating its position using two or more cameras.

directions on the hemisphere. This is important, because the surface has to be depicted in a similar manner in at least some aspects onto both cameras in order to identify the two regions as a match and thus representing the same three-dimensional position. Such aspects may include the colors, gradients, or other measures.

In reality, no surfaces are fully diffuse. However, in practice, it is usually fine with a little view-dependence, as long as the cameras are close enough together that the difference in the used measure is small.

One other fundamental limitation is that it is only possible to find per-pixel correspondences in areas where the measure can find a locally unique match, i.e. textured areas. E.g, if you have a uniform region of the same color, it is impossible to tell which of those identically colored pixels that are supposed to match between the cameras.

Another important factor to consider is the connection between depth resolution, the angle between the cameras, and the camera resolution. An easy way to visualize this is the case where a ray is shot through one of the pixels of the first camera. This ray is then projected onto the image plane of the second camera, producing a line that streaks over the image. If this line is followed one pixel at a time, and in each step projected back onto the ray sent from the first camera, a discrete set of depths are found (see Figure 5).

The effective depth resolution is thus dependent on both the resolution of the two cameras, as well as upon the angle between the two rays used

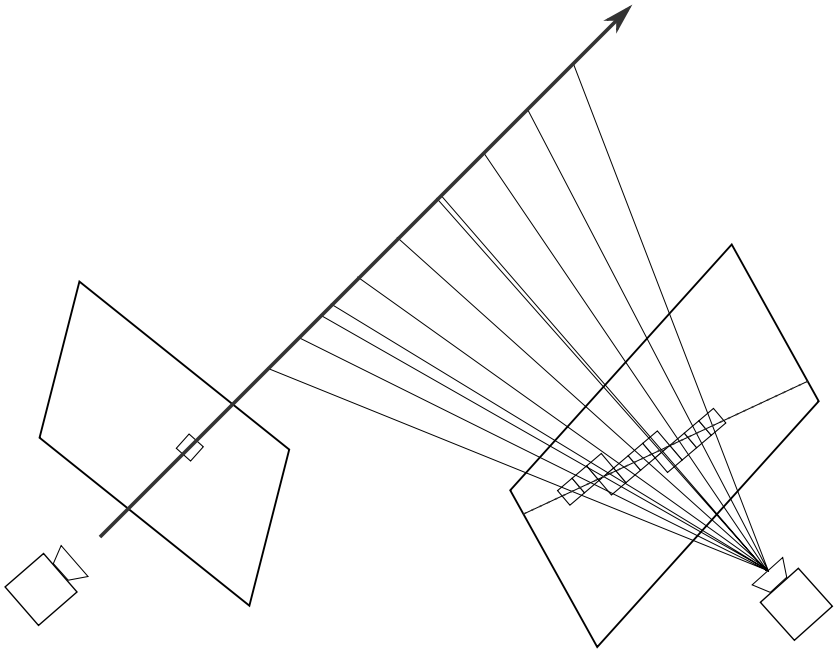


Figure 5: The depth resolution of a stereo reconstruction is dependent of the angle between the cameras and the resolution of each of the two cameras.

for triangulation. With a higher angle and/or longer distance between the cameras, the higher the depth resolution.

In practice, there is therefore often a conflict between depth resolution, which increases with higher angle between the cameras, and view-dependent effects, which also increases with a higher angle but that needs to be low to allow for pixel correspondences to be found.

1.5 Volumetric Video

Volumetric video, sometimes called free-viewpoint video, is a video format that instead of having an *image* per frame, has a full *3D scene* per frame. The actual implementation does not have to naively store scene contents for each frame; it could use e.g. voxels, key-framed meshes or some other more efficient representation. However, from the perspective of the viewer the experience should be the same. In practice, this means that the user controls the virtual camera during playback of the video, e.g. using a head-mounted display, a mouse and keyboard or a gamepad. This could be compared to 360° video, where the user is also free to change the camera orientation but cannot move its position. In volumetric video, there are no such restrictions

and the camera can move about freely.

In recent years, there have been several successful attempts in recording volumetric video using a combination of techniques from multi-view stereo and surface reconstruction [6][7]. These techniques have also been extended to 3D video conferencing [15]. Common for these methods, however, is that they require specialized setups and expensive equipment.

Part of the work in this thesis is to make volumetric-video recording more accessible, with less complicated setups.

