Real-Time Reconstruction of Static and Dynamic Scenes

Der Technischen Fakultät
der Friedrich-Alexander-Universität
Erlangen-Nürnberg
zur
Erlangung des Grades Dr.-Ing.

vorgelegt von

Michael Zollhöfer

aus Herzogenaurach

Als Dissertation genehmigt
von der Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der mündlichen Prüfung:	22.12.2014
Vorsitzende des Promotionsorgans:	Prof. Dr.-Ing. habil. Marion Merklein
Gutachter:	Prof. Dr. Günther Greiner
	Prof. Dr. Christian Theobalt

Abstract

With the release of the Microsoft Xbox 360 Kinect, an affordable real-time RGB-D sensor is now available on the mass market. This makes new techniques and algorithms, which have previously been only available to researchers and enthusiasts, accessible for an everyday use by a broad audience. Applications range from the acquisition of detailed high-quality reconstructions of everyday objects to tracking the complex motions of people. In addition, the captured data can be directly exploited to build virtual reality applications, i.e. virtual mirrors, and can be used for gesture control of devices and motion analysis. To make these applications easy-to-use in our everyday life, they should be intuitive to control and provide feedback at real-time rates. In this dissertation, we present new techniques and algorithms for building three-dimensional representations of arbitrary objects using only a single commodity RGB-D sensor, manually editing the acquired reconstructions and tracking the non-rigid motion of physically deforming objects at real-time rates. We start by proposing the use of a statistical prior to obtain high-quality reconstructions of the human head using only a single low-quality depth frame of a commodity sensor. We extend this approach and obtain even higher quality reconstructions at real-time rates by exploiting all information of a contiguous RGB-D stream and jointly optimizing for shape, albedo and illumination parameters. Thereafter, we show that a moving sensor can be used to obtain super-resolution reconstructions of arbitrary objects at sensor rate by fusing all depth observations. We present strategies that allow us to handle a virtually unlimited reconstruction volume by exploiting a new sparse scene representation in combination with an efficient streaming approach. In addition, we present a handle based deformation paradigm that allows the user to edit the captured geometry, which might consist of millions of polygons, using an interactive and intuitive modeling metaphor. Finally, we demonstrate that the motion of arbitrary non-rigidly deforming physical objects can be tracked at real-time rates using a custom high-quality RGB-D sensor.

Acknowledgements

This dissertation captures the compressed technical contributions and algorithmic achievements of four solid years of work. I started my work in January 2011 with a strong interest in geometry processing, optimization problems, rendering and GPGPU programming, but quite clueless about a suitable field of research. Little did I know that all my rather different interests would magically line up ... a few years later.

This all would not have been possible without the constant support, friendly encouragements and selfless contributions of my colleagues and coworkers. My supervisor Günther Greiner always supported me and gave me the freedom to fully pursue my own research interests. Jochen Süßmuth introduced me to the world of non-rigid registration and supervised me during my studies and in the first year of my PhD. Matthias Nießner helped me to rediscover my will to conduct research and fueled my curiosity in online 3D reconstruction methods. Marc Stamminger helped during all deadlines with his advice and crafted a lot of amazing illustrations. Frank Bauer, our Blender guru, shared his tricks and helped with the production of videos. I am thankful for all this support and feel in great debt. What impressed me most is that all of my colleagues managed to endure my constant rants about the algorithms that break under real world conditions: Christian Siegl, Kai Selgrad, Magdalena Prus, Michael Martinek, Jan Kretschmer, Quirin Meyer, Roberto Grosso, Matteo Colaianni, Matthias Innmann, Henry Schäfer, Benjamin Keinert, Franziska Bertelshofer and Christoph Weber. I feel privileged for the opportunity to work in such a great and creative environment with people that have become more than just colleagues to me over the last four years.

I would also like to thank all of my students that worked with me during their Bachelor and Master projects. Together we learned a lot and got exposed to new interesting topics and ideas. Ezgi Sert helped developing the presented lattice based deformation approach. Justus Thies helped me substantially with my first DFG project and the presented interactive model

based reconstruction method. I am happy that he will continue my research at the chair.

Thanks to Shahram Izadi for giving me the opportunity to conduct research in his group at Microsoft Research Cambridge. I met a lot of cool people during my stay: Andrew Fitzgibbon, Christoph Rhemann, Christopher Zach, David Kim, Cem Keskin and Sean Fanello. It was a thrilling time full of hard work, excellent food and great parties.

Thanks to Matthias Nießner, Matthew Fisher, Chenglei Wu and Christian Theobalt for helping with my real-time non-rigid reconstruction project and Angela Dai for the nice voice over.

I am also grateful to the German Research Foundation (DFG) for funding my work over the last four years under grant STA-662/3--1 and GRK-1773.

Last but not least, thanks to my parents Kerstin and Franz for supporting me on all of my endeavors <3.

October 2014
MZ

Contents

List of Figures

List of Tables

CHAPTER 1
Motivation and Fundamentals

1.1 Motivation

With the release of the Microsoft Xbox 360 Kinect, an affordable real-time RGB-D sensor is now available on the mass market. This makes new techniques and algorithms, which have previously been only available to researchers and enthusiasts, accessible for an everyday use by a broad audience. Such techniques range from the acquisition of detailed high-quality reconstructions of everyday objects to tracking the complex articulated motion of people.

Real-time RGB-D sensors provide a contiguous stream of color and depth data at 30Hz leading to approximately 10 million unique sample points per second. Processing such a huge amount of data at real-time rates requires fast and efficient implementations of the used data structures and underlying algorithms. Real-time speed allows to fully leverage the temporal coherence in the captured data stream, while still providing interactive feedback to the user. This allows the user to directly intervene and positively influence the output of the algorithm adding transparency and increasing the usability of the application. For example, while digitizing an object, the user can directly see the current state of the reconstruction and adapt his scanning pattern to better sample the object's geometry and remove holes.

Nowadays, such commodity RGB-D sensors enable basically everybody to create their own custom three-dimensional content, digitally share the captured datasets with friends and physically replicate the digitized objects using 3D printers. The captured content can be a three-dimensional model of an object's appearance and shape as well as a complete animation sequence that captures the object's time-varying behavior in a compact form. In addition, the broad availability of captured three-dimensional reconstructions

increases the demand for interactive and intuitive deformation approaches that enable users to easily create new variations of the captured geometry. Together these techniques have the potential to redefine the way we think about the creation and manipulation of virtual content.

The captured high-quality three-dimensional models are the basis for taking virtual measurements of our body's anatomy. In the future, this will allow for the design and fabrication of perfectly fitting cloth. Using a virtual mirror, customers will be able to use their virtual double to digitally try-on different types of cloth comfortably from home without having to go to a store. Such an application will enable them to inspect their look from a variety of different virtual viewpoints and under different simulated illumination conditions.

Capturing and tracking the motion of humans in real-time makes it possible to intuitively control arbitrary devices through simple gestures. In addition, the tracked motion can be transfered to captured and handcrafted digital models using re-targeting algorithms. Such a live virtual puppetry system will allow users to breath life into virtual characters in video games and can be used for the production of movies. Leveraging such techniques for teleconferencing will create completely new interaction and communication experiences between people.

1.2 Real-Time RGB-D Sensors

Real-time RGB-D sensors capture a contiguous stream of color and depth data at 30Hz. This enables these devices to sense the appearance as well as the geometry of objects in their field of view. In the following, we illustrate the functionality of such devices at the example of the Microsoft Xbox 360 Kinect sensor. This device was initially released as a video game controller for the Microsoft Xbox. Meanwhile, official drivers have been released which enable the general purpose use of the device on a regular personal computer. Besides this sensor, which was the first one available on the mass market, several other similar devices like the PrimeSense Carmine,

Asus Xtion Pro and the Kinect 4 Windows exist. All of these devices build on the same basic principle to capture depth information, but have slightly different specifications. Most notably is the PrimeSense Carmine 1.09 Short Range sensor that allows to capture depth at a slightly shorter minimum range. We will exploit this property later on for the reconstruction of the shape and appearance of human heads.

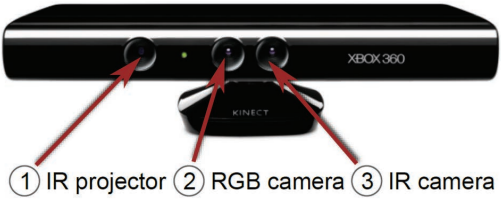
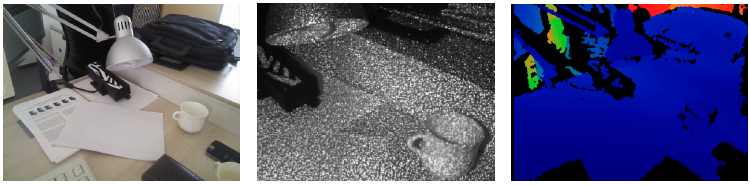


Figure 1.1: The Microsoft Xbox 360 Kinect: An IR projector (1) projects an unique dot pattern into the scene, which is recaptured by an IR camera (3). Depth is estimated based on the local shift between the projected and the captured pattern. The device simultaneously captures color information using an RGB camera (2).

The Microsoft Xbox 360 Kinect combines a regular RGB camera and a depth sensor, consisting of an infrared (IR) projector and an IR camera as shown in Figure 1.1. The IR projector emits an unique dot pattern into the scene which is reflected by objects and directly recaptured (c.p. Figure 1.2 (b)) by the IR camera. The local shift between the emitted and the captured pattern is used to estimate depth. The output is a 640×480 depth image (c.p. Figure 1.2 (c)) approximately every 33.3ms.

Since the Kinect actively illuminates the scene in the IR spectrum to generate artifical features, it belongs to the category of *active* sensing techniques. Compared to *passive* sensing techniques, i.e. passive stereo, the density of the reconstruction does not depend on the amount of texture features that is naturally available in the scene. The Kinect can sense depth approximately in a range between 0.5 and 10 meters. The accuracy of the captured depth values is heavily influenced by the actual distance to the sensed objects and degrades with increasing distance. Therefore, in most of the presented algo-

rithms, we only take depth values which are smaller than four meters into account. Projecting and capturing the dot pattern in the IR domain has



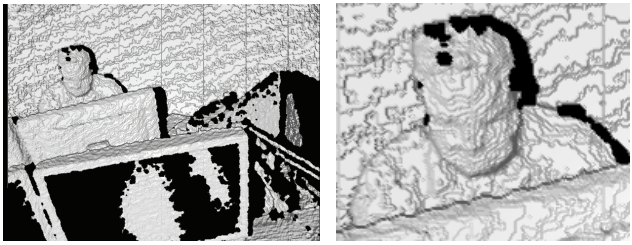
(a) RGB Image (b) IR Image (closeup) (c) Depth Image

Figure 1.2: RGB-D sensors allow to simultaneously sense depth and color information: A color (a) and depth (c) image of a contiguous (30Hz) stream of a Microsoft Xbox 360 Kinect. Depth is color coded from close (blue) to far (red). A closeup of the captured IR image is shown in (b).

the advantage of not interfering with the simultaneous capture of a corresponding RGB image. The additionally captured color data can for example be used to texture the captured geometry. The RBG image (c.p. Figure 1.2 (a)) is provided in the same resolution and frame rate as the depth data. However, the two signals are not naturally aligned due to the different extrinsic positions and orientations of the two sensors. If a mapping between the two signals is required, we have to warp one of them to the coordinate system of the other sensor. The required extrinsic and intrinsic camera parameters can for example be obtained by an off-the-shelf stereo calibration routine [Bra00].

Since depth estimation is based on observing the projected dot pattern, depth can not be recovered if the pattern is partially occluded from the IR camera's point of view. Additional disadvantages of the used technique are data dropouts if the pattern is absorbed (dark objects), reflected (highly specular objects) or transmitted (transparent objects). Problems also arise in bright direct sunlight, since the sun's emitted IR radiation over-saturates the IR camera and makes the pattern imperceptible.

Compared to high-quality structured light 3D scanners, each single captured depth image of the Kinect is of comparably low quality. Figure 1.3



(a) Raw Depth Map (b) Closeup

Figure 1.3: The three-dimensional geometry (a) captured by commodity RGB-D sensors contains a lot of noise and holes. The data is highly discretized (b) and the accuracy degrades with increasing distance to the camera.

shows a phong shaded raw depth map captured by a Kinect sensor. Most notably are the data dropouts at the display and the strong staircase discretization artifacts on the background wall. The advantage of the Kinect sensor is that depth images can be captured at 30Hz. This allows to exploit the spatio-temporal coherence in the input stream to obtain super-resolution reconstructions at real-time rates. Later on, we will also exploit statistical priors to further increase the reconstruction quality.

An imaging sensor, c.p. Figure 1.4, can be completely described by its extrinsic and intrinsic parameters. The extrinsic parameters describe the sensor's spatial location $\mathbf{o}$ and its orientation. An orthogonal coordinate system given by three vectors $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ can be used to specify the orientation. Here, we assume the z-axis to be the optical axis of the sensor and the x- and y-axis to point to the left and upward, respectively. The intrinsic parameters are given by the sensor's projective geometry. It describes the mapping of the 3D world to the 2D image plane (blue) of the sensor, see Figure 1.4 (left). Using the assumption of a *Pinhole Camera*, the sensor's projective geometry can be specified by its focal length f and the principal point $\mathbf{c} = (c_x, c_y)^T$. Geometrically, $\mathbf{c}$ is the intersection point between the optical axis and the imaging plane. In the following, the coordinate system of the image is assumed to be spanned by the vectors $\mathbf{x}_\mathbf{c}$ and $\mathbf{y}_\mathbf{c}$.

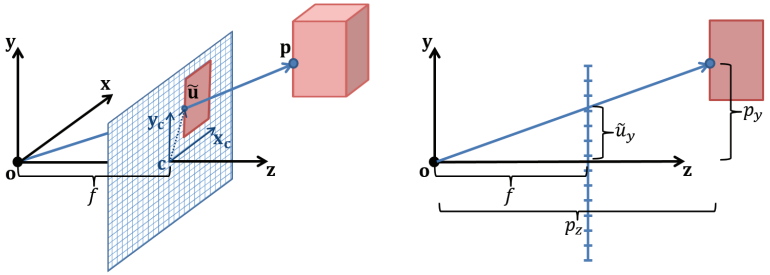


Figure 1.4: The *Pinhole Camera* and its projective geometry: A 3D point $\mathbf{p}$ is projected to the position $\tilde{\mathbf{u}} = (\tilde{u}_x, \tilde{u}_y)^T$ on the 2D image plane using perspective projection. The intrinsic properties of the camera are given by its focal length f and the principal point $\mathbf{c} = (c_x, c_y)^T$.

Let us first assume that the principal point is the origin $\mathbf{c} = (0, 0)^T$ of the image coordinate system. Then, the projection $\tilde{\mathbf{u}} = (\tilde{u}_x, \tilde{u}_y)^T$ on the 2D image plane of a 3D point $\mathbf{p} = (p_x, p_y, p_z)^T$ can be associated with the sensor's intrinsic parameters using equal triangles, see Figure 1.4 (right):

$$\frac{\tilde{u}_x}{f} = \frac{p_x}{p_z}, \; \frac{\tilde{u}_y}{f} = \frac{p_y}{p_z}.$$

Hence, the 2D projection on the image plane $\tilde{\mathbf{u}}$ can be computed as:

$$\tilde{u}_x = \frac{f \cdot p_x}{p_z}, \; \tilde{u}_y = \frac{f \cdot p_y}{p_z}.$$

If the origin of the image coordinate system does not coincide with the principal point, the coordinates of the 2D projection have to be shifted accordingly:

$$\mathbf{u} = \tilde{\mathbf{u}} + \mathbf{c}$$

In homogeneous screen space coordinates $\hat{\mathbf{u}}$, we can directly express these

two equations using linear algebra:

$$\hat{\mathbf{u}} = \mathbf{K}\mathbf{p},\ \mathbf{K} = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}.$$

$\mathbf{K}$ is called the intrinsic camera matrix and models the sensor's intrinsic imaging properties. s is an additional skew parameter that was implicitly set to zero before and can be assumed to be zero for most sensor systems. The intrinsic camera matrix is constant as long as the optical properties of the sensor remain fixed (i.e. the focal length) and can be obtained using off-the-shelf camera calibration routines [Bra00].

To recover the three-dimensional geometry given a depth map, we have to invert the image formation process and back-project the captured depth values to camera space. This requires $\mathbf{K}^{-1}$ which associates a depth map pixel $\mathbf{u} = (u_x, u_y)$ at depth d with its corresponding 3D position $\mathbf{p}$:

$$\mathbf{p} = \mathbf{K}^{-1} \begin{pmatrix} u_x \\ u_y \\ d \end{pmatrix}.$$

The presented *Pinhole Camera* model is a strong simplification of the complex real world optical properties of sensors. To obtain high quality reconstructions, we have to explicitly model and account for lens distortion effects in the image formation process.

1.3 General Purpose GPU Programming

Mainly driven by the increasingly high computational demands of video games over the last decade, Graphics Processing Units (GPUs) have steadily doubled their highly parallel compute power. Nowadays, GPUs easily outperform modern day CPUs in terms of the achieved number of floating point operations (FLOPs) per second. Additionally, GPUs have evolved from a special purpose architecture for the fast and efficient generation of

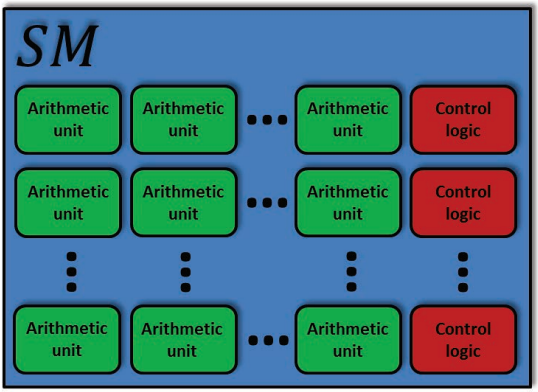


Figure 1.5: Symmetric Multiprocessor (SM): Each SM consists of a set of arithmetic units that execute operations and control logic that allows to schedule the next SIMD operation on multiple cores simultaneously.

synthetic imagery to a highly programmable and flexible compute platform that can be readily used for general purpose programming (GPGPU) tasks. This trend has been started by the CUDA [NVI10] programming architecture. The faster increase in terms of FLOPs compared to CPUs can be explained by the GPU's more compute centric hardware architecture. Following the Single Instruction Multiple Data (SIMD) paradigm, all cores in a warp (CUDA terminology for a collection of 32 cores) of a GPU simultaneously execute the same instruction code on a potentially per core varying set of inputs. This allows all threads executed by the warp to share the control unit that manages the program flow, freeing up more space on the waver for arithmetic processing units. The disadvantage of this paradigm is that operations within a warp have to be sequentialized if the control flow diverges. In this case, the GPU's potentially available peak performance can not be reached. In addition, instead of exploiting clever cache hierarchies to reduce memory latency, GPUs try to hide latency by scheduling between different warps. By getting rid of most parts of the cache hierarchy, additional space on the waver can be allocated to arithmetic units.

In this work, we will exploit the highly parallel compute power of modern GPUs to efficiently solve the presented reconstruction, deformation and inverse rendering problems. This will enable us to solve these challenging problems at a previously unmatched speed and accuracy. Since the GPU's hardware architecture significantly differs from the CPU's, smart new algorithms and techniques are required that are specifically tailored to the underlying hardware platform. In the following, we will give a short overview of the key architectural features of this compute platform.

A modern day GPU, i.e. the Nvidia Geforce GTX Titan, consists of multiple Symmetric Multiprocessors (SMs). Each SM consists of multiple arithmetic units and control logic for scheduling the execution of the next SIMD instruction, see Figure 1.5. In addition, each executed thread has access to three types of memory. These memory types have different specifications and heavily vary in size and response time. To reach the peak performance of the device, it is essential that algorithms are designed such that they exploit fast memory as best as possible.

Global Memory: The global device memory is the largest of the three memory types. It can hold multiple Gigabyte of data and can be accessed by all SMs. Compared to the other memory types, read and write access is time consuming and requires hundreds of clock cycles. This bottleneck is slightly remedied by a small data cache that has a size of approximately 16kB on current GPUs. Nevertheless, data access is expansive and will have a huge impact on the performance of the implemented algorithms. Due to its global nature, this memory region is ideal for inter SM communication.

Shared Memory: The next smaller memory type is the shared memory, it has a default size of 48kB per SM on modern GPUs. It can only be accessed by warps executed on the associated SM. Since this memory is close to the corresponding SM, memory accesses are an order of magnitude faster (tens of cycles) compared to global memory accesses. Due to its local nature, it is ideal for intra SM communication.

Registers: The memory closest to the SMs and therefore the fasted to access are the registers. Besides begin automatically assigned by the compiler to hold frequently used data, data can also be manually assigned. Each SM on a modern GPU has access to a few thousand registers that have to be shared between all running threads. The access time is in the order of a few compute cycles.

Each SM can hold and run more warps than the actual number of physical cores would permit to execute simultaneously. A time-multiplexing strategy is used to hide latencies in memory access by executing a different warp, if a currently running warp has to wait for accessed data elements. The amount of warps a SM can hold concurrently is directly limited by its physical resources. This means that sufficient shared memory and registers have to be available to hold the accumulated state of all the warps.

An additional key factor influencing the performance of GPU programs is the number of issued kernel calls. Each kernel call introduces an implicit synchronization point to the program limiting its parallelism. In addition, kernel calls are not for free, since initializing and launching the threads generates overhead. To avoid this, the algorithm has to be best broken down into a minimal number of kernel invocations. Finding such a smart decomposition of an algorithm requires in-depth knowledge of the problem domain.

CHAPTER 2
Optimization Theory

2.1 Model Fitting

Let us consider a real world system $\mathcal{S}$ that transforms an input vector $\mathbf{s} \in \mathbb{R}^{d_1}$ to a corresponding output vector $\mathbf{t} \in \mathbb{R}^{d_2}$, see Figure 2.1. The mapping $\mathbf{t} = g(\mathbf{s})$ is implemented by an unknown internal function $g : \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_2}$. To capture the behavior of $\mathcal{S}$, we have to discover its hidden internal func-

Figure 2.1: A real world system $\mathcal{S}$ maps an input $\mathbf{s}$ to a corresponding output $\mathbf{t} = g(\mathbf{s})$ based on a hidden unknown function g.

tion and build a digital model. Such a digital model $\mathcal{M}$ is an abstract entity that tries to explain the functional behavior of $\mathcal{S}$ as good as possible based on a different, potentially simpler, function f.

For example, the model used for the weather forecast, tries to predict tomorrow's weather conditions based on a sparse history of measurements. We will exploit such models to predict the appearance and shape of human heads based on a discrete set of color and depth observations by an RGB-D sensor. In addition, we will infer the spatial position and orientation of a moving sensor and track the non-rigid deformations of physically deforming objects based on the captured color and depth constraints. The presented handle based direct manipulation approach infers the pose of a model based on a sparse set of user constraints.

Due to the high complexity and intertwined dependencies of g with respect to its parameters, we have to introduce some simplifications to keep the

model f computationally feasible. These simplifications lead to a disagreement between the predictions and the actual responses of the real system $g(\mathbf{s}) \neq f(\mathbf{s})$. Ideally, we want to find a model that minimizes this discrepancy.

Having observed N outputs $\mathbf{t}_i = g(\mathbf{s}_i)$ of $\mathcal{S}$ for corresponding inputs $\mathbf{s}_i$, the goal of *model fitting* is to find the function f that best explains these observations. In general, a possibly infinite number of functions exist that all satisfy the gathered observations. To make the problem computationally tractable and allow for a unique solution, we have to restrict our search to a suitable subspace that limits the number of degrees of freedom (DoFs). Nevertheless, the used model has to be expressive enough to allow for a good fit of the data and should be fast to evaluate. For simple regression problems, the used subspace might for example be the space of polynomials of degree two. Besides directly reducing the model's DoFs, we can additionally incorporate prior information, i.e. smoothness assumptions to add further restrictions.

Let $f_{\mathbf{x}}$ be a function with n degrees of freedom $\mathbf{x} = \left(x_0, \ldots, x_{n-1}\right)^T$, then the quality of the fit directly depends on these unknown model parameters. To objectively quantify the quality of a fit, we introduce the following objective function that measures whether the model can reproduce the observations (E_{fit}) and fulfills the assumed priors (E_{prior}):

$$E_{total}(\mathbf{x}) = E_{fit}(\mathbf{x}) + \lambda E_{prior}(\mathbf{x}).$$

The parameter λ allows to balance between a tight fit of the observations and the importance of the prior constraints. The value of λ has a drastic influence on the generalization properties of the model and can be used to prevent over-fitting the input data. A suitable value can be empirically determined based on the used model and the expected signal-to-noise ratio or can be statistically learned using cross validation strategies.

E_{fit} can for example be modeled as the squared difference between the com-

puted predictions and the corresponding observations:

$$E_{fit}(\mathbf{x}) = \frac{1}{2}\sum_{i=0}^{N-1} ||f_{\mathbf{x}}(\mathbf{s}_i) - \mathbf{t}_i||_2^2.$$

If a perfect model $f_{\mathbf{x}}$ with respect to the defined objective function and the given training data exists, its parameters $\mathbf{x}$ would fulfill $E_{total}(\mathbf{x}) = 0$. Often, it is neither possible nor desirable to reduce the error to zero, since the number of free variables has to be much smaller than the number of independent constraints. This prevents overfitting and allows for good generalization properties of the model. In this case, we want to find the parameters $\mathbf{x}^*$ such that E_{total} is minimized:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} E_{total}(\mathbf{x}).$$

This is a general unconstrained optimization problem in the unknown model parameters $\mathbf{x}$. In the following, we will focus on a special class of such optimization problems that allows for efficient solution strategies.

2.2 Least Squares Optimization

If the optimization objective $E_{total} : \mathbb{R}^n \to \mathbb{R}$ can be expressed as a sum of m squared residual terms r_i

$$E_{total}(\mathbf{x}) = \frac{1}{2}\sum_{i=0}^{m-1} r_i(\mathbf{x})^2,$$

minimizing E_{total} with respect to its n unknown parameters $\mathbf{x}$ is called a *least squares problem*. Note, that our formulation of E_{fit} in the previous section fulfills this requirement. In the following, we will assume that the optimization problem at hand is over-constrained. This means that the number of unknowns n of our model is smaller than the number of independent constraints. Due to its special form, such optimization problems can be tackled

more efficiently than general energy minimization problems. To see why this is the case, we have to reformulate the associated optimization problem

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \frac{1}{2} \sum_{i=0}^{m-1} r_i(\mathbf{x})^2,$$

in terms of its corresponding residual vector field $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ by stacking all residuals in an m-dimensional vector:

$$F(\mathbf{x}) = \big(r_0(\mathbf{x}), \ldots, r_{m-1}(\mathbf{x})\big)^T.$$

Using this definition, the minimization problem can be reformulated in terms of the vector field F:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \frac{1}{2} ||F(\mathbf{x})||_2^2.$$

As we know from variational calculus, the minimizer $\mathbf{x}^*$ of E_{total} has a vanishing gradient:

$$\nabla E_{total}(\mathbf{x}^*) = 0.$$

Using the *Chain Rule*, the gradient of E_{total} can be expressed in terms of the residual vector field F and its Jacobian $\mathbf{J}$:

$$\nabla E_{total}(\mathbf{x}) = \mathbf{J}(\mathbf{x})^T F(\mathbf{x}).$$

The Jacobian is the $(m \times n)$-matrix of the first partial derivatives of F:

$$\mathbf{J}(\mathbf{x}) = \begin{pmatrix} \frac{\delta r_0}{\delta x_0}, \ldots, \frac{\delta r_0}{\delta x_{n-1}} \\ \vdots \quad \ddots \quad \vdots \\ \frac{\delta r_{m-1}}{\delta x_0}, \ldots, \frac{\delta r_{m-1}}{\delta x_{n-1}} \end{pmatrix}.$$

If the residual functions are linear in the unknowns, we speak of a *linear least squares problem*, otherwise the optimization problem is called a *nonlinear least squares problem*. In the following, we will discuss robust solution strategies for these two special cases and show how they can be solved efficiently.

2.3 Linear Least Squares Optimization

If the optimization objective is a quadratic function, which means that the actual model is linear in the unknown parameters $\mathbf{x}$, the optimization problem is called a *linear least squares problem*. In this special case, all residual functions $r_i(\mathbf{x})$ are linear and the residual vector field F can be expressed as:

$$F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}.$$

The $(m \times n)$-matrix $\mathbf{A}$ gathers all linear coefficients and has one row for each residual and one column for each unknown. The vector $\mathbf{b}$ contains the constant components of the residuals. Therefore, the associated minimization problem can be stated in the following canonical form:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} ||\mathbf{A}\mathbf{x} + \mathbf{b}||_2^2.$$

The Jacobian of the vector field $F(\mathbf{x})$ is its first derivative:

$$\mathbf{J}(\mathbf{x}) = \frac{\delta F}{\delta \mathbf{x}} = \mathbf{A}.$$

In this special case, the minimizer $\mathbf{x}^*$ of the energy function can be computed as:

$$\nabla E_{total}(\mathbf{x}^*) = \mathbf{A}^T(\mathbf{A}\mathbf{x}^* + \mathbf{b}) = 0.$$

The resulting system of linear equations are the well known *normal equations*:

$$\mathbf{A}^T\mathbf{A}\mathbf{x}^* = -\mathbf{A}^T\mathbf{b}.$$

The system matrix $\mathbf{A}^T\mathbf{A}$ is symmetric and the system allows for the computation of a unique solution. Since the objective is convex and always positive, we can see that the solution of this linear system is also the global minimizer of E_{total}. The symmetry of the system matrix can be exploited to increase the space and time efficiency of the solution strategy. The linear system can be solved with standard factorization based methods or iterative solution techniques. While factorization techniques allow for a direct solution of the system, they are computationally expensive and hard to par-

allelize. In contrast, iterative techniques allow for a better parallelization, but require a good initial estimate to warm start the optimizer. Later on, we will exploit direct solution techniques on the CPU for small dense systems as well as a fast GPU based iterative Preconditioned Conjugate Gradient (PCG) solver to compute $\mathbf{x}^*$ for large sparse systems at real-time rates.

