Shape Representations for Image-based Applications

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von Diplom-Informatiker

Alexander Hornung
aus Karlsruhe

Berichter: Prof. Dr. Leif Kobbelt
Prof. Dr. Luc Van Gool


Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.
Abstract

The mathematical representation of shape and appearance is a key issue in image-based applications. While the primary aim of 3D reconstruction is to reconstruct a geometrically accurate surface, real-time view synthesis requires efficient algorithms for computing plausible but not necessarily physically accurate images. These different objectives impose specific requirements with respect to the underlying shape representations. In this thesis three central problems from the spectrum of image-based techniques are investigated. We developed novel methods of representations and algorithms which on the one hand lead to substantial improvements of existing approaches and on the other hand offer a unified solution to problems that have previously been considered separately.

The first part of this thesis deals with creating animated character models from a set of input images. We propose a deformable, template-based shape representation which enables us to develop new solutions for problems such as camera estimation, shape deformation and tracking, and character reconstruction. We will present a variety of character animations created from single images to full body reconstructions and animations from video.

The second part of this thesis focuses on the difficulty of rendering novel views of general, static scenes instead of dynamic characters. Here, the key component is a generic, particle-based geometry representation which supports an accurate handling of object silhouettes and pixel-accurate rendering of arbitrary scenes. Every step of the process is completely implemented on the GPU in order to allow real-time, unconstrained user navigation through a photorealistic virtual reproduction of the original scene.

Finally, the third part concentrates on accurate 3D surface reconstruction. We will present a new volumetric solution to the problems of multi-view stereo and point cloud reconstruction which allows computing 3D models with a high accuracy as well as being robust to input degeneracies. Additionally, it is shown that the choice of input images is an important factor for optimizing the quality as well as the performance of image-based reconstruction techniques.
Acknowledgments

I would like to express my gratitude to everyone who contributed to this dissertation and supported me during the last few years.

First of all, I would like to sincerely thank my advisor Leif Kobbelt for his guidance, leadership and ongoing support; it has been a great privilege to work with him.

I am also very grateful to Luc Van Gool for being my co-examiner.

Having been part of Leif’s group has been an inspiring and pleasant experience, and I thank all current and former lab members for being great colleagues and friends, particularly Ellen Dekkers, Martin Habbecke, and Mario Botsch, who contributed to this work in various respects.

Most of all, I would like to thank my parents Inge and Joachim, my brother Felix, and Anja for their great support and patience.
## Contents

1 Introduction  
   1

2 Fundamental Concepts of Imaging and Shape  
   5
   2.1 Image Formation and Reconstruction  
      6
      2.1.1 Camera Model and Calibration  
      6
      2.1.2 Basic Reconstruction  
      9
   2.2 General Shape Representations  
      11
      2.2.1 Explicit Representations  
      11
      2.2.2 Volumetric Representations  
      13
      2.2.3 Relation to Reconstruction  
      15
   2.3 Image Synthesis  
      15

3 Character Reconstruction and Animation  
   19
   3.1 Discussion of 2D and 3D Approaches  
      19
   3.2 Conceptual Overview  
      23
      3.2.1 Generic Shape Template  
      24
      3.2.2 Reconstruction and Animation  
      25
   3.3 Shape Template Fitting  
      27
      3.3.1 Camera and Pose Estimation  
      28
      3.3.2 Template Projection and Fitting  
      32
   3.4 Single Input Views  
      34
      3.4.1 Texture Completion and Shape Initialization  
      34
      3.4.2 As-Similar-As-Possible Shape Deformation  
      35
      3.4.3 Animation and Rendering  
      38
   3.5 Multiple Input Views  
      40
      3.5.1 Shape Tracking  
      42
      3.5.2 Pose Synchronization  
      46
## Contents

3.5.3 Model Refinement ................................................. 49
3.5.4 Animation and Rendering ........................................ 50
3.6 Discussion .......................................................... 54