2 Summary of Included Papers

In this Section, we will describe the main contributions of each paper:

Paper I introduces a novel way to compress time-varying voxel data by exploiting coherence in time and space.

Paper II presents a end-to-end system from camera to screen of real-time volumetric video of dynamic scenes.

Paper III proposes an interactive tool simplifying the creation of real-world 3D objects from a set of images.

2.1 Paper I - Exploiting Coherence in Time-Varying Voxel Data

Problem: Encoding time-varying data from scanned geometry, such as from depth cameras, is challenging because of the sheer volume of data needed to represent each frame, especially for a system scaled to multiple cameras and/or depth cameras. This volume of data is not only problematic to keep in local memory but also becomes challenging to stream over an internet connection due to the high bandwidth requirements.

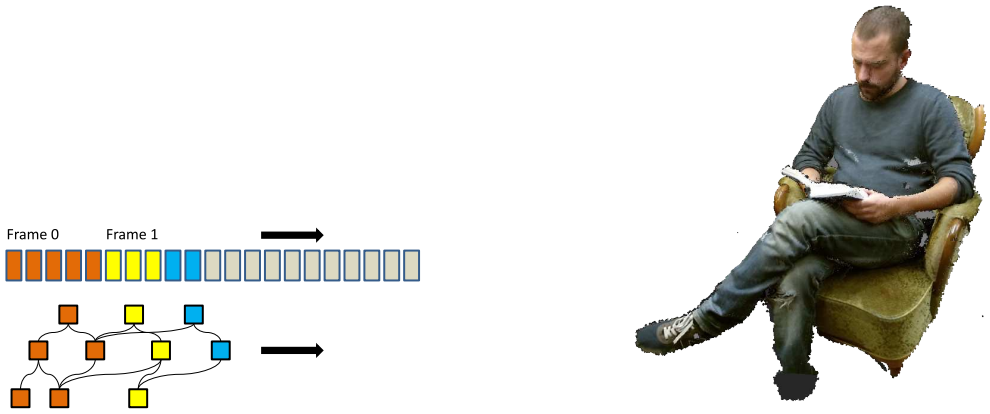
The encoding needs to compress the data in a way that enables playback frame by frame. Decoding and rendering also need to be fast enough to be performed in real time during playback.

This paper investigates exploiting temporal coherence in voxel-based time-varying geometry.

Methodology: The paper builds upon the method of Kämpe et al., where voxelized scenes are compressed using a lossless tree-compression scheme [11]. The scene is first voxelized and represented as a Sparse Voxel Octree (SVO) [13]. This structure is then compressed by iterating through each level of the tree and replacing identical subtrees with pointers to a single instance of each such subtree, creating a Sparse Voxel Directed Acyclic Graph (DAG).

This idea is extended to the time domain by regarding each frame as a standalone SVO and concatenating the trees, level by level, for all frames, searching for identical subtrees in all frames at once, while keeping a root node for each frame (see Figure 6). This creates a data structure that can be traversed frame by frame, but where decoding and rendering is not more expensive than for an ordinary static DAG.

The temporal DAG is compressed further by encoding it as a dense bit-stream using a predetermined traversal order. In this dense bit stream, pointers are encoded implicitly, reducing the final dag size by 2-3 times.



(a) Temporal Directed Acyclic Graph **(b)** Voxelized scene

Figure 6: In a): the structure of the temporal DAG, where each frame has its own root node. Exploiting coherence in time, identical nodes can be found in previous frames and reused. In b): a frame from a scene voxelized from a recording using three Microsoft KinectV2 depth cameras.

Four different scenes were tested at varying resolutions from 512^3 to 2048^3, of which two were recorded and two rendered using virtual cameras.
Contribution: The time domain offers further coherence to be exploited, and the tree-compression algorithm extends well to video voxel data. Both artificial and recorded content exhibits bitrates comparable to e.g. a high quality video from a streaming service, ranging from 2 to 55 Mbits/s.

My contribution in this paper was the recording and processing of the Kinect scene (see Figure 6). The scene was captured with three Microsoft KinectV2 cameras connected to a single computer, using the open source software libfreenect2 [1]. The cameras were calibrated both intrinsically and extrinsically using a checkerboard pattern and the OpenCV library [5][20].

The capture was challenging due to interference between the three time-of-flight cameras. The cameras had to be manually restarted one after another until no interference was observed. Multiple cameras was necessary for this scene to be able to capture it from a wide angle of views. The depth maps captured from the Kinects where then converted to point clouds in a common coordinate system, processed using a median filter and then converted into an SVO per frame using binning.