2.4 Non-linear Least Squares Optimization

If the residual vector field is non-linear in its parameters, finding optimal values $\mathbf{x}^*$ that minimize E_{total} is called a *non-linear least squares problem*:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \frac{1}{2}||F(\mathbf{x})||_2^2.$$

In this case, we can compute an optimal solution using for example the *Gauss-Newton* method. The key idea of this approach is to explicitly linearize the vector field F around an estimate $\mathbf{x}^{(k)}$ of the solution using *Taylor Expansion*:

$$F(\mathbf{x}^{(k)} + \delta) \approx F(\mathbf{x}^{(k)}) + \mathbf{J}(\mathbf{x}^{(k)})\delta,\ \delta = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}.$$

This linear approximation of the multi-variate residual vector field gives rise to a *linear least squares problem* in the unknown optimal updates δ^*:

$$\delta^* = \arg\min_{\delta} ||F(\mathbf{x}^{(k)}) + \mathbf{J}(\mathbf{x}^{(k)})\delta||_2^2.$$

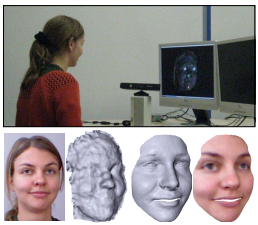
This problems can be efficiently and robustly solved as described in Section 2.3. Given a suitable starting value $\mathbf{x}^{(0)}$, the minimizer of the non-linear optimization problem can be iteratively computed by solving a series of E linearized subproblems for the unknowns $\{\mathbf{x}^{(k)}\}_{k=1}^{E}$. The starting value $\mathbf{x}^{(0)}$ has an enormous impact on the convergence of the method and the found optimum. Since the *Gauss Newton* method is locally convergent, it can not be guaranteed that the globally best solution is found. Therefore, smart strategies for initializing close to the global optimum are required. We will

see such strategies, specifically tailored to the problem at hand, later on. Note, that the system matrix is changing in each *Gauss Newton* iteration step, since the system is linearized around the last estimate. This makes it challenging to run this method at real-time rates using direct solution techniques, since a factorization of the system matrix can not readily be reused in multiple iteration steps. Later on, we will present a fast GPU based Gauss-Newton solver that builds on an iterative solution strategy and can compute the optimum at real-time rates.

CHAPTER 3
Contribution and Outline

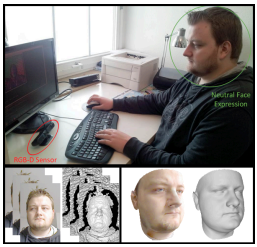
In this dissertation, we first present an algorithms that can reconstruct the geometry of a human face based on a single RGB-D image. After that, we extend this approach and obtain even higher quality reconstructions of the geometry and appearance of human heads based on a contiguous RGB-D stream. Next, we present an approach to digitize complete large scale environments in real-time using a moving consumer-grade RGB-D sensor. The acquired three-dimensional models, which can consist of millions of polygons, can be interactively and intuitively manipulated using the handle based deformation metaphor that is presented next. Finally, we present a real-time method that allows to track the non-rigid motion of physically deforming objects using only a single custom RGB-D sensor. In the following, we give a more detailed overview of the five presented techniques.

Part 1: Reconstruction of Personalized Avatars



We present a simple algorithm for computing a high quality personalized avatar from a single color image and the corresponding depth map which have been captured by Microsoft's Kinect sensor. Due to the low market price of our hardware setup, 3D face scanning becomes feasible for home use. The proposed algorithm combines the advantages of robust non-rigid registration and fitting of a morphable face model. We obtain a high quality reconstruction of the facial geometry and texture along with one-to-one correspondences with our generic face model. This representation allows for a wide range of further applications such as facial animation and manipulation. Our algorithm has proven to be very robust. Since it does not require any user interaction, even non-expert users can easily create their own personalized avatars.

Part 2: Interactive Reconstruction of the Human Head



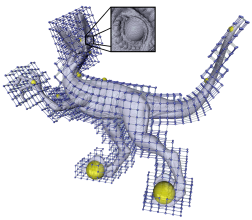
We present a novel method for the interactive markerless reconstruction of human heads using a single commodity RGB-D sensor. Our entire reconstruction pipeline is implemented on the GPU and allows to obtain high-quality reconstructions of the human head using an interactive and intuitive reconstruction paradigm. The core of our method is a fast GPU-based non-linear Quasi-Newton solver that allows us to leverage all information of the RGB-D stream and fit a statistical head model to the observations at interactive frame rates. By jointly solving for shape, albedo and illumination parameters, we are able to reconstruct high-quality models including illumination corrected textures. All obtained reconstructions have a common topology and can be directly used as assets for games, films and various virtual reality applications. We show motion re-targeting, re-texturing and re-lighting examples. The accuracy of the presented algorithm is evaluated by a comparison against ground truth data.

Part 3: Online 3D Reconstruction at Scale

Online 3D reconstruction is gaining newfound interest due to the availability of real-time consumer depth cameras. The basic problem takes live overlapping depth maps as input and incrementally fuses these into a single 3D model. This is challenging particularly when real-time performance is desired without trading quality or scale. We contribute an online system for large and fine scale volumetric reconstruction based on a memory and speed efficient data structure. Our system uses a simple spatial hashing scheme that compresses space, and allows for real-time access and updates of implicit surface data, without the need for a regular or hierarchical grid data struc-

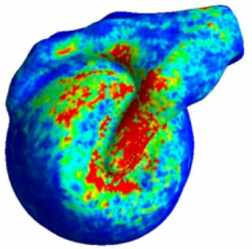
ture. Surface data is only stored densely where measurements are observed. Additionally, data can be streamed efficiently in or out of the hash table, allowing for further scalability during sensor motion. We show interactive reconstructions of a variety of scenes, reconstructing both fine-grained details and large scale environments. We illustrate how all parts of our pipeline from depth map pre-processing, camera pose estimation, depth map fusion, and surface rendering are performed at real-time rates on commodity graphics hardware. We conclude with a comparison to current state-of-the-art online systems, illustrating improved performance and reconstruction quality.

Part 4: Interactive Lattice based Deformation



We present a novel lattice based direct manipulation paradigm (LARAP) for mesh editing that decouples the runtime complexity from the mesh's geometric complexity. Since our non-linear optimization is based on the as-rigid-as-possible (ARAP) paradigm, it is very fast and can be easily implemented. Our proxy geometry automatically introduces volume-awareness into the optimization prob-lem, leading to more natural deformations. Since we compute how the space surrounding an object has to be deformed to satisfy a set of user-constraints, we can even handle models with disconnected parts. We an-alyze the bottlenecks of the presented approach and propose a data-parallel multi-resolution implementation on the GPU, which allows to pose even high-quality meshes consisting of millions of triangles in real-time.

Part 5: Real-Time Tracking of Deforming Objects



We present a combined hardware and software solution for markerless reconstruction of non-rigidly deforming physical objects with arbitrary shape in *real-time*. Our system uses a single self-contained stereo camera unit built from off-the-shelf components and consumer graphics hardware to generate spatio-temporally coherent 3D models at 30Hz. A new stereo matching algorithm estimates real-time RGB-D data. We start by scanning a smooth template model of the subject as they move rigidly. This geometric surface prior avoids strong scene assumptions, such as a kinematic human skeleton or a parametric shape model. Next, a novel GPU pipeline performs non-rigid registration of live RGB-D data to the smooth template using an extended non-linear as-rigid-as-possible (ARAP) framework. High-frequency details are fused onto the final mesh using a linear deformation model. The system is an order of magnitude faster than state-of-the-art methods, while matching the quality and robustness of many offline algorithms. We show precise real-time reconstructions of diverse scenes, including: large deformations of users' heads, hands, and upper bodies; fine-scale wrinkles and folds of skin and clothing; and non-rigid interactions performed by users on flexible objects such as toys. We demonstrate how acquired models can be used for many interactive scenarios, including re-texturing, online performance capture and preview, and real-time shape and motion re-targeting.

PART I

Reconstruction of Personalized Avatars

CHAPTER 4
Introduction

During the past few years, massive multiplayer online games (*MMOGs*) such as World of Warcraft, Aion or Second Life have gained tremendous attention. In these games, millions of users, each represented by a virtual character -- known as *avatar* -- meet in a three-dimensional virtual world. Since the users should be distinguishable from each other, each user must have a unique avatar. Most MMOGs offer a character editor in which the user can select his avatar from a set of existing models. These avatars can be further customized using simple user interfaces which allow the user to morph between different shapes or to include customized textures or models. As the graphics in computer games become ever more realistic and the focus of the games changes from fantasy scenarios towards real life settings, the demand for photo-realistic avatars, which resemble the users themselves, is constantly increasing. Unfortunately, current in-game avatar editors offer only a limited expressibility, making it almost impossible for a user to create a virtual clone for the online world. Besides their usage in online games and chats, personalized 3D avatars are also important in gadget applications such as aging simulations or digital 3D beauty salons.

We present a fully automatic method for the generation of realistic three-dimensional face models with textures. As input for our algorithm we use a depth scan and a color image that were recorded using the Microsoft Kinect sensor. Due to the low market price and its widespread use, the Microsoft Kinect sensor is ideally suited for recording personalized 3D avatars. Using our system, everyone can automatically digitize his face and create a virtual clone within seconds.

CHAPTER 5
Related Work

The reconstruction of personalized 3D face models has become very popular during the last years. Two different approaches exist for this purpose: algorithms which try to reconstruct the 3D model solely from color images and algorithms which reconstruct the 3D model by fitting a template mesh to a depth scan of the face.

The first class of algorithms reconstructs the facial geometry directly from one or more 2D images. Tang and Huang [TH96] automatically extract salient facial features from a front and profile face image. The detected features are then used to adapt a coarse template mesh. Since this method requires a one-to-one relationship between facial features and the vertices of the template mesh, it can only produce very coarse reconstructions. Blanz and Vetter [BV99] employ an iterative optimization to adapt the parameters of a three-dimensional morphable model by projecting the current morphable model onto the image plane and then minimizing the difference between the pixel colors. To improve the stability of their method, the optimization is performed in a coarse to fine manner. Breuer et al. [BKK*08] use Support Vector Machines to detect the face in a 2D image and then extract facial features using a regression- and classification-based approach. After that, they apply a flip-flop optimization to determine the best-fitting morphable model for the detected features, which is, in turn, used to improve the feature detection and classification. Instead of reconstructing the three-dimensional facial model from features in 2D images, Lee et al. [LMPM05] fit a morphable face model in such a way that the shading of the model is as close to the shading of the input image as possible. Commercially available software tools such as AvMaker, FaceGen or FaceShop also compute a 3D avatar from features in 2D images. These features can either be automatically detected or hand-picked by the user. Common to all methods which reconstruct the 3D face model solely from photographs is that while they can accurately reconstruct the face in feature-rich regions,

the fitting in feature-less regions like the cheek or the forehead is rather poor.

Methods of the second category reconstruct the 3D face by fitting a template mesh to depth scans. Weissenfeld et al. [WSQO05] construct a multi-resolution detail pyramid of the input face scan by successively applying smoothing operations. Using a set of manually selected features, they fit a generic face model to the multi-resolution detail pyramid in a coarse-to-fine manner. Blanz et al. [BSS07] describe an iterative algorithm for fitting a morphable model to a textured depth scan. In each iteration, the current morphable model is projected onto a two-dimensional cylindrical image. An energy term which considers both, the color and the depth value after the projection, is then minimized to get a better-fitting morphable model. Basso and Verri [BV07] approximate the input depth scan by an implicit function and then solve for the parameters of a morphable face model such that the distance between the face and the depth scan is minimized. To improve the convergence of their method, they split the template mesh into four sub-meshes which are fitted independently. The sub-meshes are then smoothly blended to obtain the final reconstruction. Li et al. [LSP08] employ a sophisticated non-linear optimization process to fit a source mesh to a target depth scan. By enforcing local rigidity and global smoothness of the deformation, they obtain high quality registrations given a good initial alignment. Most similar to the proposed algorithm is the work recently published by Kim et al. [KLK*10] since they also use non-rigid registration to fit a common template mesh to a depth scan. However, our algorithm is more robust in practice since we do not rely on robust 2D facial features during the non-rigid registration and since our regularization term tries to maintain surface features while their regularization term only minimizes surface stretch.

CHAPTER 6
Method

6.1 Overview

The proposed algorithm automatically reconstructs a high quality 3D face model with texture from an RGB image and a depth map by fitting a morphable face model to the input data. We use the Microsoft Kinect sensor to capture an RGB image and the corresponding depth map from which a metric point cloud is computed. Using four automatically detected feature points, we estimate an initial rigid alignment of a common template mesh with the recorded point cloud. Since this alignment is in general not good enough to be used as input for the morphable model fitting, we non-rigidly register the average face of the morphable model with the input scan. Finally, we use the morphable model to compute the face which approximates the deformed template face best. The result of the proposed algorithm is a high-quality 3D model of the scanned face which has one-to-one correspondences to the faces in the morphable model. Thus, it can be easily analyzed, animated or modified. To add more realism, we compute a corresponding texture for the faces using the captured RGB data.

6.2 Data Acquisition

In this section, we briefly describe the way to obtain a raw depth image of the face of a person sitting in front of the Kinect and the augmentation of this image with valuable feature points as well as color information.

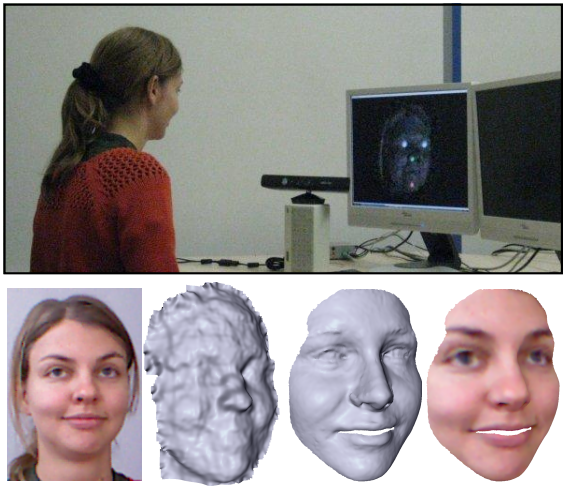


Figure 6.1: Overview of the proposed avatar reconstruction. The upper image shows our acquisition setup in action. The lower row shows the recorded color image and depth scan and the fitted mesh without and with textures.

6.2.1 Data Preparation

To assist the calibration of the IR and the RGB camera, we use OpenCV's calibration routines to specify the required DOFs. After the calibration, the data is available as 3D point cloud with natural metric units and a corresponding RGB value is assigned to each point. Yet, we still maintain the topology information in form of the 640x480 grid from the raw data to simplify the feature detection and face segmentation, which requires neighborhood information for each point.

The depth data we recieve from the Kinect in a single frame is quite noisy and can contain holes at arbitrary positions. To reduce these effects, we average the depth values of eight successive frames, which leads to a much smoother result. Also, missing points in one frame may be compensated by other frames in which the device was able to capture these points. In addition, we apply a Gauss filter and simple hole-filling to the temporally

smoothed data, which further improves the input data. Figure 6.2 shows a face scan at different stages of the preparation pipeline.

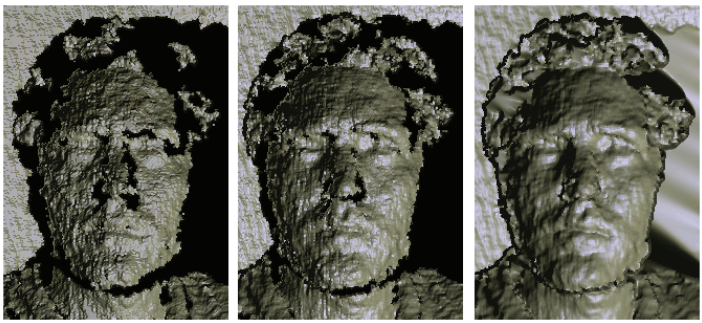


Figure 6.2: Data preprocessing: Left: raw data from a single frame. Middle: temporally smoothed data. Right: additionally Gauss-filtered data

To avoid motion blurring artifacts, it is required that the user remains motionless for at least eight frames (0.27s) while capturing the image.

6.2.2 Face and Feature Detection

Assuming that exactly one person is sitting in front of the Kinect, we first detect the face along with a number of significant feature points to assist the alignment of the scanned data at a later stage. We particularly aim to detect the middle point of each eye, the nose tip and the chin. Our idea is to locate regions of significant face parts in the RGB image, then map these regions onto the geometry data and to proceed the detection of feature points in the geometry domain.

For the first part, we use OpenCV to find the bounding rectangles of the face, the eyes and the nose in the RGB image (see Figure 6.3). After we have detected the face on the entire range of the input RGB image, we reduce the further searching domain in the image to the face's bounding box and detect the eyes and the nose. This reduction does not only provide better performance, but also increases robustness since the algorithm might

falsely identify background parts as an eye or a nose.

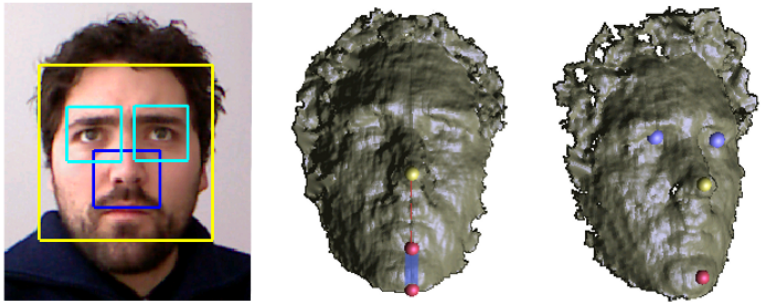


Figure 6.3: Facial feature detection: Left: face-, nose- and eye-detection on the RGB image. Middle: temporary chin features (red) and search domain for the final chin feature (blue area). Right: final feature points.

Due to the fact that the RGB data is aligned with the point cloud and that the topology is the same in both spaces, we can apply the rectangular regions directly in the geometry domain.

In order to find the nose tip, we loop through all 3D points inside the detected nose rectangle and pick the one with the smallest depth value as nose feature.

For the chin, no feature detectors are available in OpenCV which allow for a pre-selection of the chin's region in the RGB image. As a solution, we use the following heuristic: We first perform a line search from the detected nose tip down the y-axis until we find a significant ascend of the depth values. The resulting point is a temporary feature point which we call the chin edge. From this point, we sample the same line back to the nose until we detect a local maximum of the depth values. This second temporary feature point is named the chin groove. We now define the final chin feature to be the point between the chin groove and the chin edge which is closest to the camera. In order to compensate for a small roll of the subject's head (if the main face axis is not perfectly aligned with the y-axis), we extend the search region by a small offset of ± 5 pixels in x-direction.

An eye feature would ideally be the point on the eyeball closest to the cam-

era. However, the resolution and the quality of the input data is not sufficient to robustly find these points on the geometry. As an approximation, we take the center point of the detected bounding box which turned out to be a robust estimate for the initial alignment at a later stage. Figure 6.3 (right) shows the final feature points on a scanned face.

Since the feature detection runs in real-time, a person sitting in front of the camera gets real-time feedback of the resulting face scan together with the detected feature points (see Figure 27.1). This allows the user to optimize his position before he eventually captures the current data. To further stabilize the features, we perform a smoothing operation over various frames as we did on the raw depth data.

6.2.3 Face Segmentation

The recorded depth data still contains unnecessary background data at this point. In order to reduce the costs in the next steps, we separate the face from the rest of the input data as seen in the scanned images from Figure 6.3. From the feature detection, we already know some points which definitely belong to the face. The rest of the points is found with a floodfill-like algorithm using the detected face feature points as seed. In each recursion, we check the four-neighborhood of a current face point whether the depth values change by more than 5mm. If this is not the case, we add the corresponding point to the list of face points and recursively call the routine with this point. This method has turned out to robustly separate the face from the background and the body of the user.

6.3 Fitting a Generic Face Model

The point cloud that was generated in the previous step represents the geometry of the scanned face. However, except for the few detected feature points, the geometry does not contain any semantic information which would be necessary to animate or further process the face. To attribute the scanned face with a semantic meaning, we fit a morphable model [BV99] to

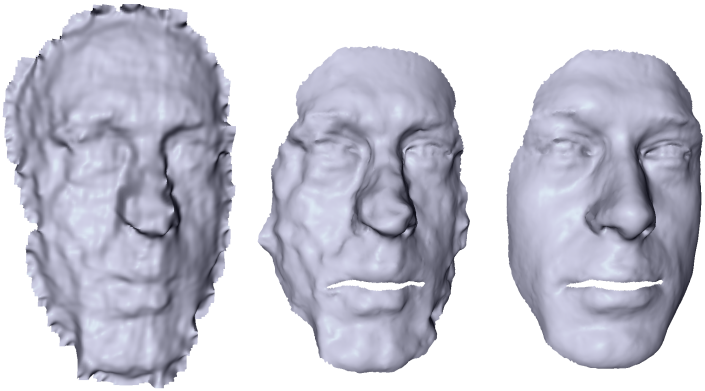


Figure 6.4: Generic face fitting: (from left to right) input scan, registered average face and best fitting morphable model.

the scan. Therefore, we compute a rough initial alignment of the scanned point cloud $\mathcal{P}$ and the average face $\mathcal{T}$ of our morphable model. Given the previously computed features points and the corresponding points on the generic face model, we can compute the shape-preserving transformation which best aligns the two data sets by performing a generalized Procrustes analysis [Gow75].

Theoretically, one could now solve for the parameters of the morphable model in such a way that it approximates the input scan as good as possible. Unfortunately, morphable models are not invariant under shape-preserving transformations which means that the input shape must be perfectly scaled and aligned with the morphable model to generate satisfactory results. We found that it is very difficult to produce such a rigid alignment without any user assistance or very accurate marker positions since a registration using the ICP algorithm [BM92] tends to converge into a local minimum if the deformation between the source and the target shape is too large. To obtain a more robust alignment for the final fitting of the morphable model, we compute a non-rigid registration of the template model $\mathcal{T}$ and the input scan.

Given a template mesh $\mathcal{T}$ and a coarsly aligned input scan $\mathcal{P}$, the goal of the non-rigid registration is to find a plausible space deformation Φ such that the distance between the deformed template mesh $\Phi(\mathcal{T})$ and the input scan $\mathcal{P}$ is minimized.

Following Suessmuth et al. [SZG10], we formulate the non-rigid registration as a variational problem. Therefore, we define a registration energy E_{reg}, which is composed of a fitting term E_{fit}, that attracts the template mesh towards the input scan geometry, and an internal energy term E_{def}, that serves as regularizer and prevents unnatural deformations of the template mesh. The deformation Φ which best aligns the template mesh with the input scan is then found by minimizing the registration energy.

6.3.1 Fitting Energy Term

As can be seen in the left image of Figure 6.4, the pre-processed scanner data is still noisy, contains striping artifacts and may contain holes. To alleviate these artifacts, we do not register the template mesh directly with the scanner data but with an implicit function f which has been fitted to it. Since the implicit function f is computed in such a way that its zero-set approximates the input data in a least squares sense, it reduces the noise and closes the remaining holes. Given the implicit function $f : \mathbb{R}^3 \mapsto \mathbb{R}$, the distance d of a point $\mathbf{x}$ to the zero-set of f can be approximated by

$$d(\mathbf{x}, f) = \frac{|f(\mathbf{x})|}{\|\nabla f(\mathbf{x})\|_2}.$$

The distance between the deformed template mesh $\Phi(\mathcal{T})$ and the zero-set of f, which provides us with the fitting energy term, can then be defined as the sum over the squared distances at the vertices $\hat{\mathbf{v}}_i$ of $\Phi(\mathcal{T})$:

$$E_{\text{fit}} = \sum_{\hat{\mathbf{v}}_i \in \Phi(\mathcal{T})} \left(\frac{|f(\mathbf{v}_i)|}{\|\nabla f(\mathbf{v}_i)\|_2} \right)^2 \tag{6.1}$$

6.3.2 Regularization Energy Term

We use a variant of the embedded graph based mesh deformation algorithm by Sumner et al. [SSP07] to model the deformation of the template mesh. Thereby, a global space deformation is obtained by blending neighboring affine transformations $A_i(\mathbf{x}) = \mathbf{M}_i(\mathbf{x} - \mathbf{p}_i) + \mathbf{p}_i + \mathbf{t}_i$ with local support, which are organized in a sparse graph. Here, $\mathbf{M}_i$ is a 3x3 matrix, $\mathbf{p}_i$ is the node's position and $\mathbf{t}_i$ is a translation vector. The local transformations define how the surrounding space is deformed. To obtain a deformation, which maintains local surface features, the local transformations should be close to rigid. An energy term E_{rot}, which punishes the deviation of a local transformation $\mathbf{M}_i$ from being rigid, can be defined as:

$$E_{\text{rot}}(\mathbf{M}_i) = \|\mathbf{M}_i^T\mathbf{M}_i - \mathbf{I}\|_F^2 \tag{6.2}$$

Since neighboring transformations have overlapping influence, they affect a common region in space. It is therefore important that they are consistent w.r.t. one another. This is enforced by a consistency energy term E_{con}, which measures the distance between the position to where a graph node is transformed by its own transformation and the position where it is mapped to by the transformation of a neighboring graph node:

$$\begin{aligned} E_{\text{con}}(e_{ij}) = &\left\|A_i(\mathbf{p}_j) - A_j(\mathbf{p}_j)\right\|_2^2 \\ &+ \left\|A_j(\mathbf{p}_i) - A_i(\mathbf{p}_i)\right\|_2^2, \end{aligned} \tag{6.3}$$

where e_{ij} is the edge connecting the two nodes i and j and $\mathbf{p}_i$ and $\mathbf{p}_j$ are the respective node positions.

6.3.3 Optimization

A combination of the fitting energy term and the two regularization energy terms leads to an energy function E_{reg}, which is a measure for the quality of

a registration introduced by a given deformation graph:

$$E_{\text{reg}} = \alpha E_{\text{fit}} + \beta \sum_i E_{\text{rot}}(\mathbf{M}_i) + \gamma \sum_{e_{ij}} E_{\text{con}}(e_{ij}). \tag{6.4}$$

Here, the first sum runs over all nodes in the deformation graph and the second sum runs over all edges. An optimal deformation Φ, which registers the template mesh $\mathcal{T}$ with the input scan can now be computed by minimizing Equation (6.4) for the unknown node transformations using a modified Gauss-Newton algorithm. As proposed by Li et al. [LSP08], we increase the influence of the fitting energy in each iteration by doubling α. Initially, we set $\alpha = 4$, $\beta = \gamma = 1$.

6.3.4 Fitting the morphable face model

The result of the non-rigid registration step is a deformed template mesh $\Phi(\mathcal{T})$ which tightly fits the input scan. However, as can be seen in Figure 6.4 (middle), this mesh still contains the bumps and dents that were originally present in the scanner data. To obtain the final 3D reconstruction of the face, we fit a morphable face model [BV99] to the deformed template obtained by the non-rigid registration. This projects the solution into the space of reasonable faces and thereby removes the mentioned artifacts.

Since the deformed template mesh provides one-to-one correspondences with the average shape of our morphable model, we can now robustly align $\Phi(\mathcal{T})$ with the morphable model using a Procrustes analysis again. Let $\hat{\mathcal{T}}$ be the deformed template mesh that was aligned with the average face $\mathcal{T}$ of the morphable model and $\mathbf{E}$ the (reduced) eigenbasis of the morphable model. We can thus construct a new face $\mathcal{F}$ from a given coefficient vector $\mathbf{c}$ by transforming the coefficient vector back into face space and adding the average face: $\mathcal{F} = \mathcal{T} + \mathbf{E} \cdot \mathbf{c}$. We find the coefficients $\mathbf{c}$ of the morphable model which approximate $\hat{\mathcal{T}}$ best by minimizing the least squares distance to the computed morph $\mathcal{F}$:

$$\min_{\mathbf{c}} \|(\mathcal{T} + \mathbf{E} \cdot \mathbf{c}) - \hat{\mathcal{T}}\|_2^2.$$

To obtain $\mathbf{c}$, we solve the resulting normal equations:

$$\mathbf{E}^T\mathbf{E} \cdot \mathbf{c} = \mathbf{E}^T \cdot (\hat{\mathcal{T}} - \mathcal{T}). \tag{6.5}$$

Since the eigenvector matrix $\mathbf{E}$ is orthogonal ($\mathbf{E}^T\mathbf{E} = \mathbf{I}$), the unknown optimal coefficients can be computed by a simple matrix-vector multiplication:

$$\mathbf{c} = \mathbf{E}^T \cdot (\hat{\mathcal{T}} - \mathcal{T}). \tag{6.6}$$

CHAPTER 7
Results

We have tested the proposed method on 20 individuals. In all cases, the proposed method was able to automatically produce accurate results. The results for four test persons are shown in Figure 17.6. As texture, we project the RGB image back onto the reconstructed face. The morphable model that we used for face fitting has been constructed from 53 faces [MS04], which were registered using the proposed algorithm for non-rigid registration. For all examples shown, we used the 25 most significant principal components of the face space to span the eigenbasis E. In the non-rigid registration, we performed six Gauss-Newton iterations to warp the template mesh towards the input scan. The reconstruction of the fully textured facial

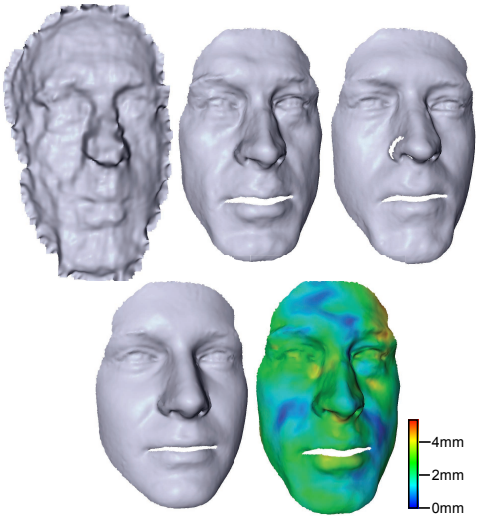


Figure 7.1: Comparison with ground truth data. From top left to bottom right: depth scan, fitted result, ground truth, average shape and deviation from ground truth.

Figure 7.2: Transferring an animation onto a captured avatar.

Figure 7.3: Face reconstruction results for four individuals. (from left to right) input RBG-D image, processed depth scan, fitted face model and textured face from three different views.

avatars took on average 18 seconds on an Intel Core i7 processor at 2.93GHz. To assess the quality of the reconstructed 3D face geometries, we compare a face model that was reconstructed using our algorithm to ground truth data in Figure 7.1. The ground truth face model was generated by scanning the test person with a high quality structured light scanner. The maximum deviation of our reconstruction from the ground truth model is 4.7mm, while the average deviation is roughly 2mm. Since the morphed face model generated by our algorithm has one-to-one correspondences with the average face, we can directly transfer semantic informations that are annotated to the average face onto our reconstruction. For example, facial animations (cf. Figure 7.2) can be easily cloned onto the new geometry.

CHAPTER 8
Conclusion

We have presented a novel system for the automatic generation of high-quality personalized avatars using the Microsoft Kinect sensor. The proposed system allows a huge audience to generate high-quality facial models within seconds. Using non-rigid registration to compute correspondences for the subsequent morphable model fitting makes our approach very robust and allows to generate convincing results even for bad input data.

The rather small morphable model we are using is currently one of the limiting factors. We plan to extend our data base and to handle models of the whole head. We further plan to extract geometry and texture information from multiple views and to capture whole head scans in super-resolution by registering the obtained input data. By using a segmented head model, we could further improve the expressibility of the morphable model. In addition, we plan to animate the computed results and provide the user the possibility to customize his avatar.

PART II

Model based Reconstruction of the Human Head

CHAPTER 9
Introduction

The method presented in Part I of this dissertation uses only a single frame of a commodity RGB-D sensor for the reconstruction of a facial avatar. While such a single-view setup is beneficial for the fast acquisition of data, it introduces occlusion and sampling problems, since most of the face is either not visible or observed from a creasing angle. This drastically limits the size of the region on the face that can be faithfully reconstructed. In addition, a single depth frame captured by a commodity RGB-D sensor is of comparably low quality and imposes a strict upper limit on the achievable accuracy. To further increase the reconstruction quality, we have to exploit the temporal coherence in the captured RGB-D stream.

Figure 9.1: Hardware setup: The RGB-D stream of a single PrimeSense Carmine is used to reconstruct high-quality head models using an interactive and intuitive reconstruction paradigm.

Here, we present a model-based reconstruction system for the human head that is intuitive and leverages a commodity sensor's RGB-D stream interactively. Using the presented system a single user is able to capture a high-quality facial avatar (see Figure 27.1) by moving his head freely in front of the sensor. During a scanning session the user receives interactive feedback from the system showing the current reconstruction state. Involving the user into the reconstruction process makes the system immersive and allows to refine the result. Our system effectively creates a high-quality virtual clone with a known semantical and topological structure that can be used in various applications ranging from virtual try-on to teleconferencing.

In the following, we discuss related work (Section 10), present an overview of our reconstruction system (Section 11.1) based on a fast GPU tracking and fitting pipeline (Section 11.2-11.4). We sum up by showing reconstruction results, applications and ground truth comparisons (Section 12) and give ideas for future work (Section 13).

CHAPTER 10
Related Work

3D-Reconstruction from RGB-D images and streams is a well studied topic in the geometry, vision and computer graphics communities. Due to the extensive amount of literature in this field, we have to restrict our discussion to approaches closely related to this work. Therefore, we will focus on model-free and model-based algorithms that are suitable for capturing a detailed digital model (shape and albedo) of a human head. We compare these approaches based on their generality and applicability and motivate our decision for a model-based reconstruction method.

10.1 Model-free 3D-Reconstruction

3D-Reconstruction mainly is about the acquisition of a real world object's shape and albedo. This includes capturing and aligning multiple partial scans [BM92, RL01, CM92] to obtain one complete reconstruction, data accumulation or fusion [CL96] and a final surface extraction step [LC87] to obtain a mesh representation. Systems based on a direct accumulation of the input sample points [WLG08, WWL*09] preserve information, but scale badly with the length of the input stream. In contrast, systems based on volumetric fusion [CL96] accumulate the data directly into a consistent representation, but do not keep the raw input for further postprocessing steps. The Kinect Fusion framework [NIH*11, IKH*11] is such a system and made real-time 3D-Reconstruction with a moving RGB-D camera viable for the first time. Because this approach deals with noise by spatial and temporal filtering, it is prone to oversmoothing (see Figure 10.1). Model-free approaches allow to digitize arbitrary real world objects with the drawback of the output to be only a polygon soup [LC87] with no topological and semantical information attached. Therefore, these reconstructions can not be automatically animated or used in virtual reality applications.

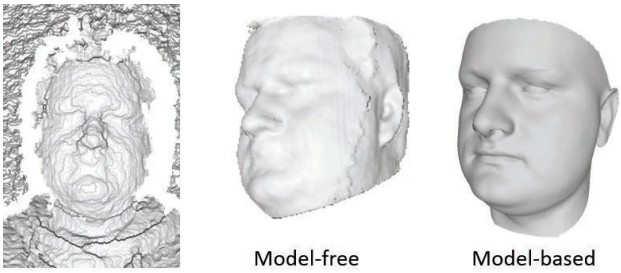


Figure 10.1: Comparison: Model-free approaches (Kinect Fusion, 256^3 voxel grid) are prone to oversmoothing. In contrast to this, our model-based approach allows to estimate fine-scale surface details.

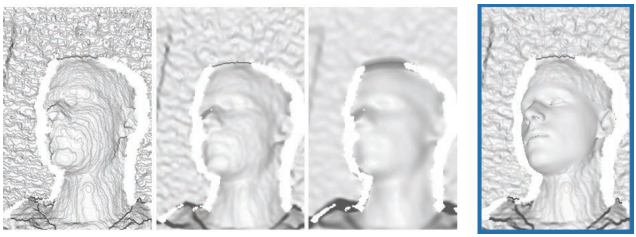


Figure 10.2: Denoising: Statistical noise removal (right) better deals with noisy input than spatial filtering approaches (left). Fine scale features are retained, while still effectivly dealing with noise.

10.2 Model-based 3D-Reconstruction

In contrast to model-free approaches, model-based methods heavily rely on statistical priors and are restricted to a certain class of objects (i.e., heads or bodies). This clear disadvantage in generality is compensated by leveraging the class-specific information built into the prior [BV99, BSS07, RV05, SE09]. In general, this leads to higher reconstruction quality, because noise can be statistically regularized (see Figure 10.2) and information can be propagated to yet unseen and/or unobservable regions. These properties make model-based reconstruction algorithms the first choice for applica-

tions that are focused on one specific object class.

Blanz and colleagues reconstruct 3D models from RGB and RGB-D input [BV99, BSS07] by fitting a statistical head model. These methods require user input during initialization and registration is performed in a time consuming offline process. Statistical models have also been extensively used to reconstruct templates for tracking facial animations [WBLP11, LYYB13, CWLZ13]. While the tracking is real-time, the reconstruction is performed offline. In addition, these methods only use depth and do not consider the RGB channels which allows to jointly estimate the illumination and can be used to improve tracking [WLGP09, CWZ*14]. Other methods specifically focused on reconstruction are either offline or do not use all the data of the RGB-D stream [aTH96, BKK*08, LPMM05, WSQO05, LMT98, GKMT01]. This also includes the method presented in Part I of this dissertation. In many cases, they only rely on a single input frame.

In contrast, our method utilizes all data provided by the RGB-D stream and gives the user immediate feedback. We specifically decided for a model-based approach because of its superior reconstruction quality and better reusability of the created models. Applications for our reconstructions range from animation re-targeting [SP04, WBLP11, LYYB13, CWLZ13] to face identification [PKA*09, RBV02], as well as virtual aging [SSSB07] and try-on [SRH*11]. The main three contributions of this work are:

- An intuitive reconstruction paradigm that is suitable even for unexperienced users
- The first interactive head reconstruction system that leverages all available information of the RGB-D stream
- A fast non-linear GPU-based Quasi-Newton solver that jointly solves for shape, albedo and illumination.

CHAPTER 11
Method

11.1 Pipeline Overview

Our reconstruction system (Figure 26.1) has been completely implemented on the GPU with an interactive application in mind. The user sits in front of a single RGB-D sensor (see Figure 27.1) and can freely move his head to obtain a complete and high-quality reconstruction. In the preprocessing stage, the captured RGB-D stream is bilaterally filtered [TM98] to remove high-frequency noise. We back-project the depth map to camera space and compute normals at the sample points using finite differences. We track the rigid motion of the head using a dense GPU-based iterative closest point (ICP) algorithm. After the global position and orientation of the head has been determined we use a non-rigid registration method that flip-flops between data fusion and model fitting. We fuse the unfiltered input data into a consistent mesh-based representation that shares its topology with the statistical prior. This step allows for super-resolution reconstructions,

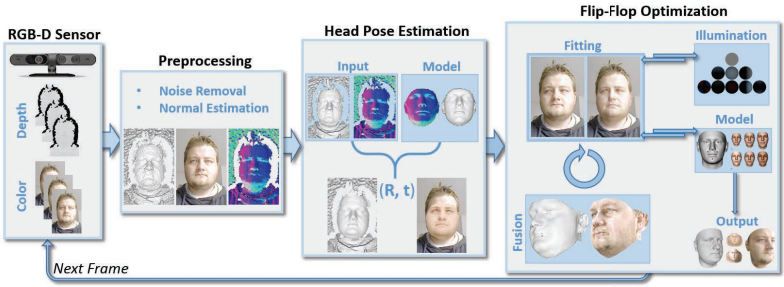


Figure 11.1: Per frame pipeline (from left to right): The RGB-D input stream is preprocessed, the rigid head pose is estimated, data is fused and a joint optimization problem for shape, albedo and illumination parameters is solved iteratively.

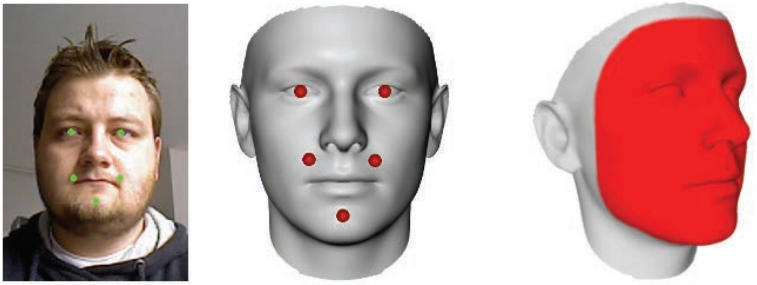


Figure 11.2: Automatically detected feature points for initial alignment (left, middle). Face region used for rigid model-to-frame alignment (right).

closes holes and fairs the data using a fast GPU-based thin-plate regularizer [BS08]. The resulting faired displacements define the position constraints for non-rigidly fitting the statistical model. After the best fitting model has been computed, we use the solution to initialize the next flip-flop step. This allows us to temporally fair and stabilize the target correspondences.

11.2 Head Pose Estimation

We compute an initial guess for the global head pose using the Procrustes algorithm [Gow75]. The required feature points, c.p. Figure 11.2 (left, middle), are automatically detected using Haar Cascade Classifiers [Bra00] for the mouth, nose and eyes. Corresponding features on the model have been manually preselected and stay constant. Starting from this initialization, we use a fast GPU-based implementation of a dense ICP algorithm in the spirit of [NIH*11, IKH*11] to compute the best fitting rigid transformation $\Phi^t(x) = R^t x + t^t$. We render the model using the last rigid transformation Φ^{t-1} to generate synthetic position $p_{i,j}$ and normal images $n_{i,j}$. Projective correspondence association is used between the input and the synthetic images. The registration error between the rendered model positions and the

target correspondences $t_{\mathcal{X}(i,j)}$ under a point-to-plane metric is:

$$\arg\min_{\hat{\Phi}} \sum_{i,j} w_{i,j} < n_{i,j}, \hat{\Phi}(p_{i,j}) - t_{\mathcal{X}(i,j)} >^2.$$

The corresponding 6×6 least squares system is constructed in parallel on the GPU and solved via SVD. We set the correspondence weights $w_{i,j}$ based on distance and normal deviation and prune correspondences ($w_{i,j} = 0$) if they are too far apart ($>$ 2cm), the normals do not match ($> 20°$) or the pixels are not associated with the head. The valid region for correspondence search, c.p. Figure 11.2 (right), is selected by a binary mask that specifies the part of the head that stays almost rigid (red) under motion. The predicted head pose of the current frame $\Phi^t(x) = \hat{\Phi}(x)\Phi^{t-1}(x)$ is used as starting point for the reconstruction of the non-rigid shape.

11.3 Data Fusion

Depth data of consumer level RGB-D sensors has a low resolution, contains holes and a lot of noise. We use a fusion scheme similar to Kinect Fusion [NIH*11, IKH*11] to achieve super-resolution reconstructions and effectivly deal with the noisy input. A per-vertex displacement map is defined on the template model to temporally accumulate the input RGB-D stream. Target scalar displacements are found by ray marching in normal direction, followed by four bisection steps to refine the solution. The resulting displacement map is faired by computing the best fitting thin-plate. We approximate the non-linear thin-plate energy [BS08] by replacing the fundamental forms with partial derivatives:

$$-\lambda_s \Delta d + \lambda_b \Delta^2 d = 0.$$

The parameters λ_s and λ_b control the stretching and bending resistance of the surface and d are the faired scalar displacements. These displacements are accumulated using an exponential average. A fast GPU-based preconditioned gradient descent with Jacobi preconditioner is used to solve the resulting least squares problem. The preconditioner is constant for a fixed

topology and can be precomputed, gradient evaluation is performed on-the-fly by iteratively applying the Laplacian kernel to compute the bi-Laplacian gradient component. This nicely regularizes out noise and fills holes in the data. For RGB, we use a one-frame integration scheme to deal with illumination changes.

11.4 Estimating Model Parameters

By projecting the accumulated data into the space of statistically plausible heads, noise can be regularized, artifacts can be removed and information can be propagated into yet unseen regions. We pose the estimation of the unknown shape α, albedo β and illumination γ parameters as a joint non-linear optimization problem. Shape and albedo is statistically modeled using the Basel Face Model [BV99, PKA*09], illumination is approximated using spherical harmonics. In the following, we give details on the used statistical model, the objective function and show how to efficiently compute best fitting parameters using a fast GPU-based non-linear Quasi-Newton solver.

11.4.1 Statistical Shape Model

The used statistical shape model encodes the shape (albedo) of 200 heads by assuming an underlying Gaussian distribution with mean μ_α (μ_β) and standard deviation σ_α (σ_β). The principal components E_α (E_β) are the directions of highest variance and span the space of plausible heads. New heads can be synthesized by specifying suitable model parameters α (β):

$$\begin{aligned} \text{Shape}: \ \mathcal{M}(\alpha) &= \mu_\alpha + E_\alpha \alpha, \\ \text{Albedo}: \ \mathcal{C}(\beta) &= \mu_\beta + E_\beta \beta. \end{aligned}$$

Synthesis is implemented using compute shaders. We use one warp per vertex and a fast warp reduction to compute the synthesized position (albedo).

11.4.2 Objective Function

Finding the instance that best explains the accumulated input observations is cast as a joint non-linear optimization problem:

$$E(\mathcal{P}) = \lambda_d E_d(\mathcal{P}) + \lambda_c E_c(\mathcal{P}) + \lambda_r E_r(\mathcal{P}).$$

The individual objectives E_d, E_c and E_r represent the depth, color and statistical regularization constraints. The empirically determined weights $\lambda_d = \lambda_c = 10$ and $\lambda_r = 1$ remain fixed and have been used for all shown examples. The parameter vector $\mathcal{P} = (\alpha, \beta, \gamma)$ encodes the degrees of freedom in the model. In the following, we will discuss the different objectives and their role in the optimization problem in more detail.

Depth Fitting Term

The depth fitting term incorporates the accumulated geometric target positions t_i into the optimization problem:

$$E_d(\mathcal{P}) = \sum_{i=1}^{n} ||\Phi^t(\mathcal{M}_i(\alpha)) - t_i||^2.$$

This functional only depends on the shape parameters α and measures the geometric point-point alignment error for every model vertex (n vertices). Minimizing this objective on its own is a linear least squares problem in the unknowns α due to the linearity of the model $\mathcal{M}$.

Color Fitting Term

The visual similarity of the synthesized model and the input RGB data is modeled by the color fitting term:

$$E_c(\mathcal{P}) = \sum_{i=1}^{n} ||\mathcal{I}(t_i) - \mathcal{R}(v_i, c_i, \gamma)||^2,$$

with $v_i = \Phi^t(\mathcal{M}_i(\alpha))$ being the current vertex position, $c_i = \mathcal{C}_i(\beta)$ the current albedo and $\mathcal{I}(t_i)$ is the RGB color assigned to the target position. This part of the objective function is non-linear in the shape α and linear in the illumination γ and albedo β. Illumination is modeled using spherical harmonics (spherical environment map). We assume a purely diffuse material and no self-shadowing:

$$\mathcal{R}(v_i, c_i, \gamma) = c_i \sum_{j=1}^{k} \gamma_j H_j(v_i).$$

H_j is the projection of the angular cosine fall-offs on the spherical harmonics basis. We use 3 spherical harmonics bands ($k = 9$ coefficients per channel).

Statistical Regularization

The heart of this method is a statistical regularizer [BV99] that takes the probability of the synthesized instances into account. This prevents overfitting the input data. Assuming a Gaussian distribution of the input, approximately 99% of the heads can be reproduced using parameters $x_i \in [-3\sigma_{x_i}, 3\sigma_{x_i}]$. Therefore, the parameters are constrained to be statistically small:

$$E_r(\mathcal{P}) = \sum_{i=1}^{m} \left[\frac{\alpha_i^2}{\sigma_{\alpha_i}^2} + \frac{\beta_i^2}{\sigma_{\beta_i}^2} \right] + \sum_{i=1}^{k} \left(\frac{\gamma_i}{\sigma_{\gamma_i}} \right)^2.$$

The standard deviations σ_{α_i} and σ_{β_i} are known from the shape model, $\sigma_{\gamma_i} = 1$ encodes the variability in the illumination and has been empirically determined. m specifies the number of used principal components.

11.4.3 Parameter Initialization

The objective function E is non-linear in its parameters $\mathcal{P}$. Therefore, a good initial guess is required to guarantee convergence to a suitable optimum. Current state-of-the-art methods heavily rely on user input in form

of sparse marker or silhouette constraints to guide the optimizer through the complex energy landscape. In this work, when tracking is started, we initialize the parameters by decoupling the optimization problem into three separate linear problems that can be solved independently. We start by fitting the model against the detected sparse set of markers to roughly estimate the size of the head. As mentioned earlier, the depth objective on its own is a linear least squares problem in the unknown shape parameters. After searching correspondences, a good approximation for α can be computed by solving the linear system that corresponds to the first 40 principal components. Then, the illumination parameters γ are estimated separately by assuming a constant average albedo. Finally, the albedo parameters β are initialized by assuming the computed shape and illumination to be fixed. Once the parameters have been initialized, a joint non-linear optimizer is used to refine the solutions.

11.4.4 Joint Non-Linear GPU Optimizer

To refine the solutions of the uncoupled optimization problems, we jointly solve for the parameters in each new input frame. We use a fast GPU-based implementation of a Quasi-Newton method to iteratively compute the best fit:

$$\mathcal{P}_{n+1} = \mathcal{P}_n - \lambda(HE(\mathcal{P}_n))^{-1}\Delta E(\mathcal{P}_n).$$

$HE(\mathcal{P}_n)$ is the Hessian matrix and ΔE the gradient of E, λ controls the step size. The step size is adaptively computed based on the change in the residual. We use a simple approximation of the inverse Hessian for scaling the descend directions. This is similar to preconditioning [WLHM10]. Because of the global support of the principal components, the derivatives with respect to α and β are influenced by all constraints. Therefore, we use one block per variable and a fast block reduction in shared memory to evaluate the derivatives. Per flip-flop step we perform 2 Quasi-Newton steps. During optimization we slowly increase the number of used principal directions to avoid local minima in the energy landscape.

CHAPTER 12
Results

In this section, we discuss the runtime behavior of our system, compare the reconstruction results to ground truth data obtained by a high-quality structured light scanner and present applications that leverage the known semantical structure of the reconstructed models.

12.1 Runtime Evaluation

Our reconstruction pipeline is completely implemented on the GPU to allow for interactive reconstruction sessions. The average per-frame runtime for the examples in Figure 17.6 is shown in Figure 12.1. We used an Intel Core i7-3770 CPU with a Nvidia Geforce GTX 780. Note, that for the person in the fourth row the beard could not be faithfully recovered by the model coefficients, this is due to the fact that facial hair is not contained in the model. But the generated texture captures and adds these details to the reconstruction. For all presented examples, we used 2 flip-flop steps (with 2 Quasi-Newton steps each) and 5 iterations of the thin-plate solver. We start

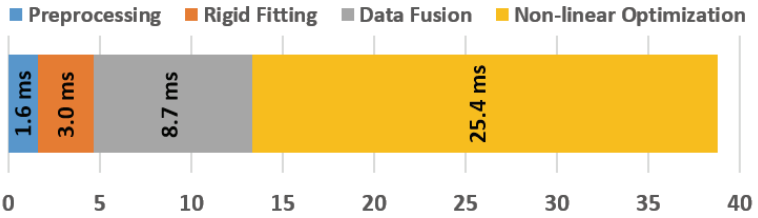


Figure 12.1: Per frame runtime: Runtime of the different stages in our reconstruction pipeline (in ms).

with 40 eigenvectors and slowly increase the number to 120. Note, that our system always remains interactive and gives the user direct feedback during the reconstruction process. For the results in Figure 17.6 the complete scanning sessions took about $3-5$ seconds each. In most cases, moving the head once from right to left is sufficient to compute a high quality model.

12.2 Reconstruction Quality

To evaluate the accuracy of the presented system we compare our reconstructions (PrimeSense Carmine 1.09) with high-quality 3D scans captured by a structured light scanner. The scanning sessions with the structured light setup took several minutes. As can be seen in Figure 12.2, the actual shape difference is small and our geometry has comparable quality. Because the eyes had to remain closed during structured light scanning, most of the error is located in the eye region. The mean error was 1.09 mm, 0.81 mm and 1.19 mm respectively (from top to bottom).

We also compare our reconstructions to the method presented in Part I of this dissertation (see Figure 12.3). In contrast to this approach, we can reconstruct the complete head, illumination correct the textures and have higher super-resolution geometry.

12.3 Applications

In this section, we discuss some applications that can directly use the reconstructed models, see Figure 27.2. We compute a complete texture by fusing multiple frames using pyramid blending [ABBO84], the required mask is automatically computed using depth and angular thresholds. The illumination parameters allow to compute illumination corrected textures and relight the head. Because of the known semantical structure, we can place a hat on the model (virtual try-on) and add textures. The known topological structure of the models allows to easily re-target animations (Figure 12.4).

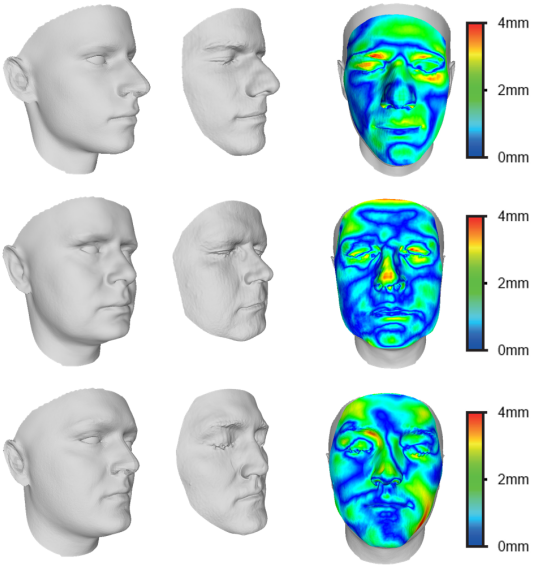


Figure 12.2: Ground truth comparison: Distance (right) between our reconstructions (left) and high-quality structured light scans (middle).

Figure 12.3: Comparison to the method presented in Part I: Previous method (left), this method (right).

Figure 12.4: Animation re-targeting: The known topology of the reconstructed models allows to easily re-target animation sequences. The input sequence has been taken from [SP04].

Figure 12.5: Applications: The textured models can be re-lighted, re-textured and used for virtual try-on.

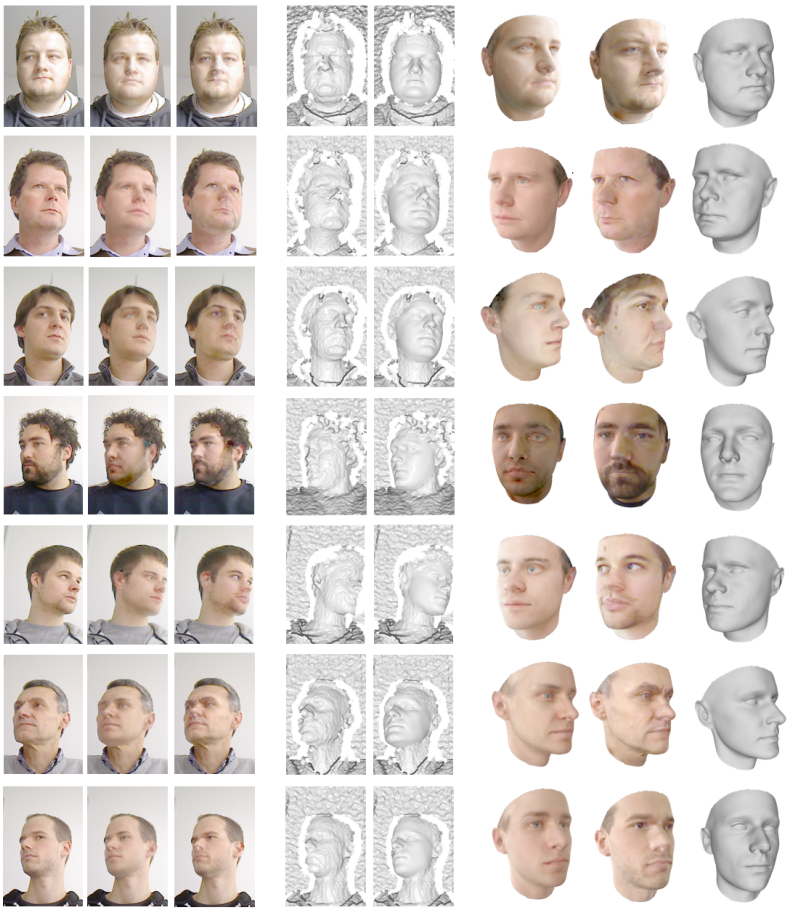


Figure 12.6: 3D-Reconstruction results (from left to right): Input color image, overlaid model (fitted color), overlaid model (textured), filtered input positions, overlaid model (phong shaded), reconstructed virtual avatar (fitted color, textured, phong shaded).

CHAPTER 13
Conclusion

We have presented a complete system for the reconstruction of high-quality models of the human head using a single commodity RGB-D sensor. A joint optimization problem is solved interactively to estimate shape, albedo and illumination parameters using a fast GPU-based non-linear solver. We have shown that the obtained quality is comparable to offline structured light scans. Because of the known topological and semantical structure of the models, they can be directly used as input for various virtual reality applications.

In the future, we plane to add motion tracking to our system to animate the reconstructions. We hope that we can leverage the reconstructed albedo to make non-rigid tracking more robust.

PART III

Online 3D Reconstruction at Scale

CHAPTER 14
Introduction

While the methods presented in Part I and Part II of this dissertation exploit a statistical prior to obtain high-quality reconstructions given the noisy data captured by commodity RGB-D sensors, they are restricted to a certain object class, i.e. human heads. To be able to capture arbitrary scenes at all different scales in real-time, a more general and yet efficient scene representation is required.

The ability to obtain reconstructions of arbitrary scenes in *real-time* opens up various interactive applications including: augmented reality (AR) where real-world geometry can be fused with 3D graphics and rendered live to the user; autonomous guidance for robots to reconstruct and respond rapidly to their environment; or even to provide immediate feedback to users during 3D scanning.

Such an online reconstruction system requires incremental *fusion* and *alignment* of many overlapping depth maps into a single 3D representation that is continuously refined. This is challenging particularly when real-time performance is required without trading fine-quality reconstructions and spatial scale. Many state-of-the-art online techniques therefore employ different types of underlying data structures accelerated using graphics hardware. These however have particular trade-offs in terms of reconstruction speed, scale, and quality.

We contribute a new real-time surface reconstruction system which supports fine-quality reconstructions at scale. Our approach carries the benefits of volumetric approaches, but does not require either a memory constrained voxel grid or the computational overheads of a hierarchical data structure. Our method is based on a simple memory and speed efficient spatial hashing technique that compresses space, and allows for real-time fusion of referenced implicit surface data, without the need for a hierarchical data structure. Surface data is only stored densely in cells where measure-

ments are observed. Additionally, data can be streamed efficiently in or out of the hash table, allowing for further scalability during sensor motion. We show interactive reconstructions of a variety of scenes, reconstructing both fine-grained and large-scale environments. We illustrate how all parts of our pipeline from depth map pre-processing, sensor pose estimation, depth map fusion, and surface rendering are performed at real-time rates on commodity graphics hardware. We conclude with a comparison to current state-of-the-art systems, illustrating improved performance and reconstruction quality.

CHAPTER 15
Related work

There is over three decades of research on 3D reconstruction. In this section we review relevant systems, with a focus on online reconstruction methods and active sensors. Unlike systems that focus on reconstruction from a complete set of 3D points [HDD*92, KBH06], online methods require incremental *fusion* of many overlapping depth maps into a single 3D representation that is continuously refined. Typically methods first register or *align* sequential depth maps using variants of the Iterative Closest Point (ICP) algorithm [BM92, CM92].

Parametric methods [CM92, HHI95] simply average overlapping samples, and connect points by assuming a simple surface topology (such as a cylinder or a sphere) to locally fit polygons. Extensions such as mesh zippering [TL94] select one depth map per surface region, remove redundant triangles in overlapping regions, and stitch meshes. These methods handle some denoising by local averaging of points, but are fragile in the presence of outliers and areas with high curvature. These challenges associated with working directly with polygon meshes have led to many other reconstruction methods.

Point-based methods perform reconstruction by merging overlapping data points, and avoid inferring connectivity. Rendering the final model is performed using point-based rendering techniques [GP07]. Given the output from most depth sensors are 3D point samples, it is natural for reconstruction methods to work directly with such data. Examples include in-hand scanning systems [RHHL02, WWLVG09], which support reconstruction of only single small objects. At this small scale, high-quality [WWLVG09] reconstructions have been achieved. Larger scenes have been reconstructed by trading real-time speed and quality [HKH*12, SB12]. These methods lack the ability to directly model connected surfaces, requiring additional expensive and often offline steps to construct surfaces; e.g., using volumet-

ric data structures [RHHL02].

Height-map based representations explore the use of more compact 2.5D continuous surface representations for reconstruction [PNF*08, GPF10]. These techniques are particularly useful for modeling large buildings with floors and walls, since these appear as clear discontinuities in the height-map. Multi-layered height-maps have been explored to support reconstruction of more complex 3D shapes such as balconies, doorways, and arches [GPF10]. While these methods support more efficient compression of surface data, the 2.5D representation fails to reconstruct many types of complex 3D structures.

An alternative method is to use a fully *volumetric* data structure to *implicitly* store samples of a continuous function [HSIW96, CL96, WSI98]. In these methods, depth maps are converted into signed distance fields and cumulatively averaged into a regular voxel grid. The final surface is extracted as the zero-level set of the implicit function using isosurface polygonisation (e.g., [LC87]) or raycasting. A well-known example is the method of Curless and Levoy [CL96], which for active triangulation-based sensors such as laser range scanners and structured light cameras, can generate very high quality results [CL96, LPC*00, ZK13]. KinectFusion [NIH*11, IKH*11] recently adopted this volumetric method and demonstrated compelling real-time reconstructions using a commodity GPU.

While shown to be a high quality reconstruction method, particularly given the computational cost, this approach suffers from one major limitation: the use of a regular voxel grid imposes a large memory footprint, representing both empty space and surfaces densely, and thus fails to reconstruct larger scenes without compromising quality.

Scaling-up Volumetric Fusion Recent work begins to address this spatial limitation of volumetric methods in different ways. [KLL*13] use a point-based representation that captures qualities of volumetric fusion but removes the need for a spatial data structure. While demonstrating compelling scalable real-time reconstructions, the quality is not on-par with true volumetric methods.

Moving volume methods [RV12, WJK*12] extend the GPU-based pipeline of KinectFusion. While still operating on a very restricted regular grid, these methods stream out voxels from the GPU based on camera motion, freeing space for new data to be stored. In these methods the streaming is one-way and lossy. Surface data is compressed to a mesh, and once moved to host cannot be streamed back to the GPU. While offering a simple approach for scalability, at their core these systems still use a regular grid structure, which means that the active volume must remain small to ensure fine-quality reconstructions. This limits reconstructions to scenes with close-by geometric structures, and cannot utilize the full range of data for active sensors such as the Kinect.