2.2 Paper II - A low-cost, practical acquisition and rendering pipeline for real-time free-viewpoint video communication

Problem: Free-viewpoint video is a multifaceted problem where each stage of the pipeline; input, 3D reconstruction, processing and rendering; is challenging in its own right. Common issues are robustness, accuracy and the raw processing power needed, especially if real-time applications such as video communication are considered.

Current solutions typically accomplish recording and processing using a controlled environment, specialized equipment, and ample processing power. Controlled lighting setups in combination with synchronized high-end cameras enable accurate multi-view stereo approaches. Specialized equipment enables powerful techniques such as active stereo. Abundant processing power makes it possible to incorporate expensive algorithms for global surface reconstruction [6][7][15][12].

In this paper, we present an end-to-end pipeline from input to final image, with real-time performance and agreeable quality, using only commodity equipment such as web cameras and a single desktop computer.

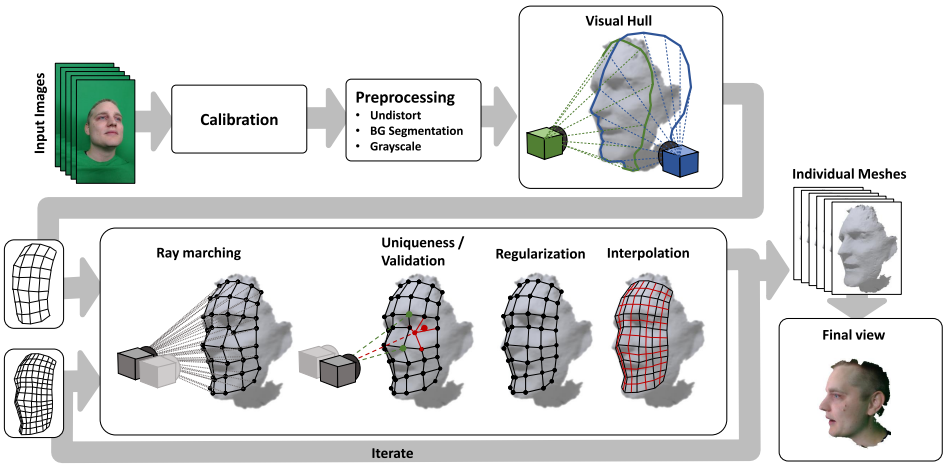


Figure 7: An overview of our real-time pipeline.

Methodology: In this paper, our setup include a green screen, a set of Logitech C922 web cameras, and a single desktop computer using an Nvidia GTX 980 graphics card. An overview of the pipeline can be seen in Figure 7.

The system is first calibrated using an external calibration target. The input images are then processed by undistorting them from calibration parameters, segmenting out the background, and finally is converted to grayscale.

The first step of the algorithm is to compute the visual hull, which provides constraints on the geometry based on the known visibility of each camera (see Figure 7). This is done efficiently using a rasterization-based algorithm and provides the rest of the pipeline with a good initial estimate of where to start searching along each ray.

The main part of our pipeline is then to reconstruct the geometry for each camera pair (e.g. all immediately adjacent cameras) given the preprocessed input images and the visual hull. This algorithm is implemented as a hierarchical stereo-reconstruction algorithm, which produces a good combination of robustness and performance.

The stereo matching starts at a low resolution, using large filter sizes. This results in a coarse oversmoothed model. This model is then used as input to the next stage, where the stereo matching runs at a higher resolution and with smaller filters in an iterative process. During all subsequent iterations, normals are computed to be able to use oriented matching windows in the stereo matching algorithm.

This main algorithm produces a set of overlapping meshes. These meshes are then rendered from the virtual camera, using a weighting scheme to sample colors from the video streams in way that avoids creating seams along overlaps. This weighting is computed using a screen-space algorithm that computes the distance to the border for each mesh. Each sample is then inversely weighted with this distance so that poor geometry along edges are weighted down.

Contribution: In this paper, we create an end-to-end pipeline that achieves real-time volumetric video on a single desktop machine. This allows for a more accessible alternative to record and render volumetric video than what is currently available.

We use an efficient GPU-centric pipeline to achieve high throughput and processing times well below our 30 Hz cameras and also below a potential 60 Hz source. For performance, one significant step is the CUDA implementation of the main reconstruction algorithm, which achieves high thread-level parallelism also for low resolutions in the hierarchy.