This limit of regular grids has led researcher to investigate more efficient volumetric data structures. This is a well studied topic in the volume rendering literature, with efficient methods based on sparse voxel octrees [LK11, KSA13], simpler multi-level hierarchies and adaptive data structures [KE02, LHN05, BC08, RCBW12] and out-of-core streaming architectures for large datasets [HBJP12, CNLE09]. These approaches have begun to be explored in the context of online reconstruction, where the need to support real-time updates of the underlying data adds a fundamentally new challenge.

For example, [ZGHG11] demonstrate a GPU-based octree which can perform Poisson surface reconstruction on 300K vertices at interactive rates. [ZZZL12] implement a 9- to 10-level octree on the GPU, which extends the KinectFusion pipeline to a larger $8m \times 8m \times 2m$ indoor office space. The method however requires a complex octree structure to be implemented, with additional computational complexity and pointer overhead, with only limited gains in scale.

In an octree, the resolution in each dimension increases by a factor of two at each subdivision level. This results in the need for a deep tree structure for efficient subdivision, which conversely impacts performance, in particular on GPUs where tree traversal leads to thread divergence. The rendering literature has proposed many alternative hierarchical data structures [LHN05, KE02, LK11, KSA13, RCBW12]. In [CBI13] an N^3 hierarchy [LHN05] was adopted for 3D reconstruction at scale, and the optimal

tree depth and branching factor were empirically derived (showing large branching factors and a shallow tree optimizes GPU performance). While avoiding the use of an octree, the system still carries computational overheads in realizing such a hierarchical data structure on the GPU. As such this leads to performance that is only real-time on specific scenes, and on very high-end graphics hardware.

CHAPTER 16
Method

16.1 Algorithm Overview

We extend the volumetric method of Curless and Levoy [CL96] to reconstruct high-quality 3D surfaces in *real-time* and at *scale*, by incrementally fusing noisy depth maps into a memory and speed efficient data structure. Curless and Levoy have proven to produce compelling results given a simple cumulative average of samples. The method supports incremental updates, makes no topological assumptions regarding surfaces, and approximates the noise characteristics of triangulation based sensors effectively. Further, while an implicit representation, stored isosurfaces can be readily extracted. Our method addresses the main drawback of Curless and Levoy: supporting efficient scalability. Next, we review the Curless and Levoy method, before the description of our new approach.

Implicit Volumetric Fusion Curless and Levoy's method is based on storing an implicit signed distance field (SDF) within a volumetric data structure. Let us consider a regular dense voxel grid, and assume the input is a sequence of depth maps. The depth sensor is initialized at some origin relative to this grid (typically the center of the grid). First, the rigid six degree-of-freedom (6DoF) ego-motion of the sensor is estimated, typically using variants of ICP [BM92, CM92].

Each voxel in the grid contains two values: a signed distance and weight. For a single depth map, data is integrated into the grid by uniformly sweeping through the volume, culling voxels outside of the view frustum, projecting all voxel centers into the depth map, and updating stored SDF values. All voxels that project onto the same pixel are considered part of the depth sample's footprint. At each of these voxels a signed distance from the

voxel center to the observed surface measurement is stored, with positive distances in front, negative behind, and nearing zero at the surface interface.

To reduce computational cost, support sensor motion, and approximate sensor noise, Curless and Levoy introduce the notion of a truncated SDF (TSDF) which only stores the signed distance in a region around the observed surface. This region can be adapted in size, approximating sensor noise as a Gaussian with variance based on depth [CCK94, NIL12]. Only TSDF values stored in voxels within these regions are updated using an averaging scheme to obtain an estimate of the surface. Finally, voxels (in front of the surface) that are part of each depth sample's footprint, but outside of the truncation region are explicitly marked as free-space. This allows removal of outliers based on free-space violations.

Voxel Hashing Given Curless and Levoy truncate SDFs around the surface, the majority of data stored in the regular voxel grid is marked either as free space or as unobserved space rather than surface data. The key challenge becomes how to design a data structure that exploits this underlying sparsity in the TSDF representation.

Our approach specifically avoids the use of a dense or hierarchical data structure, removing the need for a memory intensive regular grid or computationally complex hierarchy for volumetric fusion. Instead, we use a simple hashing scheme to compactly store, access and update an implicit surface representation.

In the graphics community, efficient spatial hashing methods have been explored in the context of a variety of 2D/3D rendering and collision detection tasks [THM*03, LH06, BC08, ASA*09, PM11, GLHL11]. Sophisticated methods have been proposed for efficient GPU-based hashing that greatly reduce the number of hash entry collisions.

Our goal is to build a real-time system that employs a spatial hashing scheme for scalable volumetric reconstruction. This is non-trivial for 3D reconstruction as the geometry is unknown ahead of time and continually chang-

ing. Therefore, our hashing technique must support dynamic allocations and updates, while minimizing and resolving potential hash entry collisions, without requiring a-priori knowledge of the contained surface geometry. In approaching the design of our data structure, we have purposefully chosen and extended a simple hashing scheme [THM*03], and while more sophisticated methods exist, we show empirically that our method is efficient in terms of speed, quality, and scalability.

The hash table sparsely and efficiently stores and updates TSDFs. In the following we describe the data structure in more detail, and demonstrate how it can be efficiently implemented on the GPU. We highlight some of the core features of our data structure, including:

- The ability to efficiently compress volumetric TSDFs, while maintaining surface resolution, without the need for a hierarchical spatial data structure.
- Fusing new TSDF samples efficiently into the hash table, based on insertions and updates, while minimizing collisions.
- Removal and garbage collection of voxel blocks, without requiring costly reorganization of the data structure.
- Lightweight bidirectional streaming of voxel blocks between host and GPU, allowing unbounded reconstructions.
- Extraction of isosurfaces from the data structure efficiently using standard raycasting or polygonization operations, for rendering and camera pose estimation.

System Pipeline Our pipeline is depicted in Figure 16.1. Central is a hash table data structure that stores sub-blocks containing SDFs, called *voxel blocks*. Each occupied entry in our hash table refers to an allocated voxel block. At each voxel we store a TSDF, weight, and an additional color value. The hash table is unstructured; i.e., neighboring voxel blocks are not stored spatially, but can be in different parts of the hash table. Our hashing function allows an efficient look-up of voxel blocks, using specified (inte-

ger rounded) world coordinates. Our hash function aims to minimize the number of collisions and ensures no duplicates exist in the table.

Given a new input depth map, we begin by performing fusion (also referred to as integration). We first allocate new voxel blocks and insert block descriptors into the hash table, based on an input depth map. Only occupied voxels are allocated and empty space is not stored. Next we sweep each allocated voxel block to update the SDF, color and weight of each contained voxel, based on the input depth and color samples. In addition, we garbage collect voxel blocks which are too far from the isosurface and contain no weight. This involves freeing allocated memory as well as removing the voxel block entry from the hash table. These steps ensure that our data structure remains sparse over time.

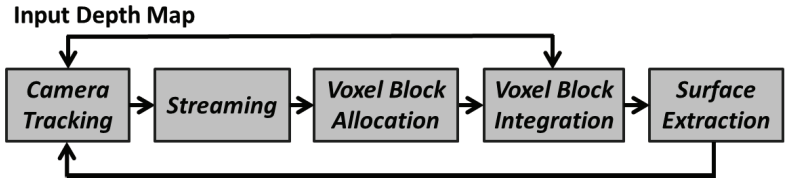


Figure 16.1: Pipeline overview.

After integration, we raycast the implicit surface from the current estimated camera pose to extract the isosurface, including associated colors. This extracted depth and color buffer is used as input for camera pose estimation: given the next input depth map, a projective point-plane ICP [CM92] is performed to estimate the new 6DoF camera pose. This ensures that pose estimation is performed *frame-to-model* rather than frame-to-frame mitigating some of the issues of drift (particularly for small scenes) [NIH*11]. Finally, our algorithm performs bidirectional streaming between GPU and host. Hash entries (and associated voxel blocks) are streamed to the host as their world positions exit the estimated camera view frustum. Previously streamed out voxel blocks can also be streamed back to the GPU data structure when revisiting areas.

16.2 Data Structure

Figure 16.2 shows our voxel hashing data structure. Conceptually, an infinite uniform grid subdivides the world into *voxel blocks*. Each block is a small regular voxel grid. In our current implementation a voxel block is composed of 8^3 voxels. Each voxel stores a TSDF, color, and weight and requires 8 bytes of memory:

```
struct Voxel {
   float sdf;
   uchar colorRGB[3];
   uchar weight;
};
```

To exploit sparsity, voxel blocks are only allocated around reconstructed surface geometry. We use an efficient GPU accelerated *hash table* to manage allocation and retrieval of voxel blocks. The hash table stores *hash entries*, each containing a pointer to an allocated voxel block. Voxel blocks can be retrieved from the hash table using integer world coordinates (x, y, z). Finding the coordinates for a 3D point in world space is achieved by simple multiplication and rounding. We map from a world coordinate (x, y, z) to the hash value $H(x, y, z)$ using the following *hashing function*:

$$H(x, y, z) = (x \cdot p_1 \oplus y \cdot p_2 \oplus z \cdot p_3) \bmod n$$

where $p_1 = 73856093$, $p_2 = 19349669$, and $p_3 = 83492791$ are large prime numbers [THM*03], and n is the hash table size. In addition to storing a pointer to the voxel block, each hash entry also contains the associated world position, and an offset pointer to handle collisions efficiently (described in the next section).

```
struct HashEntry {
   short position[3];
   short offset;
   int   pointer;
};
```

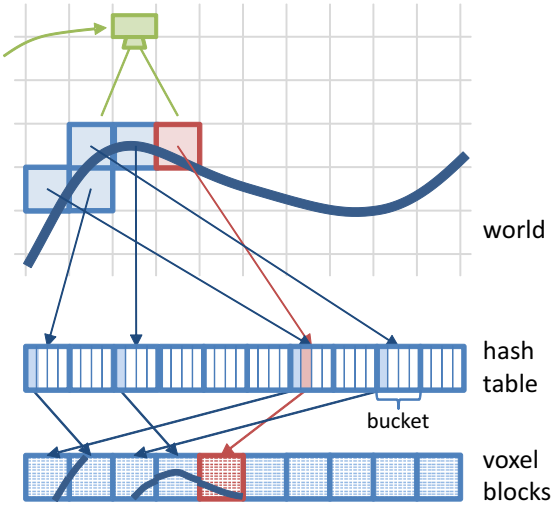


Figure 16.2: Voxel hashing data structure. Conceptually, an infinite uniform grid partitions the world. Using our hash function, we map from integer world coordinates to hash buckets, which store a small array of pointers to regular grid voxel blocks. Each voxel block contains an 8^3 grid of SDF values. When information for the red block gets added, a collision appears which is resolved by using the second element in the hash bucket.

16.2.1 Resolving Collisions

Collisions appear if multiple allocated blocks are mapped to the same hash value (see red block in Figure 16.2). We handle collisions by uniformly organizing the hash table into *buckets*, one per unique hash value. Each bucket sequentially stores a small number of hash entries. When a collision occurs, we store the block pointer in the next available sequential entry in the bucket (see Figure 16.3). To find the voxel block for a particular world position, we first evaluate our hash function, and lookup and traverse the associated bucket until our block entry is found. This is achieved by simply comparing the stored hash entry world position with the query position.

With a reasonable selection of the hash table and bucket size (see later),

rarely will a bucket overflow. However, if this happens, we append a linked list entry, filling up other free spots in the next available buckets. The (relative) pointers for the linked lists are stored in the offset field of the hash table entries. Such a list is appended to a full bucket by setting the offset pointer for the *last* entry in the bucket. All following entries are then chained using the offset field. In order to create additional links for a bucket, we linearly search across the hash table for a free slot to store our entry, appending to the link list accordingly. We avoid the last entry in each bucket, as this is locally reserved for the link list head.

As shown later, we choose a table and bucket size that keeps the number of collisions and therefore appended linked lists to a minimum for most scenes, as to not impact overall performance.

16.2.2 Hashing operations

Insertion To insert new hash entries, we first evaluate the hash function and determine the target bucket. We then iterate over all bucket elements including possible lists attached to the last entry. If we find an element with the same world space position we can immediately return a reference. Otherwise, we look for the first empty position within the bucket. If a position in the bucket is available, we insert the new hash entry. If the bucket is full, we append an element to its linked list element (see Figure 16.3).

To avoid race conditions when inserting hash entries in parallel, we lock a bucket atomically for writing when a suitable empty position is found. This eliminates duplicate entries and ensures linked list consistency. If a bucket is locked for writing, all other allocations for the same bucket are staggered until the next frame is processed. This may delay some allocations marginally. However, in practice this causes no degradation in reconstruction quality, particularly as the Curless and Levoy method supports order independent updates.

Retrieval To read the hash entry for a query position, we compute the hash value and perform a linear search within the corresponding bucket.

If no entry is found, and the bucket has a linked list associated (the offset value of the last entry is set), we also have to traverse this list. Note that we do not require a bucket to be filled from left to right. As described below, removing values can lead to fragmentation, so traversal does not stop when empty entries are found in the bucket.

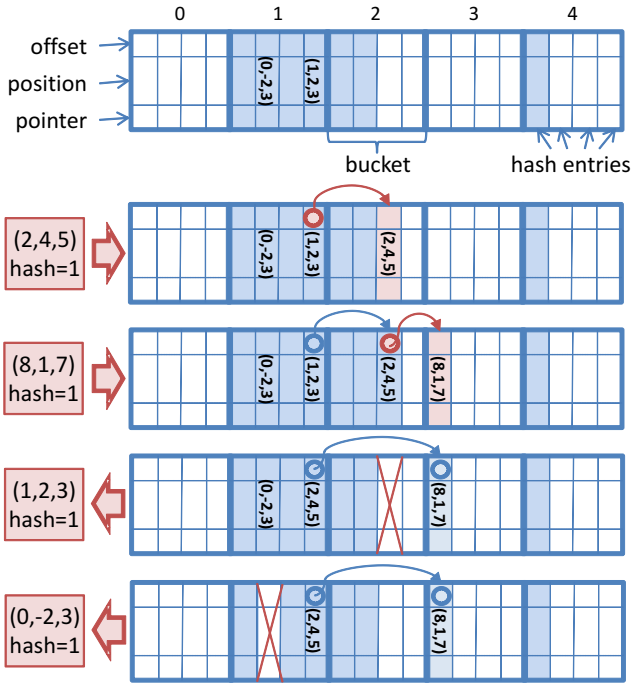


Figure 16.3: The hash table is broken down into a set of buckets. Each slot is either unallocated (white) or contains an entry (blue) storing the query world position, pointer to surface data, and an offset pointer for dealing with bucket overflow. Example hashing operations: for illustration, we insert and remove four entries that all map to hash $= 1$ and update entries and pointers accordingly.

Deletion Deleting a hash entry is similar to insertion. For a given world position we first compute the hash and then linearly search the correspond-

ing hash bucket including list traversal. If we have found the matching entry without list traversal we can simply delete it. If it is the last element of the bucket and there was a non-zero offset stored (i.e., the element is a list head), we copy the hash entry pointed to by the offset into the last element of the bucket, and delete it from its current position. Otherwise if the entry is a (non-head) element in the linked list, we delete it and correct list pointers accordingly (seeFigure 16.3). Synchronization is not required for deletion directly within the bucket. However, in the case we need to modify the linked list, we lock the bucket atomically and stagger further list operations for this bucket until the next frame.

16.3 Voxel Block Allocation

Before integration of new TSDFs, voxel blocks must be allocated that fall within the footprint of each input depth sample, and are also within the truncation region of the surface measurement. We process depth samples in parallel, inserting hash entries and allocating voxel blocks within the truncation region around the observed surface. The size of the truncation is *adapted* based on the variance of depth to compensate for larger uncertainty in distant measurements [CCK94, NIL12].

For each input depth sample, we instantiate a ray with an interval bound to the truncation region. Given the predefined voxel resolution and block size, we use DDA [AW87] to determine all the voxel blocks that intersect with the ray. For each candidate found, we insert a new voxel block entry into the hash table. In an idealized case, each depth sample would be modeled as an entire frustum rather than a single ray. We would then allocate all voxel blocks within the truncation region that intersect with this frustum. In practice however, this leads to degradation in performance (currently 10-fold). Our ray-based approximation provides a balance between performance and precision. Given the continuous nature of the reconstruction, the frame rate of the sensor, and the mobility of the user, this in practice leads to no holes appearing between voxel blocks at larger distances.

Once we have successfully inserted an entry into the hash table, we allocate a portion of preallocated heap memory on the GPU to store voxel block data. The heap is a linear array of memory, allocated once upon initialization. It is divided into contiguous blocks (mapping to the size of voxel blocks), and managed by maintaining a list of available blocks. This list is a linear buffer with indices to all unallocated blocks. A new block is allocated using the last index in the list. If a voxel block is subsequently freed, its index is appended to the end of the list. Since the list is accessed in parallel, synchronization is necessary, by incrementing or decrementing the end of list pointer using an atomic operation.

16.4 Voxel Block Integration

We update all allocated voxel blocks that are currently within the camera view frustum. After the previous step (see Section 16.3), all voxel blocks in the truncation region of the visible surface are allocated. However, a large fraction of the hash table will be empty (i.e., not refer to any voxel blocks). Further, a significant amount of voxel blocks will be outside the viewing frustum. Under these assumptions, TSDF integration can be done very efficiently by only selecting available blocks inside the current camera frustum.

Voxel Block Selection To select voxel blocks for integration, we first in parallel access all hash table entries, and store a corresponding binary flag in an array for an occupied and visible voxel block, or zero otherwise. We then scan this array using a *parallel prefix sum* technique [HSO07]. To facilitate large scan sizes (our hash table can have millions of entries) we use a three level up and down sweep. Using the scan results we compact the hash table into another buffer, which contains all hash entries that point to voxel blocks within the view frustum (see Figure 16.4). Note that voxel blocks are not copied, just their associated hash entries.

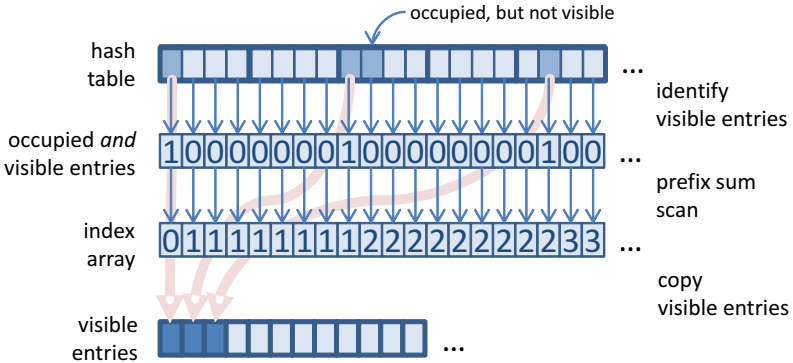


Figure 16.4: Voxel block selection: in a first step, all occupied *and* visible hash entries are identified. By using a parallel prefix sum scan and a simple copy kernel, these are copied to a much smaller, contiguous array that can be efficiently traversed in parallel in subsequent operations.

Implicit Surface Update The generated list of hash entries is then processed in parallel to update TSDF values. A single GPGPU kernel is executed for each of the associated blocks, with one thread allocated per voxel. That means that a voxel block will be processed on a single GPU multiprocessor, thus maximizing cache hits and minimizing code divergence. In practice, this is more efficient than assigning a single thread to process an entire voxel block.

Updating voxel blocks involves re-computation of the associated TSDFs, weights and colors. Distance values are integrated using a running average as in Curless and Levoy [CL96]. We set the integration weights according to the depth values in order to incorporate the noise characteristics of the sensor; i.e., more weight is given to nearer depth measurements for which we assume less noise. Colors are also updated according to a running average, but with much more weight given to recent color samples (to reduce washing out colors).

One important part of the integration step is to update *all* voxel blocks that fall into the current frustum, irrespective of whether they reside in the cur-

rent truncation region. This can be due to surfaces being physically moved, or small outliers in the depth map being allocated previously, which are no longer observed. These blocks are not treated any differently, and continuously updated. As shown next however, we evaluate all voxel blocks after integration to identify such candidates for potential garbage collection.

Garbage Collection Garbage collection removes voxel blocks allocated due to noisy outliers and moved surfaces. This step operates on the compacted hash table we obtained previously. For each associated voxel block we perform a summarization step to obtain both the minimum absolute TSDF value and the maximum weight. If the maximum weight of a voxel block is zero or the minimum TSDF is larger than a threshold we flag the block for deletion. In a second pass, in parallel we delete all flagged entries using the hash table delete operation described previously. When a hash entry gets deleted successfully, we also free the corresponding voxel block by appending the voxel block pointer to the heap (cf. Section 16.3).

16.5 Surface Extraction

We perform raycasting to extract the implicitly stored isosurface. First, we compute the start and end points for each ray by *conservatively* rasterizing the entire bounding box of all allocated voxel blocks in the current view frustum. In parallel, we rasterize each voxel block (retrieved from the compact hash table buffer computed during integration) in two passes, and generate two z-buffers for the minimum and maximum depth. This demonstrates another benefit for our linear hash table data structure (over hierarchical data structures), allowing fast parallel access to all allocated blocks for operations such as rasterization.

For each output pixel, we march a ray from the associated minimum to the maximum depth values. During marching we must evaluate the TSDF at neighboring world positions along the current ray. In this step, unallocated voxel blocks are also considered as empty space. Within occupied voxel

blocks, we apply tri-linear interpolation by looking up the eight neighboring voxels. One special case that needs to be considered is sampling across voxel block boundaries. To deal with this, we retrieve neighboring voxels by lookup via the hash table rather than sampling the voxel block directly. In practice, we use hash table lookups irrespective of whether the voxel is on a block boundary. Due to caching, reduced register count per thread, and non-divergent code, this increases performance over direct block sampling. We have also tried using a one-voxel overlap region around blocks in order to simplify tri-linear reads without the need of accessing multiple voxel blocks. However, that approximately doubled the memory footprint and we found that required overlap synchronization for surface integration bears significant computational overhead.

To locate the surface interface (zero-crossing) we determine sign changes for current and previous (tri-linearly-interpolated) TSDF values. We ignore zero-crossings from negative to positive as this refers to back-facing surface geometry. In order to speed up ray marching, we skip a predefined interval (half the minimum truncation value). This avoids missing isosurfaces but provides only coarse zero-crossing positions. To refine further, we use iterative line search once a zero-crossing is detected to estimate the true surface location.

Camera Tracking Once the surface is extracted via raycasting, it can be shaded for rendering, or used for frame-to-model camera pose estimation [NIH*11]. We use the next input frame along with the raycasted depth map to estimate pose. This ensures that the new pose is estimated prior to depth map fusion. Pose is estimated using the point-plane variant of ICP [CM92] with projective data association. The point-plane energy function is linearized [Low04] on the GPU to a 6×6 matrix using a parallel reduction and solved via Singular Value Decomposition on the CPU. As our data structure also stores associated color data, we incorporate a weighting factor in the point-plane error-metric based on color consistency between extracted and input RGB values [JBK99].

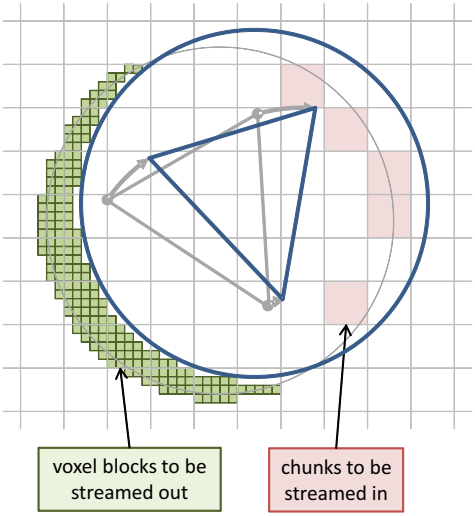


Figure 16.5: Data streaming: camera moves from left to right. Voxel blocks leaving the camera frustum are streamed out (green). Streaming in happens on a chunk basis (red blocks).

16.6 Streaming

The basic data structure described so far allows for high-resolution voxel blocks to be modeled beyond the resolution and range of current commodity depth cameras (see Section 17). However, GPU memory and performance become a consideration when we attempt to maintain surface data far outside of the view frustum in the hash table. To deal with this issue and allow unbounded reconstructions, we utilize a bidirectional GPU-Host streaming scheme.

Our unstructured data structure is well-suited for this purpose, since streaming voxel blocks in or out does not require any reorganization of the hash table. We create an *active region* defined as a sphere containing the current camera view frustum and a safety region around it. For a standard Kinect, we assume a depth range up to eight meters. We locate the center of the

sphere four meters from the camera position and use a radius of eight meters (see Figure 16.5). Bidirectional streaming of voxel blocks happens every frame at the beginning of the pipeline directly after pose estimation.

16.6.1 GPU-to-Host Streaming

To stream voxel blocks out of the active region, we first access the hash table in parallel and mark voxel blocks which moved out of the active region. For all these candidates we delete corresponding hash entries, and append them efficiently to an intermediate buffer. In a second pass, for all these hash entries, corresponding voxel blocks are copied to another intermediate buffer. The original voxel blocks are then cleared and corresponding locations are appended back to the heap, so they can be reused. Finally, these intermediate buffers are copied back to the host for access.

On the host, voxel data is no longer organized into a hash table. Instead, we logically subdivide the world space uniformly into *chunks* (in our current implementation each set to $1m^3$). Voxel blocks are appended to these chunks using a linked list. For each voxel block we store the voxel block descriptor which corresponds to hash entry data, as well as the voxel data.

16.6.2 Host-to-GPU Streaming

For Host-to-GPU streaming we first identify chunks that completely fall into the spherical active region again, due to the user moving back to a previously reconstructed region. In contrast to GPU-to-CPU streaming which works on a per voxel block level, CPU-to-GPU streaming operates on a per chunk basis. So if a chunk is identified for streaming *all* voxel blocks in that chunk will be streamed to the GPU. This enhances performance, given the high host-GPU bandwidth and ability to efficiently cull voxel blocks outside of the view frustum.

Due to limited CPU compute per frame, streaming from host-to-GPU is staggered, one chunk per frame. We select the chunk tagged for streaming that is most near to the camera frustum center. We then copy the chunk to

the GPU via the intermediate buffers created for GPU-to-Host streaming. After copying to the GPU, in parallel we insert voxel block descriptors as entries into the hash table, allocating voxel block memory from the heap, and copy voxel data accordingly. This is similar to the allocation phase (see Section 16.3), however, when streaming data, all hash entries must be inserted within a single frame, rather than staggering the insertions.

For a streamed voxel block we check the descriptor and atomically compare whether the position is occupied in the table. If an entry exists, we proceed to search for the next available free position in the bucket (as described below, we ensure that there are no duplicates). Otherwise we write the streamed hash entry at that position into the hash table. If the bucket is full, the entry is appended at the end of the list. Both writing a free entry directly in the bucket or appending it to the end of a linked list must be performed atomically.

16.6.3 Stream and Allocation Synchronization

One important consideration for streaming is to ensure that voxel blocks are never duplicated on host or GPU, leading to potential memory leaks. Given that Host-to-GPU streaming is staggered, there are rare cases where voxel blocks waiting to be streamed may enter the view frustum. We must verify that there is no new allocation of these voxel blocks in these staggered regions. To this end we store a binary occupancy grid on the GPU, where each entry corresponds to a particular chunk. Setting the bit indicates that the chunk resides on the GPU and allocations can occur in this region. Otherwise the chunk should be assumed to be on the host and allocations should be avoided. This binary grid carries little GPU memory overhead 512KB for $256^3 m^3$, and can be easily re-allocated on-the-fly to extend to larger scenes.

CHAPTER 17
Results

We implemented our data structure using DirectX 11 Compute Shaders. We use an Asus Xtion for scenes in Figure 17.5 and a Kinect for Windows camera for all other scenes, both providing RGB-D data at 30Hz. Results of live scene captures for our test scenes are shown in Figures 27.1 and 17.6 as well as supplementary material. We captured a variety of indoor and outdoor scenes under a variety of lighting conditions. While the quality of active infrared sensors is affected significantly in outdoor scenes, our system still manages to reconstruct large-scale outdoor scenes with fine quality. STATUES in Figure 27.1 shows the result after an online scan of a $\sim 20m$ long corridor in a museum with about $4m$ high statues, which was captured and reconstructed live in under 5 minutes. PASSAGEWAY (Figure 17.6 top) shows a pathway of shops $\sim 30m$ long reconstructed live. QUEENS (Figure 17.6 middle) shows a large courtyard (stretching $\sim 16m \times 12m \times 2m$) reconstructed in approximately 4 minutes. Finally, BOOKSHOP (Figure 17.6 bottom) shows three levels of a bookstore reconstructed in under 6 minutes.

These reconstructions demonstrate both scale and quality, and were all reconstructed well above the 30Hz frame rate of the Kinect as shown in Figure 17.2. This allows for potential increase of voxel resolution and additional ICP steps for more robust camera tracking. We use a voxel size of

Figure 17.1: Example output from our reconstruction system without any geometry post-processing. Scene is about $20m$ wide and $4m$ high and captured online in less than 5 minutes with live feedback of the reconstruction.

4*mm* for Figure 17.3, 17.5 and 10*mm* for Figure 27.1, 17.4, 17.6. We also tested our system with $< 2mm$ voxels without visible improvements in overall reconstruction quality. While this highlights limits of current depth sensing technology, we believe that this opens up new possibilities for future depth acquisition hardware.

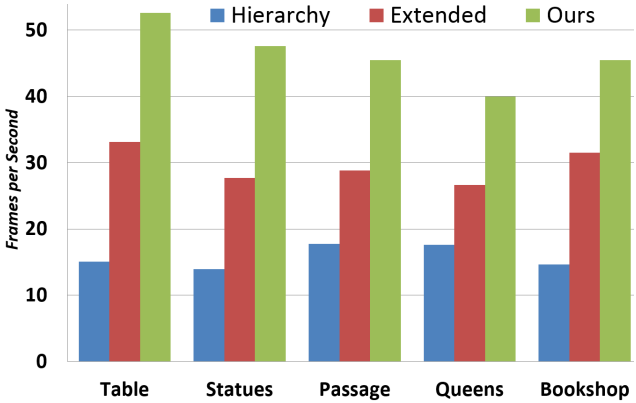


Figure 17.2: Performance comparison: frame rate measurements across our test scenes compared against two state-of-the-art reconstruction methods. *Extended* (or moving volume) regular grids and the *hierarchical* approach of [CBI13].

17.1 Performance

We measured performance of our entire pipeline including run-time overhead (such as display rendering) on an Intel Core i7 3.4GHz CPU, 16GB of RAM, and a single NVIDIA GeForce GTX Titan. Average timings among all test scenes is 21.8ms ($\sim$46*fps*) with 8.0*ms* (37% of the overall pipeline) for ICP pose estimation (15 iterations), 4.6*ms* (21%) for surface integration, 4.8*ms* (22%) for surface extraction and shading (including colored phong shading), and 4.4*ms* (20%) for streaming and input data processing. Separate timings for each test scene are provided in Figure 17.2.

Our data structure uses a total of 34MB for the hash table and all auxiliary

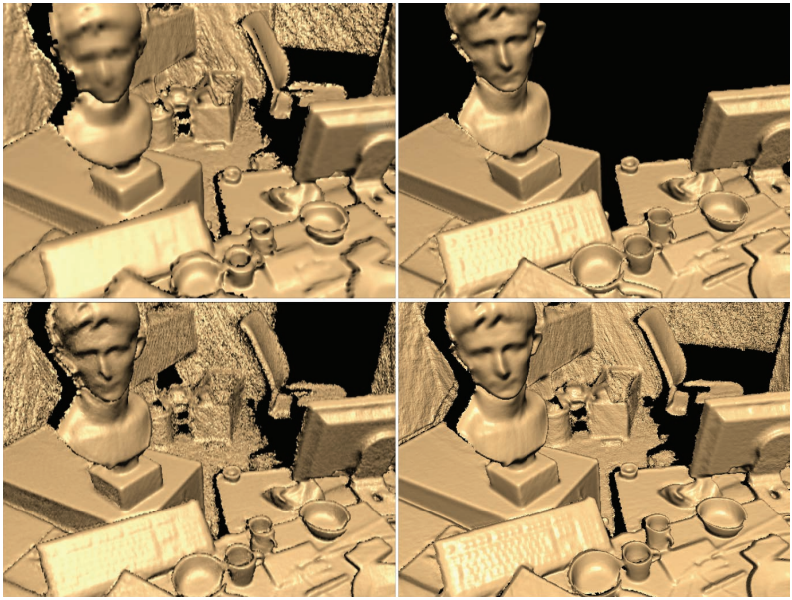


Figure 17.3: Quality and scale comparison with related systems. Bottom right: Our method maintains a large working volume with streaming at frame-rate (in this example $4mm$ voxels). Top: moving volumes based on regular grids. With the same physical extent, the voxel resolution is coarse and quality is reduced (top left), but to maintain the same voxel resolution, the size of the volume must be decreased significantly (top right). Bottom left: the performance bottleneck of hierarchical grids leads to more tracking drift, and reduces overall quality.

buffers. This allows a hash table with 2^{21} entries, each containing 12 bytes. Our experiments show that a bucket size of two provides best performance leaving us with about 1 million buckets. We pre-allocate 1GB of heap memory to provide space for voxel data on the GPU. With 8^3 voxels per block (8 byte per voxel) this corresponds to 2^{18} voxel blocks. Note that 2^{21} hash entries only index to 2^{18} voxel blocks resulting in a low hash occupancy, thus minimizing hash collisions.

On average we found that about 140K voxel blocks are allocated when cap-

turing our test scenes at a voxel size of 8mm (varying with scene complexity). This corresponds to an equal amount of occupied hash entries, resulting in a hash table occupancy with 120K buckets with a single entry, and 10K buckets with two entries. With a bucket size of two and hash table size of 2^{21}, *all* test scenes run with only 0.1% bucket overflow. These are handled by linked lists and across all scenes the largest list length is three. In total $\sim$700 linked list entries are allocated across all scenes, which is negligible compared to the hash table size.

On average less than 300MB memory is allocated for surface data (less than 600MB with color). This compares favorably to a regular grid that would require well over 5GB (including color) at the same voxel resolution (8mm) and spatial extent (8m in depth). This also leaves enough space to encode RGB data directly into the stored voxels (see Figure 17.6).

In practice this simple hashing scheme with small bucket size and large hash table size works well. In our scenario we can tolerate larger and sparser (2^{21}) hash table sizes, because the memory footprint of the hash table is insignificant ($\sim$34MB) compared to the voxel block buffer (which is pre-allocated to 1GB). Smaller hash table sizes cause higher occupancy and decrease performance. For example, in the STATUES scene our standard settings (2^{21} elements) occupies $\sim$6.4% of the hash table and runs at $\sim$21ms, with 200K elements occupancy rises to $\sim$65% and performance is reduced to $\sim$24.8ms, and with 160K elements occupancy rises to $\sim$81% with performance further falling to 25.6ms. In our live system, we chose larger table sizes as we favored performance over the small memory gains. Our pipeline currently uses atomic operations per hash bucket for allocation and streaming. As shown by our timings across all scenes, these sequential operations cause negligible performance overheads, due to hash collisions being minimal.

More sophisticated hashing approaches as [LH06, BC08, ASA*09] or the methods of [PM11, GLHL11] could reduce collisions and allow to decrease the hash tables further. However, how these methods deal with the high throughput of data, fusion and streaming is unclear. It is also important to stress that our simple hashing method works well in practice, handling scalability and quality at framerates $>$40fps across all scenes.

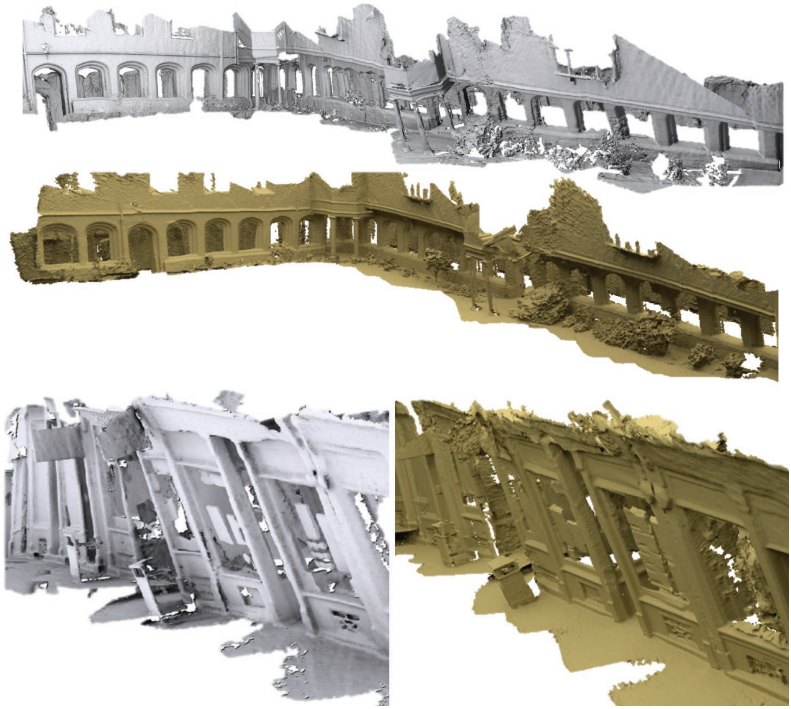


Figure 17.4: Comparison of camera tracking drift: in gray the results of the hierarchical approach of Chen et al. [CBI13] and in yellow our results. Note the twisting in the final models for Chen's approach; e.g., the center of the Queens and left hand side of the Passageway reconstruction.

17.2 Comparison

In Figure 17.4 we show the quality and performance of our method compared to previous work. All code was tested on the same hardware (see above) with a fixed number of ICP iterations (15). As our algorithm supports real-time streaming, we conducted comparisons with similar *moving volume* approaches. First, we compare against *Extended Fusion* [RV12, WJK*12] that use a regular uniform grid including streaming to scale-up

Figure 17.5: Comparison of output meshes from our online method (top) with the offline method of [ZK13] (bottom).

volumetric fusion. Second, we compare against *Hierarchical Fusion* [CBI13] that supports larger moving volumes than other approaches. Corresponding timings are shown in Figure 17.2. The most significant limitation of the hierarchy is the data structure overhead causing a performance drop, particularly in complex scenes. In our test scenes the entire hierarchy pipeline (including pose estimation, fusion, and streaming) runs at $\sim$ 15Hz, which is lower than the input frame rate. Note that these measurements are based on the reference implementation by Chen et al. [CBI13]. Our system also performs favorably compared to streaming regular grids in terms of framerate (labeled *Extended* in Figure 17.2). We attribute this to processing of empty voxels in the regular grid (particularly during random GPU memory access; e.g., raycasting) and streaming overhead.

Further, as shown in Figure 17.3, our reconstruction quality is higher than these approaches. The quality of Extended Fusion is limited by the small spatial extent of the moving volume, which means much of the Kinect data is out of range and not integrated. Hierarchical Fusion suffers from the

poor frame rate causing input data to be skipped. This severely affects pose estimation quality resulting in inaccurate surface integration and drift. In large-scale scenes this type of drift might cause unnaturally twisted models as shown in Figure 17.4.

Given our more efficient data structure, which runs faster than the Kinect camera frame rate, additional time can be spent improving the accuracy of the pose estimation by increasing the number of ICP iterations. We find our results encouraging, particularly given no drift correction is explicitly handled. In Figure 17.5 scenes captured and processed offline using the method of [ZK13], which uses a multi-pass global optimization to mitigate drift, are compared to our online method. While our method does suffer from small drifts, our system produces comparable results, and can be used for real-time applications. Our online method can also be used as a live preview, and combined with such approaches for higher-quality offline reconstruction.

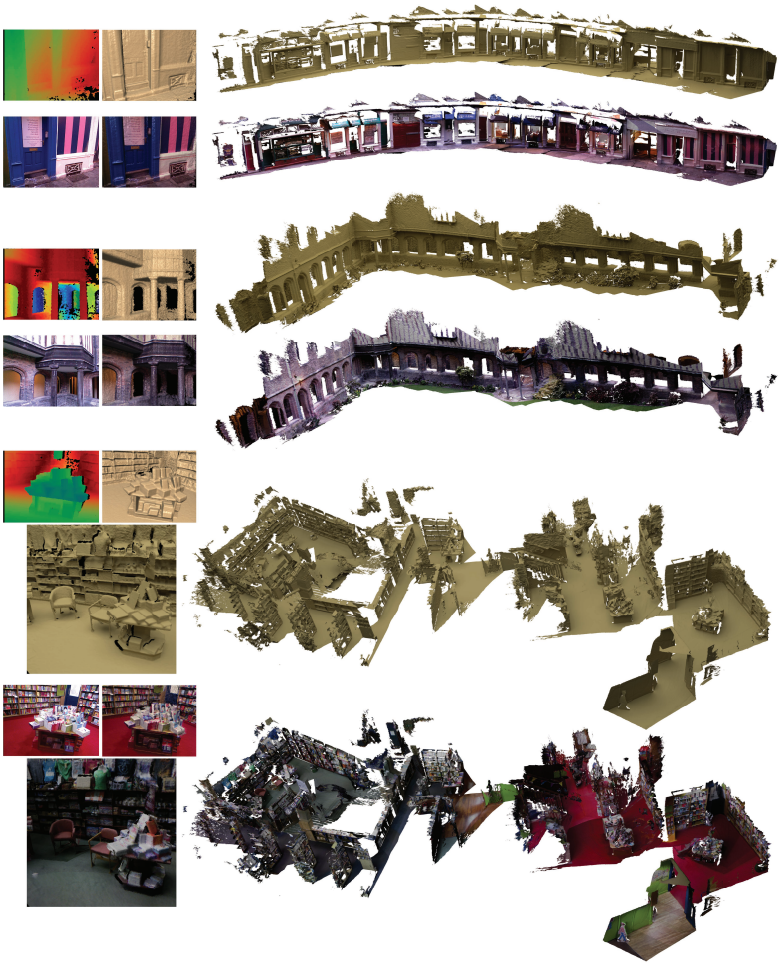


Figure 17.6: Reconstructions of the captured test scenes: a pathway of shops (passageway), a large courtyard (queens) and a three level bookstore (bookshop). Shown left: the input data from the Kinect sensor (depth and color) and the live raycasted view of our system (shaded and shaded color).

CHAPTER 18
Conclusion

We have presented a new data structure designed specifically for online reconstruction using widely-available consumer depth cameras. Our approach leverages the power of implicit surfaces and volumetric fusion for reconstruction, but does so using a compact spatial hashing scheme, which removes both the overhead of regular grids and hierarchical data structures. Our hashing scheme supports real-time performance without forgoing scale or finer quality reconstruction. All operations are designed to be efficient for parallel graphics hardware. The inherent unstructured nature of our method removes the overhead of hierarchical spatial data structures, but captures the key qualities of volumetric fusion. To further extend the bounds of reconstruction, our method supports lightweight streaming without major data structure reorganization.

We have demonstrated performance increases over the state-of-the-art, even regular grid implementations. The data structure is memory efficient and can allow color data to be directly incorporated in the reconstruction, which can also be used to improve the robustness of registration. Due to the high performance of our data structure, the available time budget can be utilized for further improving camera pose estimation, which directly improves reconstruction quality over existing online approaches.

We believe the advantages of our method will be even more evident when future depth cameras with higher resolution sensing emerge, as our data structure is already capable of reconstructing surfaces beyond the resolution of existing depth sensors such as Kinect.

PART IV

Interactive Lattice based Deformation

CHAPTER 19
Introduction

In Part III of this dissertation we have presented an approach to digitize the shape and appearance of arbitrary physical objects and scenes using just a commodity RGB-D sensor. The acquired three-dimensional models can be readily used as props in computer games, movie productions and virtual reality applications. If the virtual object only undergoes rigid motion in the application, the captured static description might be sufficient. To allow for fully articulated non-rigid motion, i.e. of a running virtual character, we have to explicitly model its time varying behavior. Thereby, the creation and use of fully animated virtual characters will be possible in our everyday

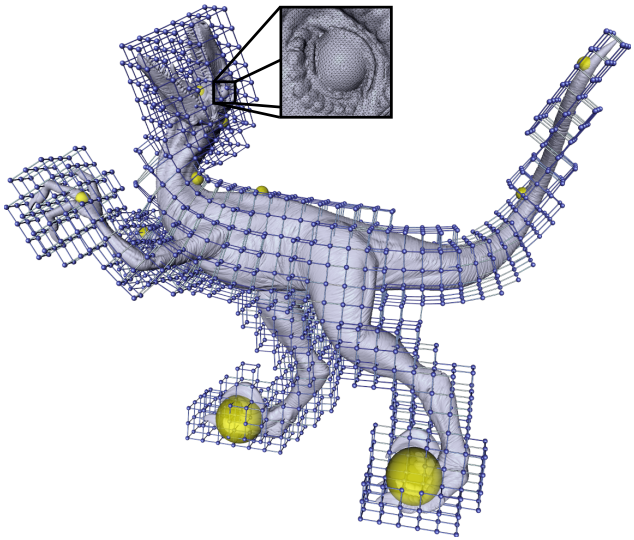


Figure 19.1: The proposed lattice based deformation approach allows to deform a raptor model with 1.7 million polygons in real-time.

life.

Modeling the time varying behavior requires the creation of all different poses the individual characters should be able to adopt. Artists usually need a lot of knowledge about the underlying deformation models to accomplish this labor-intensive task. To make the artist's life easier and allow the general public to use these techniques, current research focuses on interactive and intuitive modeling paradigms. Those give the users the possibility to directly manipulate high-resolution characters using a small number of vertex constraints. Unconstrained parts of the characters should automatically follow the user's input in real-time. A natural and physically plausible look of the deformations is of major importance for the authenticity of the generated animations. This means, that the deformations should be globally consistent and local details of the characters (e.g., the raptor's eye in Figure 27.1) should be preserved.

CHAPTER 20
Previous Work

Free-Form Deformation [SP86] allow to model space deformations by manually moving control points of a lattice. Although this scheme is conceptually simple, it is quite hard to model a specific deformation of an embedded object.

In [IMH05], the authors present a handle based approach to deform two-dimensional shapes in a distortion minimizing way. As-rigid-as-possible surface modeling (ARAP) [SA07] minimizes a surface based deformation energy to obtain detail preserving mesh edits. Because the optimization is formulated on the mesh's geometry, interactive modeling is impossible for complex meshes. The idea of the graph based deformation scheme by Sumner et al. [SSP07] is to decouple the optimization from the mesh's complexity. Primo [BPGK06] and its extension [BPWG07] are based on a decomposition of the object in rigid volumetric cells. As regularizer, the cells are connected by elastic forces. In [ZHS*05] a quadratic optimization problem is solved on a graph given extrapolated local transformations. In contrast, we use the non-linear ARAP energy which optimizes for local rotations and employ a multi-resolution GPU solver.

Most similar to our work is Hybrid Mesh Editing [BHZN10], which applies the as-rigid-as possible paradigm to an automatically generated control cage and uses mean value coordinates as transfer function. In comparison to their approach, ours is volume-aware and allows for direct manipulation. Since we deform the surrounding space, we can easily handle models with disconnected parts (e.g., the girl and the trees in Figure 22.1). In addition, we propose a data-parallel multi-resolution implementation which allows us to pose even meshes with millions of triangles in real-time.

Our lattice based as-rigid-as possible (LARAP) deformation paradigm allows artists to pose high-quality characters in an interactive and intuitive manner. We combine the well-known ARAP approach with automatically

generated control lattices to decouple the algorithm's complexity from the complexity of the characters. The regular structure of the lattice allows us to define an efficient multi-resolution approach for solving the optimization problem. To exploit the inherent parallelism, we present a highly data-parallel implementation on the GPU.

CHAPTER 21
Method

21.1 Proxy Geometry Generation

A key requirement for interactive mesh manipulation is real-time performance. To achieve this for highly-detailed meshes, we have to decouple the runtime complexity of the optimization problem from the mesh's complexity. This can be done efficiently by computing the optimal deformation on a proxy geometry and transferring it to the original mesh. While in theory arbitrary proxy geometries like cages, skeletons or scaffolds may be used, we decided to use a uniform lattice as it will allow us to quickly solve the non-linear optimization problem in a straight-forward multi-resolution way. Further on, a uniform lattice allows us to simulate solid objects, yielding volume-aware deformations. As shown in Figure 21.1, we place a uniform lattice around the input mesh and then delete all cubes that lie entirely outside the input geometry, yielding a volumetric proxy structure.

Next we need to link the mesh to the proxy geometry. A straightforward approach to link the mesh's vertices to the control lattice would be to express each vertex as tri-linear interpolation of the cube's corners that contains it. However, since this could result in artifacts, we will propose a more thorough scheme for binding the vertices to the control lattice in Section 21.3. For now, let us assume that we can express each mesh vertex $\mathbf{v}_j$

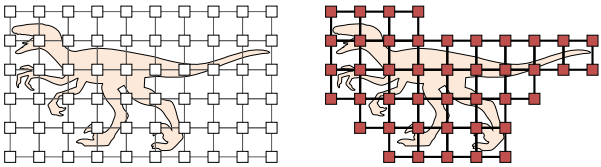


Figure 21.1: Grid generation on a uniform 6x10 voxel grid.

as a linear combination of appropriate control points $\mathbf{c}_i$ of the lattice, i.e., $\mathbf{v}_j = \sum_i \alpha_{i,j}\mathbf{c}_i$.

21.2 Modeling

In an interactive modeling session, the user first selects several vertices of the input geometry that will further serve as handles. If the handles are moved, the associated mesh vertices should move as well and the unconstrained parts of the input mesh should deform in a physically intuitive way. To obtain such a deformation, we use the *as-rigid-as-possible surface modeling* (ARAP) paradigm proposed by Sorkine and Alexa [SA07], which is based on the observation that if an object *locally* preserves its shape as good as possible *everywhere*, the object's *global* deformation will be smooth and plausible. Given an arbitrary graph $\mathcal{G}$ consisting of nodes $\mathbf{c}_i$ and edges e_{ij} in a rest pose and a deformed instance of this graph $\mathcal{G}'$ whose geometric embedding is defined by the nodes $\mathbf{c}'_i$, the ARAP energy at a node $\mathbf{c}_i$ is defined as the portion of the transformation between the local neighborhood of $\mathbf{c}_i$ and $\mathbf{c}'_i$ that cannot be represented by a rigid transformation. By defining the local neighborhood as the one-ring $\mathcal{N}_i$ of the node $\mathbf{c}_i$ and by summing over the local per-node ARAP errors, we obtain an energy function $E(\mathcal{G}, \mathcal{G}')$ that measures the plausibility of the deformation from $\mathcal{G}$ to $\mathcal{G}'$:

$$E(\mathcal{G}, \mathcal{G}') = \sum_i \sum_{j \in \mathcal{N}_i} \left\| (\mathbf{c}'_i - \mathbf{c}'_j) - \mathbf{R}_i(\mathbf{c}_i - \mathbf{c}_j) \right\|^2, \tag{21.1}$$

where the $\mathbf{R}_i$ are the rotation matrices that minimize the local deformation energies [SA07]. During interactive modeling, we seek to find the positions $\mathbf{c}'_i$ of the control lattice nodes such that the energy in Equation (21.1) is minimized -- as this will result in a natural deformation -- under the constraints that the control lattice transforms all handle vertices $\mathbf{v}_j \in \mathcal{HV}$ to the positions $\mathbf{t}_j$ defined by the user. Since we can express each vertex as a linear

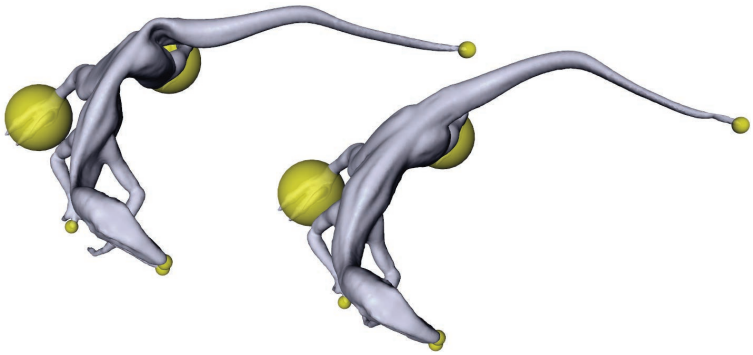


Figure 21.2: The volume-awareness of LARAP (right) prevents the surface collapsing artifacts typical for ARAP (left).

combination of lattice points, this can be stated as follows:

$$\sum_i \alpha_{i,j} \mathbf{c}'_i = \mathbf{t}_j \quad \forall \mathbf{v}_j \in \mathcal{HV}. \tag{21.2}$$

Since we may have more constraints than unknown lattice points $\mathbf{c}'_i$, the above problem may be over determined. To be able to solve the problem in general, we relax the problem and solve for the constraints in a least squares sense as well:

$$E_{\text{larap}}(\mathcal{G}, \mathcal{G}') = \gamma E(\mathcal{G}, \mathcal{G}') + \sum_{\mathbf{v}_j \in \mathcal{HV}} \left\| \sum_i \alpha_{i,j} \mathbf{c}'_i - \mathbf{t}_j \right\|^2, \tag{21.3}$$

where γ balances the influence of the regularization and the constraint term (we use $\gamma = 0.1$ for all our examples). Solving Equation (21.3) for the unknown lattice points $\mathbf{c}' = \{\mathbf{c}'_i\}$ requires solving a non-linear optimization problem in the unknowns $\{\mathbf{R}_i\}$ and $\{\mathbf{c}'_i\}$. Fortunately, similar to [SA07], the solution can be found using an iterative flip-flop optimization, where in one step the grid points are kept fixed and the energy term is minimized for the unknown rotations $\{\mathbf{R}_i\}$ using SVD (see [SA07] for details). Then we solve for the grid points $\{\mathbf{c}'_i\}$ that minimize the energy in Equation (21.1)

for fixed $\{\mathbf{R}_i\}$. Since E_{larap} is quadratic in the $\{\mathbf{c}'_i\}$, the optimal grid points can be found by solving the linear system

$$(\gamma\mathbf{L} + \mathbf{B}^T\mathbf{B}) \cdot \mathbf{c}' = \gamma\mathbf{b} + \mathbf{B}^T\mathbf{t},$$

where $\mathbf{L}$ is the uniform Laplacian of the lattice, $\mathbf{B}$ is a matrix containing the constraints from Equation (21.2) as rows, $\mathbf{t}$ is the vector containing the handle positions and $\mathbf{b}$ is a vector whose ith row is $\sum_{j\in\mathcal{N}_i} \frac{\mathbf{R}_i+\mathbf{R}_j}{2}(\mathbf{c}_i - \mathbf{c}_j)$. Since the weights are local, the matrix $\mathbf{B}$ is sparse. Thus the system can be solved efficiently using a sparse solver for semi-definite systems. By iteratively solving for the $\{\mathbf{R}_i\}$ and then for the $\{\mathbf{c}'_i\}$, interactive modeling is usually possible with 3-8 of these steps.

21.3 Preliminary Results

When comparing our method to the original ARAP algorithm, one apparent advantage of our approach is that we decouple the complexity of the deformation from the tessellation of the input geometry, which allows us to deform high quality production meshes in real time. Furthermore, since ARAP only aims at preserving the surface, unnatural folds may occur when bending the surface (Figure 21.2). We can easily prevent such artifacts by using a solid cube lattice.

Using simple tri-linear weights (i.e., encoding each vertex with respect to the surrounding lattice cell) for transferring the deformation from the control lattice onto the input geometry results in a piecewise linear deformation field which is only C^0 continuous across cells as can be seen in Figure 21.3. While this is usually sufficient for real-time editing, generating the final high quality poses requires a more elaborate weighting scheme. Therefore, we propose to use quadratic B-Spline weights when exporting the manipulated models, as this results in a C^1 continuous deformation field. Each vertex is then encoded with respect to the 27 grid points of the surrounding cells. Note that using B-Spline weights requires all cubes that contain a vertex to have a complete set of neighboring cubes. This can be guaranteed

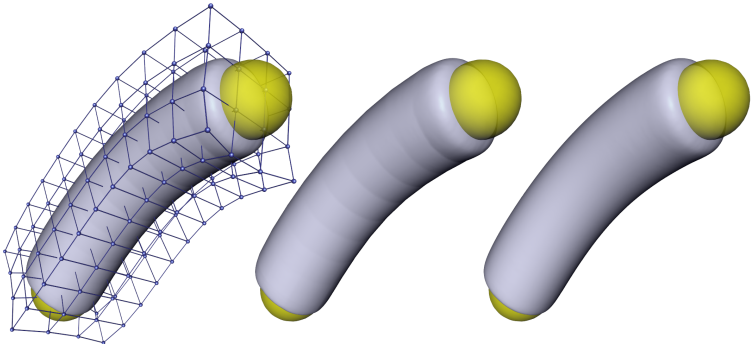


Figure 21.3: Comparison between tri-linear (middle) and B-Spline (right) interpolation.

by dilating the set of marked voxels during cage generation.

21.4 GPU based Implementation

The decoupling of the optimization problem from the mesh's complexity already results in an enormous speed-up. However, a closer analysis of the proposed algorithms reveals that many parts of the presented algorithms can be computed entirely in parallel -- namely the computation of the SVDs and right-hand sides for each lattice point and the interpolation for each model vertex. We utilized this observation and implemented the entire algorithm on the GPU using CUDA. In each flip-flop iteration, we first compute the optimal rotations for each control point's one-neighborhood in parallel. Then we update the right-hand side using the newly computed rotations. In the last step of the flip-flop iteration, we compute new positions for the lattice's control points using our parallel linear solver on the GPU. Because of the sequential dependence of these three steps, we have to synchronize between the corresponding kernel calls. We solve the linear system using either a parallel Gauss-Seidel like solver or a parallel gradient descent. New improved positions of neighbouring control points are used as soon as they

Properties			Cpu			
Model	Polygons	Control Points	SVD/RHS	Solve	Interpolation	$\sum$
Raptor	17k	10k	9.2	13.4	0.6	71.2
Raptor	170k	10k	9.4	12.6	6.6	75.6
Raptor	1.7M	10k	9.4	13.3	68	142
Dragon	2M	5k	6.1	7.7	79	122
Dragon	2M	20k	28	43	85	303
Dragon	2M	40k	53	85	84	504

Table 21.1: Timings: CPU implementation of our deformation paradigm (in *ms*).

Properties			Gpu			
Model	Polygons	Control Points	SVD/RHS	Solve	Interpolation	$\sum$
Raptor	17k	10k	1.2	11.7	0.6	44
Raptor	170k	10k	1.2	11.4	0.8	44
Raptor	1.7M	10k	1.2	11.2	1.7	45
Dragon	2M	5k	0.9	7.3	1.7	31
Dragon	2M	20k	1.5	30.0	2.0	103
Dragon	2M	40k	2.0	68.6	2.4	222

Table 21.2: Timings: GPU implementation of our deformation paradigm (in *ms*).

are available. This depends on the scheduling of the threads on the GPU. After a user-defined number of flip-flop iterations has been performed, we use an interpolation kernel to transfer the deformation of the lattice onto the input geometry.

The performance of the non-linear LARAP optimization can be improved even more by solving the problem in a multi-resolution manner. Starting from the finest control lattice, we create a hierarchy of lattices by always joining 8 adjacent cubes. Each lattice is then encoded w.r.t. the next coarser cage. We first solve for the deformation of the coarsest cage, transfer it onto the next finer cage and use the resulting positions as the starting point for another LARAP optimization, and so on. All these operations are performed in a data-parallel manner on the GPU, we can even map the deformed model directly to the rendering pipeline.

CHAPTER 22
Results

We tested our algorithm on the different (multi-part, polygon soup, high detail) models shown in Figures 27.1 and 22.1. A summary of computation times on the CPU and the GPU is given in Table 21.1 and 21.2. All timings were measured on a Core i7 860 CPU (using 8 threads) with an NVidia GeForce 580 GPU. Note that the timings refer only to solving on the finest hierarchy level. $\sum$ denotes the total time for solving the non-linear optimization (3 flip-flop steps with 800 Gauß-Seidel like iterations each, data transfer and interpolation).

When using the multi-resolution solver, we use 3 flip-flop iterations with

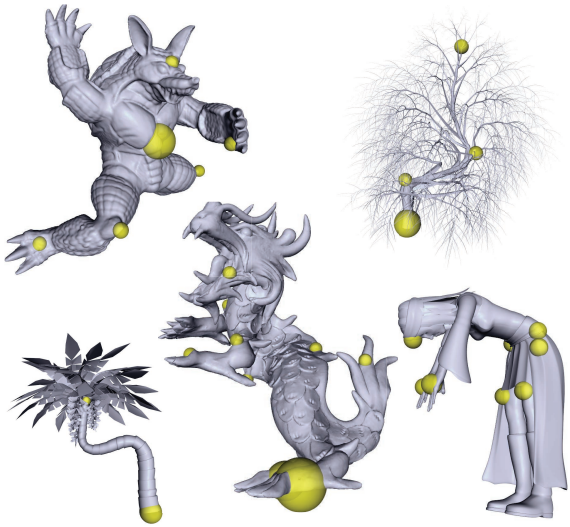


Figure 22.1: Poses generated using our interactive mesh deformation tool. The girl and the tree models contain multiple unconnected components.

200 Gauß-Seidel like iterations on each hierarchy level. Multi-resolution solving takes in total 71ms for the 2M faces Dragon model using a cage with 40k cubes in the finest lattice, which is another 300% speedup.

CHAPTER 23
Conclusion

We introduced LARAP, a novel paradigm for interactive and intuitive mesh editing. Using a simple lattice as proxy geometry decouples the algorithmic complexity from the mesh's geometric complexity. Since the algorithm is based on the simple ARAP optimization loop, it is also easy to implement. In combination with the proposed data-parallel multi-resolution implementation of the non-linear solver, we can interactively deform even high-quality meshes.

In the future, we plan to construct the lattice hierarchy in a topology preserving way (i.e., avoid that cubes that were not connected at a finer level are merged) as this will decouple the parts of the model that have a small Euclidean but large geodesic distance in coarse resolution lattices. Additionally, by using an octree and monitoring the deformation error, we plan to locally solve the optimization problem only up the resolution on which the deformation error vanishes.