Some results from our reconstruction, also compared to using a depth camera, are show in Figure 8.

Figure 8: Novel reconstructed views of three different scenes. The two rightmost images shows a comparison between our method (top) and using a Microsoft KinectV2 (bottom).

2.3 Paper III - User-guided 3D reconstruction using multi-view stereo

Problem: Automatic 3D-reconstruction using multi-view stereo is a well-researched problem [9][16]. The technology has matured enough to be used in production in the game and movie industries. The general pipeline uses either a single camera to capture the object from multiple viewpoints, or a custom-built rig of either a rotating camera or a setup of fixed cameras [18][3].

This technology, however, is still often complicated enough to refrain use by other than experts. Some tools exists, such as RealityCapture or Meshroom, that can create plausible geometry from a set of photographs [4]. In practice, however, these photographs need to be of high quality, and often extensive knowledge and special equipment is needed to achieve desired results. The results will also typically have a high polygon count and arbitrary topology and can be unsuitable for automatic decimation to a low polygon model without manual intervention.

We propose a semi-interactive tool that aims to provide an accessible alternative for 3D-reconstruction of static objects from a set of photographs. Input from the user help to create a favorable topology and alleviate problems with visibility and foreground/background segmentation.

Methodology: In the paper, we present an interface for creating 3D models using quads, given a set of input images.

The user chooses a set of views that have a clear view of the part of the object to be reconstructed. In one of the views, the *reference* view, the user

Figure 9: In the left-most image, the user starts modeling by placing a quad just at the front of the model, which is automatically aligned given two views. In the middle image, further quads have been placed out in the same views. In the right-most image, other views with better visibility have been chosen, signified by a unique color.

starts to build a coarse geometry by creating quads (see Figure 9). Upon completion of each quad, the program runs an optimization algorithm, that helps to align the quad correctly in accordance to the chosen views. This optimization keeps the topology that the user has constructed intact and only moves vertices along the epipolar line in each of the other views.

Upon completion of this coarse topology, a new view set can be chosen to continue on a different part of the object, and this can then be repeated until the whole object has been modeled. Given this coarse model, the user can choose to globally optimize the mesh, optimizing all vertices at once given a joint photometric and smoothness cost function, while still retaining the topology as before. The user is provided tools to subdivide whole or parts of the model and can follow that by further global optimization and/or subdivision.

Typically, the user only have to specify a very coarse geometry manually and can then via subdivision and optimization reach a final model with the desired polygon count.

Contribution: In this paper, we present a tool that aims to be an accessible alternative for creating usable 3D-models of static real-world objects, using images as input. Our tool leverages knowledge of the user, and provides favorable conditions for a global multi-view stereo algorithm to converge, while retaining a desired topology.

The workflow created with this tool could often be an attractive alterna-

tive compared to 3D-scanning follow by re-topologizing.

3 Discussion and Future Work

This thesis focuses on techniques and tools for simplifying content creation of real-world objects and scenes for virtual worlds, an increasingly important ingredient in creating realistic games and movies, as well as creating living worlds for Augmented and Virtual Reality.

Paper I addresses the problem of encoding time-varying geometry in a way that reduces its memory consumption, providing an efficient way to stream content over an internet connection. Paper II describes a real-time pipeline that shows that it is possible to achieve real-time, usable 3D reconstruction using only low-cost consumer equipment. Paper III presents an interactive tool that aims to be an accessible alternative for creating 3D models of static real-world objects from a set of photographs, by keeping the user in the loop.

A major set of challenges for accessible content creation is the problem of reconstructing 3D in arbitrary environments and under arbitrary lighting conditions. For example, if today's streaming video content provides any hints to what future 3D content might look like, a set of diverse environments will be present, from small bedrooms to outdoor scenes and moving cameras. The cameras used could also be of a varied nature, e.g., mobile phones, custom-made synchronized cameras, depth cameras and high-end DSLRs, or some combination of all of these.

Hence, it would be interesting to work on research related to creating robust algorithms and methods to handle this diversity of environments and equipment. Deep neural networks can probably be a useful tool in achieving robustness when handling problems such as imperfect lighting, unsynchronized cameras, and occluded objects. One question that this raises is where to obtain data for efficient training of such algorithms.

Bibliography

[1] libfreenect2. URL https://doi.org/10.5281/zenodo.50641.

[2] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering, Fourth Edition*. A. K. Peters, Ltd., Natick, MA, USA, 4th edition, 2018. ISBN 0134997832, 9781138627000.