PART V

Real-Time Tracking of Deforming Objects

CHAPTER 24
Introduction

In Part IV of this dissertation, we have shown that a user can easily breath life into a static three-dimensional model using the presented interactive and intuitive deformation approach. Despite its good usability and the real-time preview of the modeled poses, manually creating all poses of a complex animation sequence is still a tedious and time consuming task. This overhead can be drastically reduced by using an RGB-D sensor to also capture the object's time varying behavior. Unfortunately, the captured RGB-D stream contains no information about space-time correspondences between the captured frames. If we want to extract an object's animation sequence given such data, we have to track the object's non-rigid motion through time. This will allow us to store the object's motion in a consistent and compact manner.

In addition, the ability to reconstruct the fine-grained non-rigid motions and shape of physical objects in a live and temporally consistent manner opens up many new applications. For example, in real-time, a user can re-target their motions and detailed expressions to avatars for gaming or video conferencing. An actor's performance and motions can be captured online for live feedback and preview. By reconstructing the detailed motion and shape of surfaces, systems can overlay digital content onto the physical world in more convincing ways; e.g., for virtual clothing, makeup, and other augmented reality applications. Finally, deforming physical objects can become props for digital interaction, further bridging the gap between real and virtual worlds.

Despite considerable advances in the field of non-rigid tracking and reconstruction, there has been limited work on real-time techniques that work on general scenes. In special cases such as hands, faces or full bodies, researchers have demonstrated compelling real-time reconstructions of non-rigid articulated motion [OKA11, TSSF12] and shape [WBLP11, CWLZ13].

However, these rely on strong priors based on either pre-learned statistical models, articulated skeletons, or morphable shape models, prohibiting capture of general scenes. Reconstruction techniques that can handle more general scenes are far from real-time in terms of performance, and need seconds to hours to compute a single frame [HVB*07, LZW*09, LAGP09].

We present the first real-time reconstruction system capable of capturing a variety of non-rigid shapes and their deformations. As demonstrated, our entire pipeline from depth acquisition to non-rigid deformation runs at 33ms per frame, orders of magnitude faster than state-of-the-art methods, while achieving reconstruction quality and robustness that approach offline methods with more complex sensor setups.

Our system is markerless, uses a single self-contained stereo camera unit, and consumer graphics hardware. The stereo camera is built from off-the-shelf components and uses a new stereo matching algorithm to generate RGB-D images, with a greater degree of flexibility and accuracy than current consumer depth cameras. Using this camera, a smooth template model of the rigidly moving subject is acquired online. This acts as a geometric and topological prior for non-rigid reconstruction, but avoids strong assumptions about the scanned scene, such as a kinematic skeleton or a parametric shape model (e.g., for faces, hands or bodies) that would limit the generality of our system. Then, for each live RGB-D frame, a novel GPU pipeline performs non-rigid registration to the acquired template model with an as-rigid-as-possible (ARAP) regularizer and integrates detail using a thin shell deformation model on a displacement map [SA07].

We show precise real-time reconstructions, including: large deformations of users' heads, hands, and upper bodies; fine-scale wrinkles and folds of skin and clothing; and non-rigid interactions performed by users on flexible objects such as toys. We demonstrate how acquired models can be used for many interactive scenarios, including re-texturing, online performance capture and preview, and real-time shape and motion re-targeting.

CHAPTER 25
Related Work

The emergence of depth cameras, such as the Kinect, has spawned new interest in *real-time* rigid 3D scanning as exemplified by systems such as KinectFusion [NIH*11, IKH*11] or the method presented in Part III of this dissertation. It is therefore a natural next step to think about online capture of non-rigid scenes using RGB-D cameras. Interestingly, follow-up work based on KinectFusion specifically focused on scanning humans (e.g., for 3D printing or generating avatars) where the user rotates in front of the Kinect while maintaining a roughly rigid pose, these include [WHB11, LVG*13, TZL*12, ZZCL13, HBB*13]. Similar to [BR07, WWL*09], in these *offline* systems non-rigid registration techniques are employed to accommodate for small deviations in the motion between different viewpoints. These systems are motivated by producing a *single* mesh as output from an RGB-D sequence, whereas we wish to continuously reconstruct non-rigid motions during live capture.

Many multi-camera techniques for non-rigid reconstruction of geometry and motion have been proposed. Some are specifically motivated by modeling complex human motion and dynamic geometry, including people with general clothing, possibly along with pose parameters of an underlying kinematic skeleton (see [TdAS*10] for a full review). Some methods employ variants of shape-from-silhouette [WWC*05] or active or passive stereo [SH07]. Model-based approaches deform a static shape template (obtained by a laser scan or image-based reconstruction) such that it matches a human [dAST*08, VBMP08, GSDA*09] or a person's apparel [BPS*08]. Vlasic et al. [VPB*09] use dynamic photometric stereo in a sophisticated controlled light stage dome with multiple high-speed cameras to capture temporally incoherent geometry of a human at high detail. In the work of Dou et al. [DFF13] precise surface deformations are captured using an eight-Kinect rig, by deforming a human template, generated from a KinectFusion scan, using embedded deformation [SSP07]. Other methods jointly track

a skeleton and the non-rigidly deforming surface [VBMP08, GSDA*09], while some treat the template as a generally deformable shape without skeleton and use volumetric [dAST*08] or patch-based [CBI10] deformation methods.

These multi-camera approaches have runtime performances far from real-time, and require dense camera setups in controlled studios, with sophisticated lighting and/or chroma-keying for background subtraction. However, with the availability of consumer depth cameras, lightweight camera setups have been proposed in [HVB*07, LZW*09, LAGP09] and [WBLP11, VWB*12, CIF12, WSVT13, GVWT13]. Ye et al. [YLH*12] capture multi-person performances with three moving Kinects. Furthermore, in special cases, such as for hands, faces and full bodies, researchers have demonstrated compelling *real-time* reconstructions of articulated motion [OKA11, WZC12, TSSF12] and/or non-rigid shape and motion [WBLP11, CWLZ13, HBB*13]. However, these rely on strong priors based on either an offline learned model [TSSF12], an articulated skeleton [OKA11] or morphable shape model [BV99, WBLP11, HBB*13, CWLZ13], which prohibits capture of general scenes. Additionally, these real-time methods are unable to reconstruct high-frequency shape detail obtained with state-of-the-art offline approaches.

The approach of [LAGP09] uses a coarse approximation of the scanned object as a shape prior to obtain high quality non-rigid reconstructions. Other non-rigid techniques do not require a shape or template prior, but assume small and smooth motions [ZZCL13, WAO*09, MFO*07]; or deal with topology changes in the input data (e.g., the fusing and then separation of hands) but suffer from drift and oversmoothing of results for longer sequences [TBW*12, BHLW12]. These more general techniques are far from real-time, ranging from seconds to hours to compute a single frame.

Our system attempts to hit a `sweet spot' between methods that can reconstruct general scenes, and techniques that rely on a stronger shape prior (e.g., a blendshape face model or body or hand skeleton), which are beginning to demonstrate real-time performance. To our knowledge, our system is the first that provides real-time performance, several orders of magnitude

faster than general methods, but does not require a specific `baked in' kinematic or shape model. Instead our system allows users to acquire a template online, and use this for *live* non-rigid reconstructions. Our system is simple to use and self-contained, with a single lightweight stereo camera setup, moving closer to commodity or consumer use. Additionally, in terms of reconstructed geometry detail, our method also narrows that gap between offline and online methods. This simplicity and real-time performance however does not come at a significant cost of reconstruction quality (as shown in the results section), and brings us a step closer to high-quality real-time performance capture systems for consumer scenarios, including gaming, home and semi-professional movie and animation production, and human-computer interaction.

The specific contributions of this work are:

- A general, real-time, non-rigid reconstruction pipeline, non-trivially realized on the GPU. While in the spirit of previous non-rigid reconstruction frameworks, in particular Li et al. [LAGP09], our GPU pipeline is orders of magnitude faster, with many algorithmic and implementation differences.
- The creation of a fully automatic real-time non-rigid capture system. This allows novice users to quickly generate template models of arbitrary objects, whose motions and non-rigid deformations can be captured with live user feedback.
- An interactive application of our non-rigid reconstruction pipeline that demonstrates spatio-temporal coherent models for motion and shape re-targeting in video games, performance capture, and augmented reality.
- A lightweight visible light and infrared (IR) stereo camera setup for generating compelling RGB-D input at real-time rates, which allows us to capture higher quality RGB-D data at closer ranges than consumer depth cameras.

CHAPTER 26
Method

26.1 System Overview

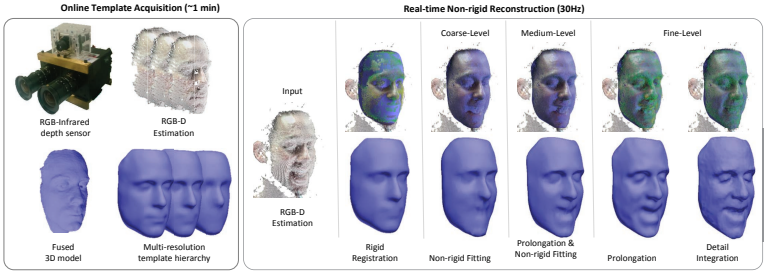


Figure 26.1: Main system pipeline. **Left:** the initial template acquisition is an online process. Multiple views are volumetrically fused, and a multi-resolution mesh hierarchy is precomputed for the tracking phase. **Right:** in the tracking phase, each new frame is rigidly registered to the template, and a sequence of calls to the GPU-based Gauss-Newton optimizer is issued from coarse to fine mesh resolution. At the finest resolution, detail is integrated using a thin-plate spline regularizer on the finest mesh.

Our system is designed to deal with close range non-rigid reconstructions of single objects, such as faces, hands, upper bodies, or hand held physical objects. The general usage scenario is illustrated in Figure 27.1, and the system pipeline in Figure 26.1, and comprises two phases: online template acquisition and real-time non-rigid reconstruction.

The first part of the pipeline is a *online template acquisition* phase that takes $\sim$1 minute to perform. The user sits or stands in front of our custom RGB-D sensor (up to 1½ meters away from the sensor). First, the desired object is scanned while undergoing mostly rigid deformations. Immediate feedback is provided during scanning using the volumetric fusion frame-

work presented in Part III of this dissertation, from which a triangle mesh model is automatically extracted. The mesh is preprocessed to create a multi-resolution hierarchy to be used in the online phase.

The second phase of our pipeline performs *real-time non-rigid reconstruction*, which produces a deformed mesh at every time step, executing the following three steps at every frame:

1. **Rigid registration** roughly aligns the template to the input data.
2. **Non-rigid surface fitting** by minimization of a fitting energy which combines dense geometric and photometric constraints, as well as an as-rigid-as-possible (ARAP) regularizer. The energy is minimized using a new efficient GPU-based Gauss-Newton solver using the preconditioned conjugate gradient method (PCG) in its inner loop. This solver is applied in a coarse-to-fine manner, using a multi-resolution mesh hierarchy prepared at template acquisition. At each level, the fitting energy is optimized at the current resolution using several iterations of Gauss-Newton, and then a *prolongation* step interpolates the solution to the next finer level.
3. **Detail integration** at the finest template level: a thin-shell deformation energy under model-to-data constraints is minimized by solving a linear least squares system for displacements along the model normal at each vertex.

These components are now explained in detail after a description of our custom stereo sensor.

26.2 Lightweight Active Stereo Sensor

For acquisition, we designed a new *RGB-IR* stereo rig which reconstructs pixel synchronized *RGB-D data* in real-time. The use of a custom depth camera provides us with a greater deal of flexibility than consumer sensors. In particular, our specific scenario requires close range capture at high quality, and most existing sensors are limited in these terms. For comparison,

in Section 27 we will also present results of our system running with a consumer Kinect camera.

The sensor comprises a fully-calibrated pair of video cameras of resolution 1024×768 providing both RGB and IR images with the same center of projection (by employing a beam splitter). For high-quality depth computation, we employ Kinect-type infrared emitters in order to project a suitable pattern onto the surface to be reconstructed. In contrast to the Kinect sensor in our setup, the emitters are not calibrated with respect to the cameras and can be placed freely to maximize the coverage of the emitted pattern in the scene. For further technical details see the supplemental material.

For real-time depth acquisition we use a patch-match based stereo algorithm inspired by [BRR11], but in analogy to [PRI*13] we reduce the search space complexity significantly by not estimating local surface normals. Thus, the only unknown to be determined at pixel p (at location (u_p, v_p)) is the scene depth D_p. Further, we deviate from the original propagation schedule of patch-match stereo to achieve better GPU utilization. Our patch-match stereo algorithm can be summarized as follows: the starting phase to initialize the depth map with random samples is followed by four (left-to-right, top-to-bottom, right-to-left, and bottom-to-top) propagation steps, where the matching score of the current depth hypothesis is compared with the one of the respective neighboring pixel, and the better scoring depth value is retained. This propagation strategy allows all rows (or columns) of the image to be easily processed in parallel in the GPU implementation. We use the zero-mean normalized cross-correlation (ZNCC) computed over 7×7 windows as a matching score to assess the similarity of image patches. The advantage of our patch-match stereo implementation is its high speed (100Hz to estimate 1024×768 depth images), but all patch-match inspired algorithms produce piece-wise constant outputs with an undesired ``blocky'' appearance. Consequently, we refine the raw depth map produced by our patch-match stereo method using a variational approach, which combines the (local) matching score profile with a global smoothness prior as follows: if we denote the depth map returned by patch-match

stereo as $\widehat{D}_p$, then we minimize

$$\mathcal{E}(\mathbf{D}) = \sum_p \alpha_p \left(D_p - \widehat{D}_p\right)^2 + \sum_{(p,q)\in E} \omega_{pq} \left(D_p - D_q\right)^2 \tag{26.1}$$

with respect to the depth values $\mathbf{D} = [D_1, ..., D_{WH}]$, with E the set of 4-neighbor pixel pairs. We model the local behavior of the matching scores near $\widehat{D}_p$ using a quadratic model, i.e., we fit a quadratic function to the matching scores of $\widehat{D}_p - 1$, $\widehat{D}_p$, and $\widehat{D}_p + 1$. This determines the coefficient α_p. In general, the local quadratic model of matching scores should be a convex parabola, i.e., $\alpha_p > 0$, if $\widehat{D}_p$ is at a (local) minimum. We avoid a non-sensible concave parabola fitted to the matching scores by setting $\alpha_p = 0$ in these cases. Our regularizer prefers smooth depth maps, but we avoid smoothing over depth discontinuities by using a contrast-aware regularization term, i.e., we introduce weights $\omega_{pq} \in [0, 1]$ for neighboring pixels p and q, which are based on strong color edges in the RGB images, $\omega_{pq} = 1/(1 + \beta\|\nabla I^L(u_p, v_p)\|)$. Here $I^L(\cdot)$ denotes the left color image, and β is a tuning parameter always set to 20. The objective in Equation 26.1 is quadratic in the unknowns $\mathbf{D}$, and we use a GPU-implemented successive over-relaxation (SOR) solver to obtain the refined depth values. Since we are only interested in estimating and retaining depth for foreground objects, we utilize a simple, color-based background subtraction step to discard depth values corresponding to undesired background.

Our active sensor has a variety of advantages over existing real-time scanners or low-cost depth cameras. The use of active (infrared) illumination allows high-quality shape acquisition without solely relying on the object's texture (as in passive stereo) or distorting the color image (as in some fringe-based techniques). Our stereo setup allows the baseline between the cameras to be modified easily in order to adapt to the observed volume of interest. Changing the baseline in our setup requires a standard geometric calibration procedure to determine the new relative pose between the cameras. This is in contrast to the Kinect camera, which has a fixed baseline and would require a more difficult projector-camera calibration if the baseline is changed. Compared to time-of-flight cameras, it features a much higher

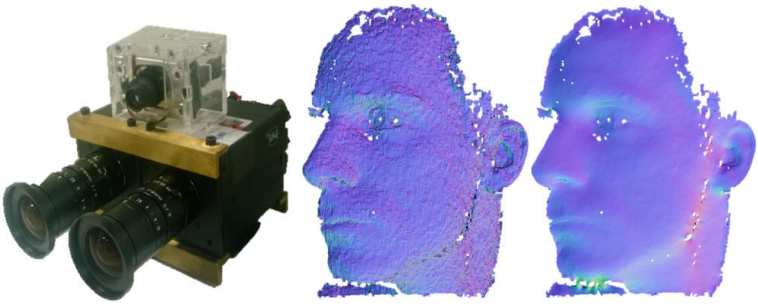


Figure 26.2: Left: our active stereo sensor. Middle: patch-match result. Right: Result after variational refinement.

depth and image resolution and does not suffer from their well-known systematic data distortions due to light modulation, reflectance dependencies and multi-path light transport [KBKL09]. We employ a prototype setup built from standard vision cameras and do not have the same small form factor as mass-manufactured depth cameras. However, in mass production similar form factors and production cost could be achieved while maintaining its technical advantages.

26.3 Surface Tracking as Model Fitting

Our template model is a hierarchy of triangle meshes (typically three levels; see Figure 26.1) where vertices of a finer level are connected by a space deformation to the next coarser level [SSP07]. This connection is used to apply the prolongation operator (see Section 26.3.2). The hierarchy levels are computed through a series of mesh simplification and Laplacian smoothing steps. Note that the transition between template capture and non-rigid tracking is fully automated and seamless, requiring a few seconds to execute.

Each triangle mesh is defined by n vertices $\mathrm{V}^0 = \{\mathbf{v}_i^0 \in \mathbb{R}^3 \mid i = \{1, \ldots, n\}\}$ and m edges. The mesh topology is constant during tracking, and is queried

only via the sets $\mathcal{N}_i$, which hold the indices of vertices sharing an edge with vertex *i*. This allows the use of non-manifold meshes, and indeed we use internal edges on some sequences to add a weak form of volume preservation; i.e., we tetrahedralize the template interior. Any internal vertices have visibility flags (see below) permanently zeroed.

26.3.1 Energy Function

Our goal in surface tracking is to determine, at time *t*, the 3D positions of the model vertices $\mathrm{V}^t = \{\mathbf{v}_i^t\}_{i=1}^n$, and global rotation and translation $\mathbf{R}^t$, $\mathbf{t}^t$. This will be achieved by running a Gauss-Newton solver on a suitable energy function, using the values $(\mathrm{V}^{t-1}, \mathbf{R}^{t-1}, \mathbf{t}^{t-1})$ from the previous time step as an initial estimate. As each frame is processed otherwise independently, the *t* superscripts are dropped below.

We are given as input a depth image $\mathbf{d}$ which maps 2D points $\mathbf{u}$ to 3D world points using the sensor output and the known camera calibration information, so $\mathbf{d}(\mathbf{u}) \in \mathbb{R}^3$. We also have data normals $\mathbf{n} : \mathbb{R}^2 \mapsto \mathbb{R}^3$ computed by Sobel filtering on the depth map. These images are evaluated at non-integer locations using bilinear interpolation, and the derivatives $\nabla_{\mathbf{u}}\mathbf{d}(\mathbf{u})$ and $\nabla_{\mathbf{u}}\mathbf{n}(\mathbf{u})$ are therefore also well-defined.

The energy function is a sum of data terms, which encourage every visible model vertex to be as close as possible to the sampled data, and regularizers which control the smoothness of deformations and motion. **Visibility** is defined by a variable η_i associated with each model vertex, and is determined statically before each Gauss-Newton solve by rendering the model under the current parameters. In practice this computes correct visibilities for the majority of data points and a robust kernel in the energy handles the remainder.

Data Terms

The **data term** measures the distance to the closest data point, and is written with an explicit minimization over the corresponding 2D image position $\mathbf{u}$:

$$\mathcal{E}_{\text{point}}(\mathrm{V}) = \lambda_{\text{point}} \sum_{i=1}^{n} \eta_i \min_{\mathbf{u}} \psi\left(\frac{\|\mathbf{v}_i - \mathbf{d}(\mathbf{u})\|}{\sigma_d}\right), \tag{26.2}$$

where σ_d is an estimate of sensor noise, and ψ is a robust kernel similar to Tukey's biweight, but with additional properties, described below. In practice, including the closest-point search within the energy gives a complicated energy surface, so we ``lift'' the inner minimizers to become search parameters $\mathrm{U} = \{\mathbf{u}_i\}_{i=1}^{n}$

$$\mathcal{E}_{\text{point}}(\mathrm{V}, \mathrm{U}) = \lambda_{\text{point}} \sum_{i=1}^{n} \eta_i \psi\left(\frac{\|\mathbf{v}_i - \mathbf{d}(\mathbf{u}_i)\|}{\sigma_d}\right). \tag{26.3}$$

Note that this is exact: $\mathcal{E}_{\text{point}}(\mathrm{V}) = \min_{\mathrm{U}} \mathcal{E}_{\text{point}}(\mathrm{V}, \mathrm{U})$. In essence, we are trading complexity of the energy surface for a $5/3$-fold increase in the number of unknowns. It does *not* imply an iterated closest point strategy of alternating minimization over V and U, which is known to have poor convergence properties. Rather, it suggests a simultaneous optimization over all unknowns, which is particularly important near the optimum. Also, as will be shown below, the new unknowns lead to a simple and sparse augmentation of the system Jacobian, so that optimization runtime is only very mildly affected by the increase in problem size. Although one could use the values from the previous timestep as an initial estimate for U, a more effective strategy is described in Section 26.3.3.

To further improve the properties of the energy, we adopt the common strategy of including a point-to-plane term

$$\mathcal{E}_{\text{plane}}(\mathrm{V}, \mathrm{U}) = \lambda_{\text{plane}} \sum_{i=1}^{n} \eta_i \psi\left(\frac{\mathbf{n}(\mathbf{u}_i)^{\top}(\mathbf{v}_i - \mathbf{d}(\mathbf{u}_i))}{\sigma_n}\right) \tag{26.4}$$

which allows incorrectly assigned correspondences to ``slide'' along the sur-

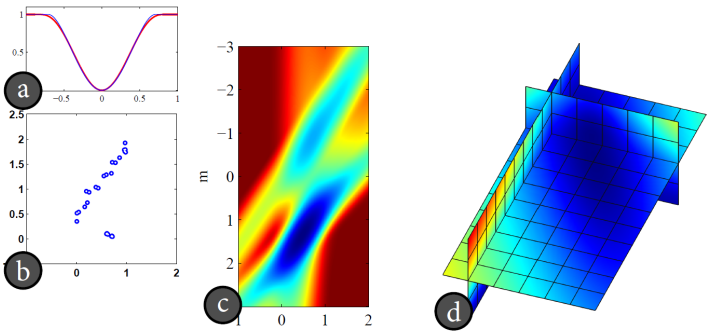


Figure 26.3: Robust kernel (Section 26.3.1). (**a**) Our kernel $\psi(e)$ (blue) has similar shape to the standard Tukey's biweight kernel (red). (**b**) A 2D line fitting problem with two minima. Data points $y_i \approx mx_i + c$. (**c**) Energy landscape of $f(m, c) = \sum_i \psi(y_i - mx_i - c)$ is complicated. (**d**) 3D slice through $(2 + n)$ dimensional landscape of lifted function $F(m, c, w_1, \ldots, w_n) = \sum_i w_i^2(y_i - mx_i - c)^2 + (1 - w_i^2)^2$ is simpler. Minimization of lifted F found the global optimum on 82.4% of runs, in contrast to 43.0% on two-parameter f, which also had 20.1% outright failures vs. 0% on lifted.

face, thus improving convergence. Again, σ_n is a noise level estimate.

We further encourage vertices to preserve their RGB appearance from frame to frame with a color term

$$\mathcal{E}_{\text{color}}(\mathrm{V}) = \lambda_{\text{color}} \sum_{i=1}^{n} \eta_i \psi\left(\frac{\|\mathbf{I}_i - \mathbf{I}(\pi(\mathbf{v}_i))\|}{\sigma_c}\right), \tag{26.5}$$

where π is the projection from 3D to image coordinates, known from camera calibration, and $\mathbf{I}_i = \mathbf{I}^{t-1}(\pi(\mathbf{v}_i^{t-1}))$ is a color attached to each vertex from the previous timestep. In our implementation only the intensity channel is used. Here σ_c is the noise level of the RGB sensor.

Robust Kernel

A further amelioration of the energy surface is obtained by rewriting the non-convex robust kernel $\psi(e)$ using a similar ``lifting'' technique as was used for the correspondences. As the wide variety of published robust kernels might indicate, the precise shape of ψ is not of great importance, but it should typically have a standard form: linear or quadratic for small e values, reducing to linear or constant for larger e. We observe that the function

$$\psi(e) = \min_w \left(\frac{2w^2e^2}{\tau^2} + (1-w^2)^2 \right) = \begin{cases} \frac{e^2}{\tau^2}(2 - \frac{e^2}{\tau^2}) & \text{if } e^2 < \tau^2 \\ 1 & \text{otherwise} \end{cases}$$

has the shape in Figure 26.3(a), which has the required properties. τ is a width parameter, normally set to 1. Applying this in our framework again uses the lifting trick, so that terms of the form

$$E(\Theta) = \sum_i \psi(f_i(\Theta)) = \sum_i \min_w \left(2w^2 f_i(\Theta)^2 + (1-w^2)^2 \right)$$

become, when lifted to depend on parameters $\mathrm{W} = \{w_i\}_{i=1}^n$

$$E(\Theta, \mathrm{W}) = 2\sum_i w_i^2 f_i(\Theta)^2 + \sum_i (1-w_i^2)^2. \tag{26.6}$$

Again, the number of parameters increases, but the error function is more amenable to optimization (see also Figure 26.3). Note that this is the same weighting used in [LSP08], but the connection to robust estimation was not made there. We avoid introducing separate W vectors for each energy term by applying the robust kernel to sums of terms per vertex. That is, we replace

$$\sum_{\alpha \in \{\text{point,plane,color}\}} \lambda_\alpha \sum_i \psi(f_i^\alpha(\Theta)) \rightarrow \sum_i \psi\left(\sqrt{\sum\nolimits_\alpha \lambda_\alpha f_i^\alpha(\Theta)^2} \right)$$

where the f^α are the residual terms in (26.3), (26.4), (26.5), so the robust kernel is applied to the sum of the squared residuals, not to each separately.

Shape Regularizer

The geometric prior term $\mathcal{E}_{\text{reg}}$ forces local deformations of the surface to be as close possible to isometry, and thus approximates elastic deformation behavior. The as-rigid-as-possible (ARAP) framework [SA07] measures deformation between a pair of meshes $\mathrm{V}, \hat{\mathrm{V}}$ as follows

$$\begin{aligned} D(\mathrm{V}, \hat{\mathrm{V}}) &= \min_{\mathbf{R}_1,\ldots,\mathbf{R}_n} \sum_{i=1}^{n} \sum_{j \in \mathcal{N}_i} \|(\mathbf{v}_i - \mathbf{v}_j) - \mathbf{R}_i(\hat{\mathbf{v}}_i - \hat{\mathbf{v}}_j)\|^2 \\ &= \min_{\mathcal{R}} D_{\text{ARAP}}(\mathrm{V}, \hat{\mathrm{V}}, \mathcal{R}) \end{aligned} \tag{26.7}$$

In our energy, ARAP controls the deformation of the shape in the current frame from the rigidly transformed initial template $\mathbf{R}\mathrm{V}^0 + \mathbf{t}$ as follows:

$$\mathcal{E}_{\text{reg}}(\mathrm{V}, \mathcal{R}, \mathbf{R}, \mathbf{t}) = \lambda_{\text{reg}} D_{\text{ARAP}}(\mathrm{V}, \mathbf{R}\mathrm{V}^0 + \mathbf{t}, \mathcal{R}) \tag{26.8}$$

The distance measure is itself a minimization problem over n rotations, and is typically solved via an alternating block coordinate descent strategy. Again, we prefer to lift the inner minimization parameters into a block $\mathcal{R} = \{\mathbf{R}_1, \ldots, \mathbf{R}_n\}$, and solve for them simultaneously with all others in order to enjoy the superlinear convergence of the Gauss-Newton method. Our implementation parameterizes $\mathcal{R}$ by Euler angles, so the energy is more correctly written $\mathcal{E}_{\text{reg}}(\mathrm{V}, \mathcal{R}(\Phi))$ where Φ is a vector of $3n$ angle parameters. Notice that the global transformation parameters are not strictly necessary here, because $D(\mathrm{V}, \hat{\mathrm{V}}) = D(\mathrm{V}, \mathbf{R}\hat{\mathrm{V}} + \mathbf{t})$, but their inclusion will improve our initial estimates, and ensures the Euler angles are always parameterizing near-identity rotations, avoiding gimbal lock.

26.3.2 Energy Minimization: Gauss-Newton Core Solver

To summarize the above, we wish to minimize, at every timestep, the sum

$$\begin{aligned} \mathcal{E}(\mathrm{V}^t, \mathrm{U}, \mathrm{W}, \Phi, \mathbf{R}, \mathbf{t}) &= \mathcal{E}_{\text{point}}(\mathrm{V}^t, \mathrm{U}, \mathrm{W}) \\ &+ \mathcal{E}_{\text{plane}}(\mathrm{V}^t, \mathrm{U}, \mathrm{W}) + \mathcal{E}_{\text{color}}(\mathrm{V}^t) + \mathcal{E}_{\text{reg}}(\mathrm{V}^t, \mathcal{R}(\Phi), \mathbf{R}, \mathbf{t}). \end{aligned} \tag{26.9}$$

The main computational tool will be a Gauss-Newton solver. The primary requirement for such a solver is that the energy function be in the form of a sum of squared *residuals*, that is that if $\mathbf{x}$ is the vector of unknown parameters, we have

$$E(\mathbf{x}) = \sum_i f_i(\mathbf{x})^2 = \|\mathbf{f}(\mathbf{x})\|^2.$$

This form is ensured by the various lifting transformations described above, noting that terms of the form $\psi(\|\mathbf{e}\|)$, which expand to include $w^2\|\mathbf{e}\|^2 = w^2 e_1^2 + \ldots$, are stacked into $\mathbf{x}$ as $w\mathbf{e}$.

At solver iteration k, a Gauss-Newton iteration step updates a parameter vector $\mathbf{x}^k$ as

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \mathbf{h} \quad \text{with} \quad J^\top J\mathbf{h} = J^\top \mathbf{f} \tag{26.10}$$

where J is the Jacobian of $\mathbf{f}$ evaluated at $\mathbf{x}^k$, with $(i,j)^{\text{th}}$ entry $J_{ij} = \frac{\partial f_i}{\partial x_j}$.

To compute $\mathbf{h}$, we have to solve a linear system, which we do iteratively using a preconditioned conjugate gradient (PCG) solver, the essential computational unit of which is repeated multiplication of the Jacobian by a vector. The key to real-time performance is thus to implement routines for the computation of $J\mathbf{h}$ (and $J^\top\mathbf{h}$) as efficiently as possible. This is enabled by exploiting the particular sparsity structure of J, illustrated in Figure 26.4. To make this structure as sparse as possible, we implement a hybrid optimizer: the global parameters $\mathbf{R}$ and $\mathbf{t}$ are estimated in an initial ICP step, and W is updated in an outer loop. This means the core solver is optimizing the $8n$ variables $\mathbf{x} = (\mathrm{V}, \mathrm{U}, \Phi)$, giving the structure in Figure 26.4.