[3] Thabo Beeler, Bernd Bickel, Paul Beardsley, Bob Sumner, and Markus Gross. High-quality single-shot capture of facial geometry. *ACM Trans. Graph.*, 29(4):40:1–40:9, July 2010. ISSN 0730-0301. doi: 10.1145/1778765.1778777. URL http://doi.acm.org/10.1145/1778765.1778777.

[4] Simone Bianco, Gianluigi Ciocca, and Davide Marelli. Evaluating the performance of structure from motion pipelines. *Journal of Imaging*, 4(8), 2018. ISSN 2313-433X. doi: 10.3390/jimaging4080098. URL https://www.mdpi.com/2313-433X/4/8/98.

[5] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[6] Alvaro Collet, Ming Chuang, Pat Sweeney, Don Gillett, Dennis Evseev, David Calabrese, Hugues Hoppe, Adam Kirk, and Steve Sullivan. High-quality streamable free-viewpoint video. *ACM Trans. Graph.*, 34(4): 69:1–69:13, July 2015. ISSN 0730-0301. doi: 10.1145/2766945. URL http://doi.acm.org/10.1145/2766945.

[7] Mingsong Dou, Sameh Khamis, Yury Degtyarev, Philip Davidson, Sean Ryan Fanello, Adarsh Kowdle, Sergio Orts Escolano, Christoph Rhemann, David Kim, Jonathan Taylor, Pushmeet Kohli, Vladimir Tankovich, and Shahram Izadi. Fusion4d: Real-time performance capture of challenging scenes. *ACM Trans. Graph.*, 35(4):114:1–114:13, July 2016. ISSN 0730-0301. doi: 10.1145/2897824.2925969. URL http://doi.acm.org/10.1145/2897824.2925969.

[8] Miles Hansard, Seungkyu Lee, Ouk Choi, and Radu Horaud. *Time-of-Flight Cameras: Principles, Methods and Applications*. Springer Publishing Company, Incorporated, 2012. ISBN 1447146573, 9781447146575.

[9] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[10] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 559–568, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0716-1. doi: 10.1145/2047196.2047270. URL `http://doi.acm.org/10.1145/2047196.2047270`.

[11] Viktor Kämpe, Erik Sintorn, and Ulf Assarsson. High resolution sparse voxel dags. *ACM Trans. Graph.*, 32(4):101:1–101:13, July 2013. ISSN 0730-0301. doi: 10.1145/2461912.2462024. URL `http://doi.acm.org/10.1145/2461912.2462024`.

[12] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association. ISBN 3-905673-36-3. URL `http://dl.acm.org/citation.cfm?id=1281957.1281965`.

[13] Samuli Laine and Tero Karras. Efficient sparse voxel octrees. *IEEE transactions on visualization and computer graphics*, 17:1048–59, 10 2010. doi: 10.1109/TVCG.2010.240.

[14] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, Oct 2011. doi: 10.1109/ISMAR.2011.6092378.

[15] Sergio Orts-Escolano, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip L. Davidson, Sameh Khamis, Mingsong Dou, Vladimir Tankovich, Charles Loop, Qin Cai, Philip A. Chou, Sarah Mennicken, Julien Valentin, Vivek Pradeep, Shenlong Wang, Sing Bing Kang, Pushmeet Kohli, Yuliya Lutchyn, Cem Keskin, and Shahram Izadi. Holoportation: Virtual 3d teleportation in real-time. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, pages 741–754, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4189-9. doi: 10.1145/2984511.2984517. URL `http://doi.acm.org/10.1145/2984511.2984517`.

[16] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 519–528, 6 2006. doi: 10.1109/CVPR.2006.19.

[17] J. Sell and P. O'Connor. The xbox one system on a chip and kinect sensor. *IEEE Micro*, 34(2):44–53, Mar 2014. ISSN 1937-4143. doi: 10.1109/MM.2014.9.

[18] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: Exploring photo collections in 3d. *ACM Trans. Graph.*, 25(3):835–846, July 2006. ISSN 0730-0301. doi: 10.1145/1141911.1141964. URL `http://doi.acm.org/10.1145/1141911.1141964`.

[19] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010. ISBN 1848829345, 9781848829343.

[20] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 11 2000. ISSN 1939-3539. doi: 10.1109/34.888718.

[21] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE MultiMedia*, 19:4–12, April 2012.

Part II
Appended Papers