GPU Implementation of Jacobian Multiplication

We never compute the Jacobian explicitly, but compute its entries on-the-fly in the routines for $J\mathbf{h}$ and $J^\top\mathbf{h}$. This strategy, common in CPU-based optimizers (e.g., see [WY10], or the JacobMult parameter to the MATLAB function `lsqnonlin`), has less computational overhead [WMdS*05] on the GPU, but does not appear to be widely employed in the vision, graphics, and learning literature, where many implementations appear to explicitly store an (unstructured) Jacobian.

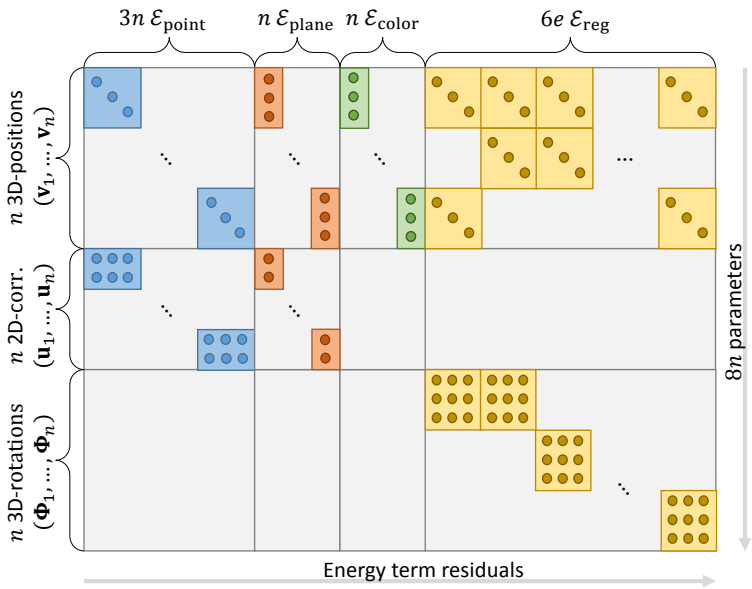


Figure 26.4: Block structure of the $(\mathrm{V}, \mathrm{U}, \Phi)$ Jacobian (transposed). The Jacobian is never stored explicitly; instead the products $J\mathbf{h}$ and $J^\top\mathbf{h}$ are computed on demand on the GPU.

To multiply J and $J^\top$ with a vector $\mathbf{h}$, we use two computational kernels. The first one multiplies a given row i of J with a given vector $\mathbf{h}$. According to the row i, the kernel determines which energy term is used, which columns are not equal to zero, computes the non-zero entries and immediately multiplies them with the appropriate entries of $\mathbf{h}$ and sums them up.

The second kernel does the same for $J^\top$. Yet, in this kernel each row corresponds to a parameter, so the kernel has to determine the non-zero entries in the i^{th} *column* of J and compute the scalar product. By this, we can well exploit the sparsity of J, and the GPU vector capabilities. Computing $J\mathbf{h}$ or $J^\top\mathbf{h}$ requires only one kernel call with $3n$ and $3n + 2m$ threads, respectively.

As a further optimization, both kernels interpret the $\mathbf{h}$ as a vector of 3D

floats. Furthermore, rows of J and $J^\top$ are merged to 3D floats. This allows us to map multiple operations to float3 vector arithmetic.

Preconditioned Conjugate Gradient on the GPU

As well as implementing Jacobian multiplication on the GPU, we also implemented the PCG solver itself. In its loop, we have to evaluate the matrix-vector product $J^\top(J\mathbf{h})$, which we can do very efficiently as shown above, but also a number of other terms. A naive implementation would require 12 kernel calls in the inner loop, where the kernel switches dominate computation. So instead we use a variant of the fast PCG-solver described by Weber et al. [WBS*13]. This solver reduces the number of kernel calls in the inner loop to three. However, since in our case the system matrix is J^TJ, we end up with four kernel calls. We use block diagonal preconditioning. The inverses of the diagonal blocks of $J^\top J$ are precomputed in the initialization stage of PCG, using a similar kernel to the Jacobian multipliers.

Energy Minimization: Coarse-to-fine Outer Loop

The previous sections describe the energy minimization for the model vertices at a single resolution. For both speed and accuracy, a multi-scale optimizer is used.

In an outer loop, the solver iterates over the mesh hierarchy from coarse to fine. In its inner loop, it optimizes for the optimal deformation on the current resolution as described in Section 26.3.2. At the end of the inner loop, a prolongation step transfers the solution on the current hierarchy level to the next finer one as described by Sumner et al. [SSP07], using the weights from Li et al. [LAGP09]. The weights are precomputed at the time of creation of the initial template model. We found that it was necessary to apply prolongation not just to the model vertices V but also to the rotation parameters Φ. The latter is performed by estimating rotations at the new scale using the closed-form Kabsch algorithm to solve (26.7) before starting Gauss-Newton. In conjunction with this hierarchical solving strat-

egy, a number of other continuation strategies are used to improve speed and/or convergence. Each run of the Gauss Newton solver is limited to $5-8$ iterations. On the finest hierarchy level, we apply an exponential average in order to reduce temporal flickering. Note that this only affects visualization, but not the optimization procedure.

The parameters which affect the location of energy minima are the energy weights $\lambda_{\text{point}}, \lambda_{\text{plane}}, \lambda_{\text{reg}}$ and the robust kernel width τ, and their setting is discussed below. Typically $\lambda_{\text{point}} = 0.2, \lambda_{\text{plane}} = 0.8$ can be kept fixed, and the value of λ_{reg} is chosen to coarsely reflect the amount of deformation in a given sequence. These settings are for the finest scale of the hierarchy. To improve convergence at the coarser scales, λ_{reg} is increased by a factor of 20, and τ by 10, so that only gross outliers are rejected. Note that these parameters affect only the *rate* of convergence, not the location of the energy minimum, which is affected by rather fewer parameters (see next paragraph). This can certainly mean that with different settings, the model may or may not converge completely in one frame if the object has undergone fast motion, but it will typically converge in a number of frames, particularly if the object slows down (see Figure 27.3)

26.3.3 Initialization: Correspondence Finding

As mentioned above, initialization of V^t simply takes the value from the previous frame, and initialization of Φ is to the parameters of the identity rotation. The parameters U represent, for each model vertex, the image location of the closest point to the vertex, and given that $\mathbf{d}(\mathbf{u})$ may be quite non-smooth, a more careful initialization is warranted. This is achieved by a simple local search in a window around the previous frame's estimate transformed by the global transformation $\mathbf{R}, \mathbf{t}$:

$$\mathbf{u}_i^t = \arg\min_{\mathbf{u} \in Q_i} \|\mathbf{R}(\mathbf{v}_i^{t-1}) + \mathbf{t} - \mathbf{d}^t(\mathbf{u})\|^2$$

where the window Q_i is $\pi^t(\mathbf{R}(\mathbf{v}_i^{t-1}) + \mathbf{t}) + [-24, 24] \times [-24, 24]$. For speed, this is computed in two stages: first checking only every third pixel in Q_i, then checking all pixels in a 5×5 window around the sub-sampled an-

swer. The GPU implementation uses an efficient parallel block reduction in shared memory.

At this stage, we can also update the visibility flags $\{\eta_i\}_{i=1}^n$ to discard correspondences to data points that lie close to the boundaries to the background. We also threshold the orientation difference between data point (provided by finite differencing) and model vertex normals, and impose a maximal distance threshold between associated point pairs.

26.3.4 Detail Integration

The result of energy minimization is a mesh at the second-finest resolution which matches the data, but does not feature fine-scale details, such as folds or wrinkles, that may be present in the current depth data. Because such transient surface details cannot be built into the initial template model, we prolong the fitted result to the finest hierarchy level and add the missing detail by computing in the least-square-sense optimal per-vertex scalar displacements d_i along the vertex normal.

Since the mesh is already very close to the measured data, the residual detail displacements can be assumed to be small. Therefore, the following algorithm can be used that fulfills our speed and plausibility requirements. We assume a thin shell deformation model whose minimal energy deformation state is found by minimizing the stretching and bending energies expressed as the differences in the first and second fundamental forms [BS08]. The thin shell deformation energy is simplified by replacing the change of first and second fundamental forms by changes of first and second order partial derivatives of the 3D displacement function $\mathbf{r}$ on the surface. This deformation energy is minimized by variational calculus, and linearization yields the following Euler Lagrange equations [BS08]:

$$\begin{aligned} -\lambda_s \Delta\mathbf{r} + \lambda_b \Delta^2 \mathbf{r} &= 0 \\ \text{with } \Delta\mathbf{r} &= \text{div}\nabla\mathbf{r} = \mathbf{r}_{uu} + \mathbf{r}_{vv} \\ \Delta^2\mathbf{r} &= \mathbf{r}_{uuuu} + 2\mathbf{r}_{uuvv} + \mathbf{r}_{vvvv} \end{aligned} \tag{26.11}$$

Here, $\mathbf{r}_u$, $\mathbf{r}_{uu}$, and $\mathbf{r}_{uuuu}$ are the first, second, and fourth partial derivatives of $\mathbf{r}$ w.r.t the surface parameterization of the template mesh; v-directions are defined analogously. λ_s and λ_b define stretching and bending resistance, respectively. To obtain the target displacements for each vertex, we find the closest intersection point in the input data by raymarching in normal direction. The resulting intersection point is further refined using a simple bisection approach. We incorporate the resulting target displacements as soft-constraints into the optimization problem. In our case, $\mathbf{r}$ is the residual displacement field on the mesh, and can be found by minimizing Equation 26.11 under the soft constraints using the fast GPU-based preconditioned conjugate gradient solver from Section 26.3.2. As initial guess for the iterative solve, we use the computed displacements for the previous frame to warm start the optimizer leading to fast convergence. Given the noise in the input data, we employ a temporal averaging scheme, similar to Li et al. [LAGP09], based on exponential weighting to compute the final displacements. This nicely removes noise, while still being responsive to changes in transient surface detail.

CHAPTER 27
Results

Now that we have described our system in detail, in this section we present a variety of results from live capture, ground truth experiments, and comparisons to existing work.

27.1 Live Non-rigid Capture

Our system is fully implemented on the GPU using CUDA. Results of live scene captures for our test scenes are shown in Figures 27.1 and 27.5 as well as in the supplementary material. It is important to stress that all these sequences were captured online and in real-time, including depth estimation

Figure 27.1: Our system enables the real-time capture of general shapes undergoing non-rigid deformations using a single depth camera. **Top left:** the object to be captured is scanned while undergoing rigid deformations, creating a base template. **Bottom left:** the object is manipulated and our method deforms the template to track the object. **Top and middle row:** we show our reconstruction for upper body, face, and hand sequences being captured in different poses as they are deformed. **Bottom row:** we show corresponding color and depth data for the reconstructed mesh in the middle row.

and non-rigid reconstruction. Further, these sequences are tracked over long time periods comprising several minutes.

We captured a variety of diverse non-rigidly moving and deforming objects. The table in Figure 27.5 shows the number of vertices in the different hierarchy levels, and indicates whether a tetrahedralized version of the mesh has been used to incorporate weak volume constraints. In the FACE sequence, we show how our system can generate compelling reconstructions of faces. Our results convey subtle expressions including detailed skin deformations and wrinkles. Our system also models large deformations captured during actions such as talking, frowning, and smiling, and demonstrates the benefits of modeling the fine facial deformations. This is in contrast to existing real-time methods based on parametric morphable models [LYYB13, WBLP11, WLGP09], which often fail to convey facial details.

However, our system is also able to reconstruct many other types of scenes beyond faces. In the HAND and UPPER BODY sequence, we show two challenging sequences which exhibit large amounts of occlusions when the user either places the hand in front of his/her body or significantly bends the fingers. Despite these occlusions our system is able to track non-rigid motions, although extreme poses and rapid motions can cause errors. In the TEDDY and BALL sequence, we finally show how our method can generalize to non-human tracking and reconstruction.

27.2 Performance

We measured performance of our entire non-rigid tracking pipeline including run-time overhead on an Intel Core i7 3.4GHz CPU, 16GB of RAM, and a single NVIDIA GeForce GTX780. The average timing (see Table 27.1 and 27.2) among all test scenes (see also Figure 27.5) is 33.1ms (i.e., 30.2fps) with 4.6ms for preprocessing (computing derivatives of normals, depth, and color data) (14% of the overall pipeline), 2.93ms (8.9%) for rigid ICP pose estimation (on average 4 iterations), 21.3ms (64%) for the non-rigid

	#vert. coarse	#vert. medium	#vert. fine	tetr. mesh	#frames
Face	1.2k	2.5k	20k	no	1490
Hand	0.6k	2.5k	20k	yes	587
Ball	1.2k	2.5k	40k	no	813
Teddy	1.0k	2.5k	40k	no	599
Body	1.1k	2.5k	20k	yes	1500

Table 27.1: Used settings for the different deformable objects.

	Prepro-cess	Rigid Fit	Non-Rig. Fit	Lin. Fit	Misc	Sum
Face	4.65	3.22	20,9	2.46	1.16	32.4
Hand	4.60	2.62	20,6	2.70	0.79	31.3
Ball	4.66	3.12	19,5	4.20	1.16	32.6
Teddy	4.58	2.84	19,0	4.69	0.80	31.9
Body	4.64	2.85	26.3	2.73	0.80	37.3
Avg.	4.62	2.93	21.3	3.36	0.94	33.1

Table 27.2: Timings for different deformable objects.

fitting on the coarse and medium mesh level (2×5 Gauss-Newton iterations, each with 10 PCG iterations), and 3.36ms (10%) for fine detail integration (10 iteration steps). On top of this, the timings of our stereo matcher are 17ms, or alternatively 26ms with variational refinement enabled. Note that in our current implementation we run the depth estimation on a separate second GPU, which allows for a complete runtime of 33ms (30fps) for our full pipeline by introducing a delay of one frame.

27.3 Applications

This type of non-rigid capture enables many compelling applications as shown in Figure 27.2. In the Re-target sequence we demonstrate a real-time, motion re-targeting scenario, where the user controls two avatars by transferring detailed non-rigid motions and expressions. Real-time avatar re-targeting can lead to new scenarios for gaming or video conferencing, where more detailed shape and motion can be reconstructed resulting in

Figure 27.2: Applications for live non-rigid capture. Left: detailed facial expressions are re-targeted. Right: spatio-temporal coherent re-texturing of template meshes for the same input sequence.

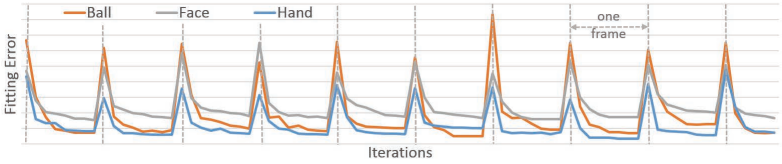


Figure 27.3: Convergence of the non-rigid deformation. The spikes correspond to new frames. Note convergence ``through'' the new-frame spike on the last frame of ``Face''.

more expressive user experiences. We use a simple re-targeting method by manually specifying a sparse set of per-vertex correspondences between our reconstructed template and the new target mesh. We use these correspondences to drive the animation using mesh skinning. Although this simple approach produces compelling results, more advanced re-targeting techniques such as [SP04] could easily be applied. Our live system can also be used for performance and motion capture in home and semi-professional movie and animation production. In the RE-TEXTURE sequence (Figure 27.2) we demonstrate how the estimation of detailed deformations enables convincing augmented reality applications such as pasting digital content onto non-rigidly deforming physical objects. Application scenarios include virtual clothing and makeup. Our deformation regularization prevents geometric drift which keeps texturing locally stable.

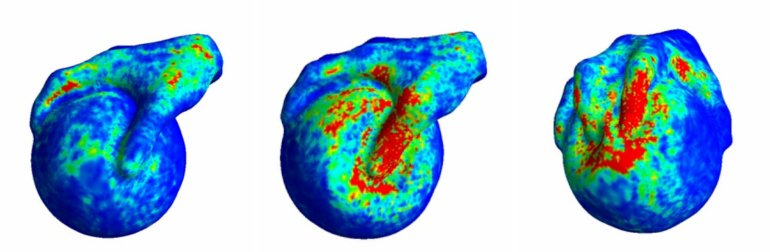


Figure 27.4: Energy of the ARAP regularizer at each vertex for the BALL example.

27.4 Evaluation

Fitting and Regularization Error Figure 27.3 shows the fitting error with respect to the iteration count over several frames of the examples FACE, HAND, and BALL. The spikes in the graph coincide with the arrival of new input frames. To examine convergence, we perform 8 Gauss-Newton iteration steps per frame on a single hierarchy level, with 20 PCG iterations in the inner loop. It can be seen that the registration converges quickly and in most of the cases, convergence is reached in less than 5 Gauss-Newton iterations.

The energy of the ARAP regularizer for example scenes is provided in Figure 27.4. This allows us to localize the regions undergoing locally non-rigid deformations in real-time. Interesting areas of future work include leveraging the ARAP residuals to either: 1) refine the template model in these regions (akin to the method of Li et al. [LAGP09]) in order to adapt the deformation model to the seen deformations, or 2) use this residual error to localize user interactions with objects. For example, in the ball sequence we clearly identify where the user is touching and pressing the ball. This leads to the possibility of making such physical objects interactive and enables new types of user experiences, e.g., for gaming.

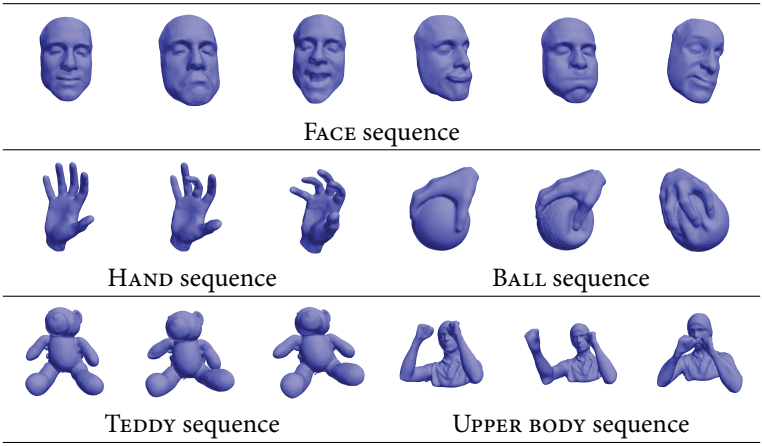


Figure 27.5: A number of different deformable objects during a live session. The corresponding timings and properties of the tracked template model are given in Table 27.1 and 27.2.

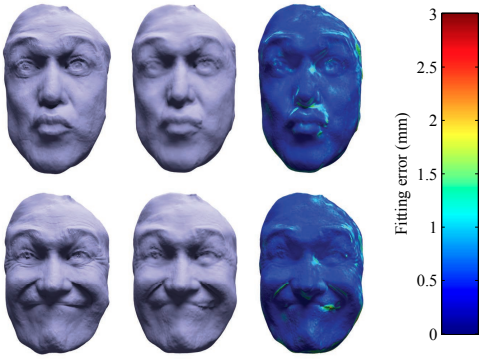


Figure 27.6: Ground truth comparison. Left: detailed input data. Middle: reconstruction from synthetic depth maps. Right: fitting error.

27.5 Comparisons

In Figure 27.6, we compare results obtained with our system with ground truth data. To this end, we used data from [VWB*12]. We generate synthetic depth maps by rendering the mesh from a single view. Our method is applied using eight Gauss-Newton iteration steps with 20 PCG iterations in the inner loop. The figure compares renderings of the original mesh and our reconstruction, as well as plots of the deviation for three frames of the animation, where red corresponds to a fitting error of 3mm. This shows that our algorithm can match the facial expression and fine scale details exhibited in this sequence. The method of Valgaerts et al. is an offline technique, with has a runtime of about 9 minutes per frame on the CPU. Our results show qualitatively similar results but with a system that is about 4 orders of magnitude faster, and with the ability to track a variety of general objects beyond faces.

Figure 27.7 shows a comparison with the results of Li et al. [LAGP09]. Both sequences were generated from the same input data. In both cases the reconstructed mesh has 70k vertices. Whereas Li's method requires more than one minute per frame on the CPU[1], our novel GPU pipeline runs at almost 30Hz and is thus more than three orders of magnitude faster.

27.6 Other Reconstruction Scenarios

Our technique can also be applied to multi-view setups. In Figure 27.8, we show reconstructions obtained with our method from the data sets Samba and Squat from [VBMP08]. The figure also shows a reconstruction of the ghost data set [VPB*09], which demonstrates our ability to deal with motions that cannot be parameterized by a skeleton. For multiple views, our reconstruction pipeline has to perform more work in the preprocessing stage of the pipeline, processing the data from each camera separately. We assign each vertex to the best suited camera, based on visibility and orienta-

[1]Timings by Li et al. [LAGP09]; expected to run faster on current CPUs.

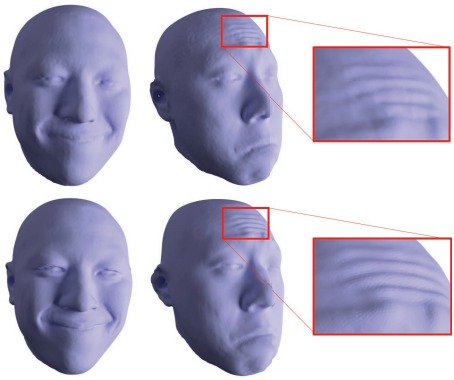


Figure 27.7: Comparison of our real-time reconstruction (top row) with offline reconstructions [LAGP09] while only using depth data (i.e., no color term).

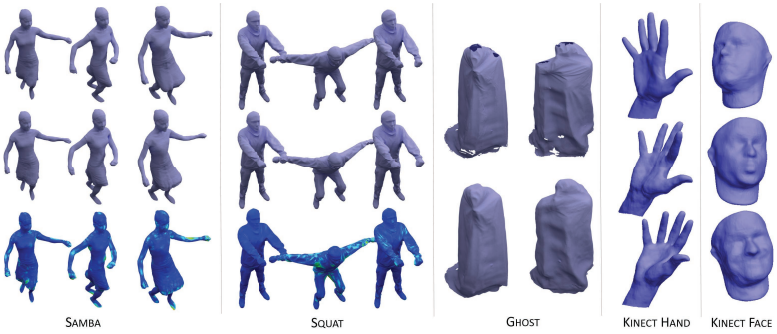


Figure 27.8: Samba, Squat: Reconstructions of synthetic multi-view input (8 depth cameras). Input (top row), our reconstruction (middle row), error (bottom row) with red=30mm. The right column of Squat shows the result at the end of the animation after four squats. Ghost: also reconstructed from synthetic multi-view input (8 cameras). Top row: input, bottom row: reconstruction. Kinect Hand & Face: Three frames of a sequences reconstruction using a single Kinect sensor.

tion, after which we can perform surface fitting within our presented fitting pipeline, which is independent of the number of cameras.

Finally, we also illustrate results of our method using a regular Kinect camera (see also Figure 27.8). Note that while our method produces far higher quality results with our stereo setup (for close ranges), the Kinect results could still be used in interactive applications, where quality is perhaps a secondary requirement, or where larger distance reconstruction is desired. This provides the exciting possibility of building both new multi-camera systems for performance capture using our method, as well as the possibility to use consumer depth cameras for certain interactive scenarios.

CHAPTER 28
Limitations

Even though we demonstrated one of the first methods for real-time non-rigid reconstruction from a single view, this problem is still ill-posed. In particular, the restriction to a single view leads to large occlusions resulting in missed correspondences. At each time step, typically less than half of the tracked object is visible [LAGP09], and the behavior of unobserved regions has to be inferred through regularization constraints. Offline methods tackle this problem with sophisticated correspondence finding mechanisms coupled with a slow relaxation of the model's rigidity during the optimization process in order to avoid local minima in the energy landscape. Given the tight real-time constraint (33ms/frame) of our approach, we rely on temporal coherence of the RGB-D input stream making the processing at 30Hz a necessity. If the frame rate is too low, or frame-to-frame motion is too large, our method might lose tracking. Similar problems may be caused by occluded regions, sparse/noisy input data, or a violation of the topological prior. Offline methods, e.g., [LSP08, LAGP09, BHB*11], fail in similar cases as ours; however, they are more stable due to a larger time budget that allows for more elaborate strategies, such as global optimization, remeshing of the deformation template, or anchor frames. In the following,

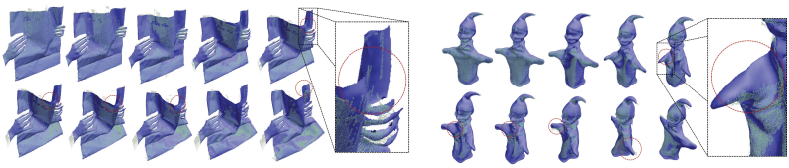


Figure 28.1: Limitations. Left: tracking instability due to sparse input leading to a slight misalignment of the paper. Right: tracking of the puppet's arm fails due to large and fast motion. However, our method recovers at the end of the sequence. Note, that our approach is less stable in these sequences, compared to the ones shown in Figure 27.5, since no color data is used.

we address specific failure cases in more detail and give ideas on how to improve in these situations.

Topological Changes Most template-based methods, no matter if they are online or offline, share our inability to robustly and efficiently handle topological changes. Only a few methods handle such situations [WAO*09], but at computation times that are far from real-time performance. A semantically incorrect prior counteracts the actual deformation (e.g., opening the mouth) which inevitably leads to surface sliding. While one could imagine the template to be modified at runtime, it would cause severe optimization instabilities and add significant computational complexity, which is (currently) infeasible in real-time. Similar problems occur if object parts not represented in the template are revealed during surface tracking (e.g., teeth). In scenarios where the topological assumption is satisfied (e.g., hand, boxer, teddy, ball), our method allows for robust tracking without surface sliding (see Figure 27.2).

Sparse Input and Occlusions Sparse input data and occlusions are inherent problems of a single-view camera setup. This causes missing correspondences, and thus increases the importance of regularization constraints. In these regions, there is no guarantee that deformations conform to the real-world, since we do not consider material or statistical shape priors. Methods focusing on a single domain, such as faces [WBLP11, LYYB13], are more robust towards occlusions since they have less degrees of freedom; however, they are less general and require a significant amount of training data. If our method misses large deformations due to occlusions, the temporal coherence assumption is violated once these regions become visible. This might lead to tracking instabilities or slow convergence. Given the tight real-time constraint, we can only afford searching correspondences on each level of the hierarchical solver independently. In theory, we would always like to consider alignment at the highest resolution, even when processing lower hierarchy levels; however, this comes at additional costs. Another problem of our vertex-to-input correspondence search is the possibility of undersampling the input depth data, which might lead to misalign-

ments; see Figure 28.1 (left). Ideally, one would prefer explaining all input data instead. Again, this is currently infeasible due to computational limitations.

Fast and Large Deformation A typical strategy to deal with fast and large deformations, is to incrementally relax the model rigidity in order to avoid local minima in the energy landscape. Therefore, offline approaches spend significant effort on slowly relaxing regularization constraints using many iterations. In our real-time scenario, we can only handle a limited amount of frame-to-frame deformation. In order to process reasonably fast motion, we enforce high temporal coherence leveraging our 30Hz input RGB-D stream. If the temporal coherence assumption is violated, tracking might fail; e.g., seeFigure 28.1 (right). However, note that our method can recover in most cases. In the future, we also expect RGB-D cameras to have higher frame rates, thus making faster motion possible.

CHAPTER 29
Conclusions

We have introduced what we believe to be the first `general purpose' non-rigid reconstruction system that provides *real-time* performance, several orders of magnitude faster than general methods, without using a specific `baked in' kinematic or shape model. Instead our system allows users to acquire a template online, and use it for *live* non-rigid reconstructions. Our system is simple to use and self-contained, with a single lightweight stereo camera setup, moving closer to commodity or consumer use. Additionally, in terms of reconstructed geometric detail, our method also narrows the gap between offline and online methods. Our system attempts to hit a `sweet spot' between methods that can reconstruct general scenes, and techniques that rely on a stronger shape prior (e.g., a blendshape face model or body or hand skeleton), which are beginning to demonstrate real-time performance. As shown in the results section, the simplicity of our method and its real-time performance does not significantly compromise the overall reconstruction quality. Our work brings us a step closer to high-quality real-time performance capture systems for consumer scenarios including gaming, home and semi-professional movie production, and human-computer interaction.

CHAPTER 30
Summary and Outlook

In this dissertation, we have presented methods and algorithms for the real-time acquisition, deformation and tracking of static and physically deforming scenes. We have shown that low-quality depth maps of commodity RGB-D sensors can be used to obtain high-quality super-resolution reconstructions of large-scale environments. For specific object classes, i.e. human heads, statistical information can be exploited to further increase the reconstruction quality by jointly optimizing for the geometry, surface albedo and illumination parameters. New variations of the three-dimensional reconstructions, which can consist of millions of polygons, can be interactively and intuitively created using the presented handle based deformation metaphor. This allows basically everybody to create detailed digital models that can be used as props in virtual reality applications, video games and movie productions. In addition, we have shown that the non-rigid motion of arbitrary physically deforming objects can be tracked at real-time rates using a general deformation framework. The obtained dense set of acquired temporal correspondences can be leveraged to analyze an object's motion and seamlessly re-target it to a variety of different objects. This will allow users to easily animate digital models and breath life into virtual characters using virtual puppetry. The captured reconstructions and animations can be shared digitally with friends or can be used in teleconferencing applications. Currently, our non-rigid tracking method is based on a single custom RGB-D camera that outperforms consumer grade devices in terms of accuracy.

In the future, we expect commodity RGB-D sensors to make a leap forward in accuracy and resolution of the captured color and depth streams. This will enable the broad public to leverage all of the presented techniques in their everyday lifes and will further increase the demand for fast algorithms and efficient techniques that can handle such a high amount of captured RGB-D data at real-time rates. We strongly believe, that the availabil-

ity of higher quality sensors on the mass market will not only revolutionize the field of 3D reconstruction, but will have a huge impact in general and shape many aspects of our everyday lifes. With virtual reality applications such as virtual mirrors and virtual try-on, i.e. trying on new cloth will be a completely new experience. Digital measurements of the customer's anatomy will enable the custom design and fabrication of perfectly fitting cloth. Teleconferencing and virtual puppetry will create completely new interaction and communication experiences between people. In addition, we will be able to track and analyze facial motions and control arbitrary devices through simple movements and gestures in real-time.

We are looking forward to a bright future, full of new opportunities that will redefine the way we think about digital content.

Bibliography

[ABBO84] Anderson C. H., Bergen J. R., Burt P. J., Ogden J. M.: *Pyramid Methods in Image Processing*.

[ASA*09] Alcantara D. A., Sharf A., Abbasinejad F., Sengupta S., Mitzenmacher M., Owens J. D., Amenta N.: *Real-time parallel hashing on the GPU*. ACM Transactions on Graphics (TOG) 28, 5 (2009), 154.

[aTH96] an Tang L., Huang T. S.: *Automatic construction of 3D human face models based on 2D images.* In ICIP (3) (1996), IEEE, pp. 467--470.

[AW87] Amanatides J., Woo A.: *A fast voxel traversal algorithm for ray tracing.* In Proc. Eurographics (1987), vol. 87, pp. 3--10.

[BC08] Bastos T., Celes W.: *GPU-accelerated adaptively sampled distance fields*. In Shape Modeling and Applications, 2008. SMI 2008. IEEE International Conference on (2008), IEEE, pp. 171--178.

[BHB*11] Beeler T., Hahn F., Bradley D., Bickel B., Beardsley P., Gotsman C., Sumner R. W., Gross M.: *High-quality passive facial performance capture using anchor frames.* ACM TOG (Proc. SIGGRAPH) 30, 4 (2011), 75:1--75:10.

[BHLW12] Bojsen-Hansen M., Li H., Wojtan C.: *Tracking surfaces with evolving topology.* ACM Trans. Graph. 31, 4 (2012), 53.

[BHZN10] Borosán P., Howard R., Zhang S., Nealen A.: *Hybrid Mesh Editing.* In EG Short Papers (2010), pp. 41--44.

[BKK*08] Breuer P., Kim K. I., Kienzle W., Blanz V., Schölkopf B.: *Automatic 3D Face Reconstruction from Single Images or Video*. In Proc. FG'08 (2008), pp. 1--8.

[BM92] Besl P. J., McKay N. D.: *A Method for Registration of 3-D Shapes*. IEEE Trans. Pattern Anal. Mach. Intell. 14, 2 (1992), 239--256.

[BPGK06] Botsch M., Pauly M., Gross M., Kobbelt L.: *PriMo: Coupled Prisms for Intuitive Surface Modeling*. In Proceedings of SGP'07 (2006), pp. 11--20.

[BPS*08] Bradley D., Popa T., Sheffer A., Heidrich W., Boubekeur T.: *Markerless Garment Capture*. ACM TOG (Proc. SIGGRAPH) 27, 3 (2008), 99.

[BPWG07] Botsch M., Pauly M., Wicke M., Gross M.: *Adaptive Space Deformations Based on Rigid Cells*. Computer Graphics Forum (Proc. EG '07) 26, 3 (2007), 339--345.

[BR07] Brown B. J., Rusinkiewicz S.: *Global non-rigid alignment of 3D scans*. ACM TOG 26, 3 (2007), 21--30.

[Bra00] Bradski G.: *OpenCV 2.4.5 (Open SOurce Computer Vision)*. *Dr. Dobb's Journal of Software Tools* (2000).

[BRR11] Bleyer M., Rhemann C., Rother C.: *PatchMatch Stereo: Stereo Matching with Slanted Support Windows*. In Proc. BMVC (2011), vol. 11, pp. 1--11.

[BS08] Botsch M., Sorkine O.: *On Linear Variational Surface Deformation Methods*. IEEE Trans. Vis. Comp. Graph 14, 1 (2008), 213--230.

[BSS07] Blanz V., Scherbaum K., Seidel H.-P.: *Fitting a Morphable Model to 3D Scans of Faces*. In Proc. ICCV'07 (2007), pp. 1--8.

[BV99] Blanz V., Vetter T.: *A Morphable Model for the Synthesis of 3D Faces*. In Proc. Siggraph'99 (1999), pp. 187--194.

[BV07] Basso C., Verri A.: *Fitting 3D Morphable Models using Implicit Representations*. JVRG 4, 718 (2007).

[CBI10] Cagniart C., Boyer E., Ilic S.: *Free-Form Mesh Tracking: a Patch-Based Approach*. In Proc. CVPR (2010).

[CBI13] Chen J., Bautembach D., Izadi S.: *Scalable real-time volumetric surface reconstruction*. ACM TOG 32, 4 (2013), 113.

[CCK94] Chang C., Chatterjee S., Kube P. R.: *A quantization error analysis for convergent stereo*. In Proc. ICIP 94 (1994), vol. 2, IEEE, pp. 735--739.

[CIF12] Chen J., Izadi S., Fitzgibbon A.: *KinÊtre: animating the world with the human body*. In Proc. UIST (2012), ACM, pp. 435--444.

[CL96] Curless B., Levoy M.: *A volumetric method for building complex models from range images*. In Proc. of the 23rd annual conference on Computer graphics and interactive techniques (1996), ACM, pp. 303--312.

[CM92] Chen Y., Medioni G.: *Object Modelling by Registration of Multiple Range Images*. Image Vision Comput. 10, 3 (Apr. 1992), 145--155.

[CNLE09] Crassin C., Neyret F., Lefebvre S., Eisemann E.: *Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering*. In Proc. Symp. Interactive 3D Graphics and Games (2009), ACM, pp. 15--22.

[CWLZ13] Cao C., Weng Y., Lin S., Zhou K.: *3D Shape Regression for Real-time Facial Animation*. ACM Trans. Graph. 32, 4 (July 2013), 41:1--41:10.

[CWZ*14] Cao C., Weng Y., Zhou S., Tong Y., Zhou K.: *FaceWarehouse: A 3D Facial Expression Database for Visual Computing*. IEEE Trans Vis Comput Graph 20, 3 (2014), 413--25.

[dAST*08] de Aguiar E., Stoll C., Theobalt C., Ahmed N., Seidel H.-P., Thrun S.: *Performance capture from sparse multi-view video*. *ACM TOG (Proc. SIGGRAPH) 27* (2008), 1--10.

[DFF13] Dou M., Fuchs H., Frahm J.-M.: *Scanning and tracking dynamic objects with commodity depth cameras*. In Proc. ISMAR (2013), IEEE, pp. 99--106.

[GKMT01] Goto T., Kshirsagar S., Magnenat-Thalmann N.: *Automatic Face Cloning and Animation*. IEEE Signal Processing Magazine 18, 3 (May 2001), 17--25.

[GLHL11] García I., Lefebvre S., Hornus S., Lasram A.: *Coherent parallel hashing*. ACM Transactions on Graphics (TOG) 30, 6 (2011), 161.

[Gow75] Gower J. C.: *Generalized Procrustes Analysis*. Psychometrika 40, 1 (1975), 31--51.

[GP07] Gross M., Pfister H.: *Point-based graphics*. Morgan Kaufmann, 2007.

[GPF10] Gallup D., Pollefeys M., Frahm J.-M.: *3D reconstruction using an n-layer heightmap*. In Pattern Recognition. Springer, 2010, pp. 1--10.

[GSDA*09] Gall J., Stoll C., De Aguiar E., Theobalt C., Rosenhahn B., Seidel H.-P.: *Motion capture using joint skeleton tracking and surface estimation*. In Proc. CVPR (2009), IEEE, pp. 1746--1753.

[GVWT13] Garrido P., Valgaert L., Wu C., Theobalt C.: *Reconstructing Detailed Dynamic Face Geometry from Monocular Video*. ACM TOG (Proc. SIGGRAPH Asia) 32, 6 (2013), 158:1--158:10.

[HBB*13] Helten T., Baak A., Bharaj G., Muller M., Seidel H.-P., Theobalt C.: *Personalization and Evaluation of a Real-Time Depth-Based Full Body Tracker*. In Proc. 3DV (2013), pp. 279--286.

[HBJP12] Hadwiger M., Beyer J., Jeong W.-K., Pfister H.: *Interactive Volume Exploration of Petascale Microscopy Data*

Streams Using a Visualization-Driven Virtual Memory Approach. Visualization and Computer Graphics, IEEE Transactions on 18, 12 (2012), 2285--2294.

[HDD*92] Hoppe H., DeRose T., Duchamp T., McDonald J., Stuetzle W.: *Surface reconstruction from unorganized points*. ACM SIGGRAPH Computer Graphics 26, 2 (1992), 71--78.

[HHI95] Higuchi K., Hebert M., Ikeuchi K.: *Building 3-D models from unregistered range images*. Graphical Models and Image Processing 57, 4 (1995), 315--333.

[HKH*12] Henry P., Krainin M., Herbst E., Ren X., Fox D.: *RGB-D Mapping: Using Kinect-Style Depth Cameras for Dense 3D Modeling of Indoor Environments*. *Int. J. Robotics Research 31* (Apr. 2012), 647--663.

[HSIW96] Hilton A., Stoddart A., Illingworth J., Windeatt T.: *Reliable surface reconstruction from multiple range images*. *J. Computer Vision (Proc. ECCV)* (1996), 117--126.

[HSO07] Harris M., Sengupta S., Owens J. D.: *Parallel prefix sum (scan) with CUDA*. GPU gems 3, 39 (2007), 851--876.

[HVB*07] Hernández C., Vogiatzis G., Brostow G. J., Stenger B., Cipolla R.: *Non-rigid photometric stereo with colored lights*. In Proc. ICCV (2007), IEEE, pp. 1--8.

[IKH*11] Izadi S., Kim D., Hilliges O., Molyneaux D., Newcombe R., Kohli P., Shotton J., Hodges S., Freeman D., Davison A., Fitzgibbon A.: *KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera*. In Proc. UIST (2011), pp. 559--568.

[IMH05] Igarashi T., Moscovich T., Hughes J. F.: *As-Rigid-As-Possible Shape Manipulation*. *ACM Trans. Graph. 24* (2005), 1134--1141.

[JBK99] Johnson A. E., Bing Kang S.: *Registration and integra-*

tion of textured 3D data. Image and vision computing 17, 2 (1999), 135--147.

[KBH06] Kazhdan M., Bolitho M., Hoppe H.: *Poisson surface reconstruction*. In Proc. EG Symp. Geometry Processing (2006).

[KBKL09] Kolb A., Barth E., Koch R., Larsen R.: *Time-of-Flight Sensors in Computer Graphics*. In Proc. Eurographics State-of-the-art Reports (2009), pp. 119--134.

[KE02] Kraus M., Ertl T.: *Adaptive texture maps*. In Proc. of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware (2002), Eurographics Association, pp. 7--15.

[KLK*10] Kim Y. S., Lim H., Kang B., Choi O., Lee K., Kim J. D. K., Kim C.-Y.: *Realistic 3D Face Modeling Using Feature-Preserving Surface Registration*. In Proc. ICIP'10 (2010), pp. 1821--1824.

[KLL*13] Keller M., Lefloch D., Lambers M., Izadi S., Weyrich T., Kolb A.: *Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion*. In Proc. of Joint 3DIM/3DPVT Conference (3DV) (2013), IEEE, pp. 1--8.

[KSA13] Kämpe V., Sintorn E., Assarsson U.: *High resolution sparse voxel DAGs*. ACM Transactions on Graphics (TOG) 32, 4 (2013), 101.

[LAGP09] Li H., Adams B., Guibas L. J., Pauly M.: *Robust single-view geometry and motion reconstruction*. ACM TOG 28, 5 (2009), 175.

[LC87] Lorensen W. E., Cline H. E.: *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*. Comput. Graph. 21, 4 (1987), 163--169.

[LH06] Lefebvre S., Hoppe H.: *Perfect spatial hashing*. ACM Transactions on Graphics (TOG) 25, 3 (2006), 579--588.

[LHN05] Lefebvre S., Hornus S., Neyret F.: *GPU Gems 2. chapter 37: Octree Textures on the GPU*, 2005.

[LK11] Laine S., Karras T.: *Efficient sparse voxel octrees*. Visualization and Computer Graphics, IEEE Transactions on 17, 8 (2011), 1048--1059.

[LMPM05] Lee J., Machiraju R., Pfister H., Moghaddam B.: *Estimation of 3D Faces and Illumination from Single Photographs Using A Bilinear Illumination Model*. In Proc. EGSR'05 (2005), pp. 73--82.

[LMT98] Lee W.-S., Magnenat-Thalmann N.: *Head Modeling from Pictures and Morphing in 3D with Image Metamorphosis Based on Triangulation*. In Proc. of the International Workshop on Modelling and Motion Capture Techniques for Virtual Environments (London, UK, UK, 1998), CAPTECH '98, Springer-Verlag, pp. 254--267.

[Low04] Low K.-L.: *Linear least-squares optimization for point-to-plane ICP surface registration*. Tech. rep., Chapel Hill, University of North Carolina, 2004.

[LPC*00] Levoy M., Pulli K., Curless B., Rusinkiewicz S., Koller D., Pereira L., Ginzton M., Anderson S., Davis J., Ginsberg J., et al.: *The digital Michelangelo project: 3D scanning of large statues*. In In Proc. Computer graphics and interactive techniques (2000), ACM Press/Addison-Wesley Publishing Co., pp. 131--144.

[LPMM05] Lee J., Pfister H., Moghaddam B., Machiraju R.: *Estimation of 3D Faces and Illumination from Single Photographs Using A Bilinear Illumination Model.* In Rendering Techniques (2005), Deussen O., Keller A., Bala K., Dutr P., Fellner D. W., Spencer S. N., (Eds.), Eurographics Association, pp. 73--82.

[LSP08] Li H., Sumner R. W., Pauly M.: *Global Correspondence Op-*

timization for Non-Rigid Registration of Depth Scans. Computer Graphics Forum (Proc. SGP'08) 27, 5 (2008), 1421--1430.

[LVG*13] Li H., Vouga E., Gudym A., Luo L., Barron J. T., Gusev G.: *3D Self-Portraits*. ACM TOG 32, 6 (2013), 187.

[LYYB13] Li H., Yu J., Ye Y., Bregler C.: *Realtime Facial Animation with On-the-fly Correctives*. ACM Transactions on Graphics 32, 4 (July 2013).

[LZW*09] Liao M., Zhang Q., Wang H., Yang R., Gong M.: *Modeling deformable objects from a single depth camera*. In Proc. ICCV (2009), IEEE, pp. 167--174.

[MFO*07] Mitra N. J., Flöry S., Ovsjanikov M., Gelfand N., Guibas L. J., Pottmann H.: *Dynamic geometry registration*. In Proc. SGP (2007), pp. 173--182.

[MS04] Moreno A. B., Sánchez A.: *GavabDB: A 3D Face Database*, March 2004.

[NIH*11] Newcombe R. A., Izadi S., Hilliges O., Molyneaux D., Kim D., Davison A. J., Kohli P., Shotton J., Hodges S., Fitzgibbon A.: *KinectFusion: Real-time dense surface mapping and tracking*. In Proc. IEEE Int. Symp. Mixed and Augmented Reality (2011), pp. 127--136.

[NIL12] Nguyen C., Izadi S., Lovell D.: *Modeling Kinect Sensor Noise for Improved 3D Reconstruction and Tracking*. In Proc. Int. Conf. 3D Imaging, Modeling, Processing, Visualization and Transmission (Oct. 2012), pp. 524--530.

[NVI10] NVIDIA Corporation: *NVIDIA CUDA C Programming Guide*, 2010. Version 3.2.

[NZIS13] Niessner M., Zollhöfer M., Izadi S., Stamminger M.: *Real-time 3D reconstruction at scale using voxel hashing*. ACM TOG 32, 6 (2013), 169.

[OKA11] Oikonomidis I., Kyriazis N., Argyros A. A.: *Efficient model-based 3D tracking of hand articulations using Kinect.* In Proc. BMVC (2011), pp. 1--11.

[PKA*09] Paysan P., Knothe R., Amberg B., Romdhani S., Vetter T.: *A 3D Face Model for Pose and Illumination Invariant Face Recognition*. IEEE.

[PM11] Pan J., Manocha D.: *Fast GPU-based locality sensitive hashing for k-nearest neighbor computation*. In Proc. of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (2011), ACM, pp. 211--220.

[PNF*08] Pollefeys M., Nistér D., Frahm J., Akbarzadeh A., Mordohai P., Clipp B., Engels C., Gallup D., Kim S., Merrell P., et al.: *Detailed real-time urban 3D reconstruction from video*. Int. J. Comp. Vision 78, 2 (2008), 143--167.

[PRI*13] Pradeep V., Rhemann C., Izadi S., Zach C., Bleyer M., Bathiche S.: *MonoFusion: Real-time 3D reconstruction of small scenes with a single web camera*. In Proc. ISMAR (2013), IEEE, pp. 83--88.

[RBV02] Romdhani S., Blanz V., Vetter T.: *Face identification by fitting a 3D morphable model using linear shape and texture error functions*. In in European Conference on Computer Vision (2002), pp. 3--19.

[RCBW12] Reichl F., Chajdas M. G., Bürger K., Westermann R.: *Hybrid Sample-based Surface Rendering*. In Vision, Modeling & Visualization (2012), The Eurographics Association, pp. 47--54.

[RHHL02] Rusinkiewicz S., Hall-Holt O., Levoy M.: *Real-time 3D model acquisition*. ACM Transactions on Graphics (TOG) 21, 3 (2002), 438--446.

[RL01] Rusinkiewicz S., Levoy M.: *Efficient Variants of the ICP*

Algorithm. In Third International Conference on 3D Digital Imaging and Modeling (3DIM) (June 2001).

[RV05] Romdhani S., Vetter T.: *Estimating 3d shape and texture using pixel intensity, edges, specular highlights, texture constraints and a prior*. In Edges, Specular Highlights, Texture Constraints and a Prior, Proc. of Computer Vision and Pattern Recognition (2005), pp. 986--993.

[RV12] Roth H., Vona M.: *Moving Volume KinectFusion*. In British Machine Vision Conf. (2012).

[SA07] Sorkine O., Alexa M.: *As-Rigid-As-Possible Surface Modeling*. In Proceedings of SGP'07 (2007), pp. 109--116.

[SB12] Stückler J., Behnke S.: *Integrating depth and color cues for dense multi-resolution scene mapping using RGB-D Cameras*. In Proc. of the IEEE Int. Conf. on Multisensor Fusion and Information Integration (MFI),(Hamburg, Germany) (2012).

[SE09] Schneider D. C., Eisert P.: *Fitting a Morphable Model to Pose and Shape of a Point Cloud*. In VMV (2009), Magnor M. A., Rosenhahn B., Theisel H., (Eds.), DNB, pp. 93--100.

[SH07] Starck J., Hilton A.: *Surface capture for performance-based animation*. Computer Graphics and Applications 27, 3 (2007), 21--31.

[SP86] Sederberg T. W., Parry S. R.: *Free-form deformation of solid geometric models*. Comput. Graph. 20, 4 (1986), 151--160.

[SP04] Sumner R. W., Popović J.: *Deformation Transfer for Triangle Meshes*. In ACM SIGGRAPH 2004 Papers (New York, NY, USA, 2004), SIGGRAPH '04, ACM, pp. 399--405.

[SRH*11] Scherbaum K., Ritschel T., Hullin M. B., Thormählen T., Blanz V., Seidel H.-P.: *Computer-Suggested Facial Makeup*. Comput. Graph. Forum 30, 2 (2011), 485--492.

[SSP07] Sumner R. W., Schmid J., Pauly M.: *Embedded deformation for shape manipulation.* ACM TOG 26, 3 (2007), 80.

[SSSB07] Scherbaum K., Sunkel M., Seidel H.-P., Blanz V.: *Prediction of Individual Non-Linear Aging Trajectories of Faces.* Comput. Graph. Forum 26, 3 (2007), 285--294.

[SZG10] Süssmuth J., Zollhöfer M., Greiner G.: *Animation Transplantation.* Computer Animation and Virtual Worlds 21, 3-4 (2010), 173--182.

[TBW*12] Tevs A., Berner A., Wand M., Ihrke I., Bokeloh M., Kerber J., Seidel H.-P.: *Animation cartography - intrinsic reconstruction of shape and motion.* ACM TOG 31, 2 (2012), 12.

[TdAS*10] Theobalt C., de Aguiar E., Stoll C., Seidel H.-P., Thrun S.: *Performance Capture from Multi-view Video.* In Image and Geometry Processing for 3D-Cinematography, Ronfard R., Taubin G., (Eds.). Springer, 2010, p. 127ff.

[TH96] Tang L.-A., Huang T. S.: *Automatic Construction of 3D Human Face Models Based on 2D Images.* In Proc. Image Processing'96 (1996), pp. 467--470.

[THM*03] Teschner M., Heidelberger B., Müller M., Pomeranets D., Gross M.: *Optimized spatial hashing for collision detection of deformable objects.* In Proc.of Vision, Modeling, Visualization VMV'03 (2003), pp. 47--54.

[TL94] Turk G., Levoy M.: *Zippered polygon meshes from range images.* In In Proc. Computer graphics and interactive techniques (1994), pp. 311--318.

[TM98] Tomasi C., Manduchi R.: *Bilateral Filtering for Gray and Color Images.* In Proc. of the Sixth International Conference on Computer Vision (Washington, DC, USA, 1998), ICCV '98, IEEE Computer Society, pp. 839--.

[TSSF12] Taylor J., Shotton J., Sharp T., Fitzgibbon A.: *The*

Vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation. In Proc. CVPR (2012), IEEE, pp. 103--110.

[TZL*12] Tong J., Zhou J., Liu L., Pan Z., Yan H.: *Scanning 3D full human bodies using Kinects*. TVCG 18, 4 (2012), 643--650.

[VBMP08] Vlasic D., Baran I., Matusik W., Popović J.: *Articulated mesh animation from multi-view silhouettes*. *ACM TOG (Proc. SIGGRAPH)* (2008).

[VPB*09] Vlasic D., Peers P., Baran I., Debevec P., Popovic J., Rusinkiewicz S., Matusik W.: *Dynamic shape capture using multi-view photometric stereo*. ACM TOG (Proc. SIGGRAPH Asia) 28, 5 (2009), 174.

[VWB*12] Valgaerts L., Wu C., Bruhn A., Seidel H.-P., Theobalt C.: *Lightweight Binocular Facial Performance Capture under Uncontrolled Lighting*. ACM TOG (Proc. SIGGRAPH Asia) 31, 6 (November 2012), 187:1--187:11.

[WAO*09] Wand M., Adams B., Ovsjanikov M., Berner A., Bokeloh M., Jenke P., Guibas L., Seidel H.-P., Schilling A.: *Efficient reconstruction of nonrigid shape and motion from real-time 3D scanner data*. *ACM TOG 28* (2009), 15:1--15:15.

[WBLP11] Weise T., Bouaziz S., Li H., Pauly M.: *Realtime Performance-based Facial Animation*. In ACM SIGGRAPH 2011 Papers (New York, NY, USA, 2011), SIGGRAPH '11, ACM, pp. 77:1--77:10.

[WBS*13] Weber D., Bender J., Schnoes M., Stork A., Fellner D.: *Efficient GPU Data Structures and Methods to Solve Sparse Linear Systems in Dynamics Applications*. Computer Graphics Forum 32, 1 (2013), 16--26.

[WHB11] Weiss A., Hirshberg D., Black M. J.: *Home 3D body scans from noisy image and range data*. In Proc. ICCV (2011),

IEEE, pp. 1951--1958.

[WJK*12] Whelan T., Johannsson H., Kaess M., Leonard J., McDonald J.: *Robust Tracking for Real-Time Dense RGB-D Mapping with Kintinuous*. Tech. rep., 2012. Query date: 2012-10-25.

[WLG08] Weise T., Leibe B., Gool L. J. V.: *Accurate and robust registration for in-hand modeling.* In CVPR (2008), IEEE Computer Society.

[WLGP09] Weise T., Li H., Gool L. V., Pauly M.: *Face/Off: Live Facial Puppetry*. In Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer animation (Proc. SCA'09) (ETH Zurich, August 2009), Eurographics Association.

[WLHM10] Waziri M., Leong W., Hassan M., Monsi M.: *A New Newton's Method with Diagonal Jacobian Approximation for Systems of Nonlinear Equations. Journal of Mathematics and Statistics 6* (2010), 246--252.

[WMdS*05] White B. S., McKee S. A., de Supinski B. R., Miller B., Quinlan D., Schulz M.: *Improving the Computational Intensity of Unstructured Mesh Applications*. In Proc. ACM Intl. Conf. on Supercomputing (2005), pp. 341--350.

[WSI98] Wheeler M., Sato Y., Ikeuchi K.: *Consensus surfaces for modeling 3D objects from multiple range images*. In Proc. IEEE Int. Conf. Computer Vision (1998), pp. 917--924.

[WSQO05] Weissenfeld A., Stefanoski N., Qiuqiong S., Ostermann J.: *Adaptation of a Generic Face Model to a 3D Scan*. In Proc. ICOB'05 (2005).

[WSVT13] Wu C., Stoll C., Valgaerts L., Theobalt C.: *On-set performance capture of multiple actors with a stereo camera*. ACM TOG 32, 6 (2013), 161.

[WWC*05] Waschbüsch M., Würmlin S., Cotting D., Sadlo F., Gross M.: *Scalable 3D Video of Dynamic Scenes*. In Proc. Pacific Graphics (2005), pp. 629--638.

[WWL*09] Weise T., Wismer T., Leibe B., , Gool L. V.: *In-hand Scanning with Online Loop Closure*. In IEEE International Workshop on 3-D Digital Imaging and Modeling (October 2009).

[WWLVG09] Weise T., Wismer T., Leibe B., Van Gool L.: *In-hand scanning with online loop closure*. In Proc. IEEE Int. Conf. Computer Vision Workshops (2009), pp. 1630--1637.

[WY10] Wilamowski B. M., Yu H.: *Improved computation for Levenberg-Marquardt training*. IEEE Trans. Neural Networks 21, 6 (2010), 930--937.

[WZC12] Wei X., Zhang P., Chai J.: *Accurate realtime full-body motion capture using a single depth camera*. ACM TOG 31, 6 (Nov. 2012), 188:1--188:12.

[YLH*12] Ye G., Liu Y., Hasler N., Ji X., Dai Q., Theobalt C.: *Performance capture of interacting characters with handheld kinects*. In Proc. ECCV. Springer, 2012, pp. 828--841.

[ZGHG11] Zhou K., Gong M., Huang X., Guo B.: *Data-Parallel Octrees for Surface Reconstruction*. IEEE Trans. Vis. and Comp. Graph. 17, 5 (2011), 669--681.

[ZHS*05] Zhou K., Huang J., Snyder J., Liu X., Bao H., Guo B., Shum H.-Y.: *Large mesh deformation using the volumetric graph Laplacian. ACM Trans. Graph. 24* (July 2005), 496--503.

[ZK13] Zhou Q.-Y., Koltun V.: *Dense Scene Reconstruction with Points of Interest*. ACM Transactions on Graphics (TOG) 32, 4 (2013), 112.

[ZMG*11] Zollhöfer M., Martinek M., Greiner G., Stamminger M., Süssmuth J.: *Automatic Reconstruction of Personalized*

Avatars from 3D Face Scans. Computer Animation and Virtual Worlds (Proceedings of CASA 2011) 22, 2-3 (2011), 195--202.

[ZNI*14] Zollhöfer M., Niessner M., Izadi S., Rhemann C., Zach C., Fisher M., Wu C., Fitzgibbon A., Loop C., Theobalt C., Stamminger M.: *Real-time Non-rigid Reconstruction Using an RGB-D Camera*. ACM Trans. Graph. 33, 4 (2014), 156:1--156:12.

[ZSGS12] Zollhöfer M., Sert E., Greiner G., Süssmuth J.: *GPU based ARAP Deformation using Volumetric Lattices.* In Eurographics (Short Papers) (2012), Andújar C., Puppo E., (Eds.), Eurographics Association, pp. 85--88.

[ZTC*14] Zollhöfer M., Thies J., Colaianni M., Stamminger M., Greiner G.: *Interactive model-based reconstruction of the human head using an RGB-D sensor*. Computer Animation and Virtual Worlds 25, 3-4 (2014), 213--222.

[ZZCL13] Zeng M., Zheng J., Cheng X., Liu X.: *Templateless Quasi-rigid Shape Modeling with Implicit Loop-Closure*. In Proc. CVPR (2013), IEEE, pp. 145--152.

[ZZZL12] Zeng M., Zhao F., Zheng J., Liu X.: *Octree-based Fusion for Realtime 3D Reconstruction*. *Graphical Models* (2012).

Echtzeit Rekonstruktion von statischen und dynamischen Szenen

Zusammenfassung

Seit der Veröffentlichung der Microsoft Xbox 360 Kinect ist ein echtzeitfähiger und erschwinglicher RGB-D Sensor auf dem Massenmarkt verfügbar. Dadurch sind viele der Verfahren, die vorher nur Technik-Verrückten und Forschern zur Verfügung standen, jetzt auch für eine breite Masse im Alltag direkt einsetzbar. Anwendungsgebiete für diese Technik erstrecken sich von der Erstellung detailgetreuer dreidimensionaler Modelle bis hin zur Verfolgung und Analyse von komplexen Bewegungsabläufen. Diese Verfahren stellen auch die Grundlage für Anwendungen der virtuellen Realität, wie zum Beispiel virtuellen Spiegeln, dar und sind sowohl die Basis für die Gestensteuerung als auch die Analyse von Bewegungsabläufen. Um die Bedienungsfreundlichkeit solcher Anwendungen zu erhöhen ist eine intuitive Steuerung und interaktive Darstellung der berechneten Ergebnisse zwingend erforderlich. In dieser Dissertation werden neue Techniken und Verfahren vorgestellt, die es erlauben dreidimensionale Beschreibungen von Objekten mittels einer handelsüblichen RGB-D Kamera zu erstellen, diese manuell nachzubearbeiten und die Bewegungen von realen Objekten in Echtzeit zu verfolgen. Das erste Verfahren verwendet statistisches Wissen um qualitativ hochwertige Rekonstruktionen des menschlichen Kopfes auf Basis von verrauschten Daten einer handelsüblichen RGB-D Kamera zu erstellen. Darauf aufbauend wird gezeigt, dass durch eine gekoppelte Optimierung von Form-, Farb- und Beleuchtungs-Parametern der Detailgrad der erstellten Rekonstruktionen weiter erhöht werden kann. Danach wird eine bewegte RGB-D Kamera verwendet um potentiell unbegrenzt große Areale in Echtzeit zu vermessen. Um die Skalierbarkeit des Verfahrens zu garantieren kommt eine neue dünnbesetzte Beschreibung der rekonstruierten Geometrie zum Einsatz. Zusätzlich stellen wir eine Technik vor, um die erstellten Rekonstruktionen, die aus Millionen von Polygonen bestehen können, in einem interaktiven Schritt nachzubearbeiten. Abschließend wird ein Verfahren beschrieben, welches die Bewegungen von realen Objekten mittels einer einzelnen RGB-D Kamera in Echtzeit verfolgen kann.

Curriculum Vitae

Michael Zollhöfer

PERSONAL INFORMATION

Person:

Name	Michael Zollhöfer
Gender	male
Nationality	German
Date of Birth	July 28, 1985
Place of Birth	Neustadt a. d. Aisch, Germany

Address:

Street	Siedlung 9
City	91074 Herzogenaurach
Country	Germany

Contact:

Phone	+49 9131 85-29929
Fax	+49 9131 85-29931
Email	michael.zollhoefer@cs.fau.de
Web	http://www9.informatik.uni-erlangen.de/people/card/michael/zollhoefer

SCHOOL AND ACADEMIC EDUCATION

1992/93 -- 1995/96	Elementary School (Herzogenaurach)
1996/97 -- 2004/05	German "Gymnasium" (Herzogenaurach)
Summer 2005	University-Entrance Qualification (German "Abitur")
Winter Term 2005/06 -- Summer Term 2010	Studies in Computer Science at the Friedrich Alexander University (FAU) Erlangen-Nuremberg
October 2007	Intermediate Diploma in Computer Science (German "Vordiplom")
August 2010	German "Diplom" with honors in Computer Science (Comp. to M.Sc., Final Grade: 1.0)
Since January 2011	PhD Student of Prof. Dr. Günther Greiner
August 2013 -- October 2013	Internship at Microsoft Research Cambridge