4 Interactive Free Viewpoint Rendering ................................. 57
  4.1 Conceptual Overview ................................................ 59
  4.1.1 Input View Proxies ............................................ 61
  4.1.2 Output View Synthesis ......................................... 62
  4.2 Particle Photo-Consistency ......................................... 63
  4.2.1 Volumetric Supersampling .................................... 65
  4.2.2 Silhouette Aware Sampling ................................. 67
  4.3 Proxy Generation ................................................... 68
  4.3.1 View-Space Parameterization .............................. 70
  4.3.2 Optimization .................................................... 72
  4.3.3 Regularization and Filtering ................................ 74
  4.4 View Synthesis ...................................................... 76
  4.4.1 View-Dependent Output Proxy ........................... 76
  4.4.2 Color Estimation ............................................. 78
  4.5 Efficient GPU-based Implementation ........................... 79
  4.6 Results and Discussion .......................................... 83

5 High Quality Model Reconstruction .................................. 87
  5.1 Discussion of Existing Techniques .............................. 87
  5.2 Conceptual Overview ............................................. 90
  5.2.1 Volumetric Confidence Map .................................. 92
  5.2.2 Energy Minimization based on Graph Cuts ................ 94
  5.2.3 Hierarchical Approach ....................................... 95
  5.2.4 Surface Mesh Extraction .................................... 95
  5.3 Surface Confidence Estimation ................................ 96
  5.3.1 Image-based Surface Confidence .......................... 96
  5.3.2 Confidence Diffusion for Point Clouds .................. 102
  5.4 Graph Construction and Surface Computation ............... 104
  5.4.1 Octahedral Graph Structure ................................ 105
  5.5 Hierarchical Crust Refinement ................................ 107
  5.5.1 Iterative Visibility Update ................................ 108
1 Introduction

Images represent the common fundamental basis of computer vision and graphics research with largely different but complementary research objectives in the two fields. In computer vision research, images are primarily considered as input data with the aim of transforming the contained information into new digital representations. Examples for that are problems such as object detection and recognition, motion tracking, and scene reconstruction. In contrast, the main focus in computer graphics is on processing existing digital data in order to synthesize new output images as, for example, research on efficient rendering and data visualization, geometric and physical modeling, and computer animation.

However, there occur problems at the borderline between vision and graphics research, which exhibit a strong coupling of the input and output data and hence can be solved more efficiently by combining methods from both fields. A typical example is the spectrum of image-based rendering and reconstruction techniques. In this field an essential requirement is the realistic preservation or reproduction of the input data. For instance, in most photo and video editing applications it is of importance to retain the original photorealism or visual appearance of the input pictures. The primary concern in these applications is the plausibility of the generated output images, whereas the accuracy, e.g., concerning physical properties of the depicted scene, is only of subordinate importance. In contrast, techniques for multi-view 3D model reconstruction are often focused on specific properties of the input data such as a precise reconstruction of an object’s surface geometry. Other attributes like the surface reflectance have to be disregarded or approximated due to the overall complexity of the problem. In these applications, the understanding of realism is related to actual physical correctness of certain properties rather than to a generally plausible reproduction. Thus, within the range of image-based techniques, the underlying concept of realism may vary considerably (see Figure 1.1).
1 Introduction

Figure 1.1: The spectrum of image-based techniques investigated in this thesis ranges from representations optimized for character reconstruction and animation over a more general technique for free viewpoint rendering of arbitrary scenes to a robust method for accurate surface reconstruction.

The key issue that has to be addressed in order to meet these different objectives is the underlying representation of shape: It controls the balance between plausibility and correctness since it defines which attributes of the input data are preserved. Furthermore, it is fundamental for the design of versatile and efficient algorithms. Hence, the development of appropriate data structures supporting the respective definition of realism for a given problem setting is crucial to any type of image-based application.

This dissertation thesis investigates three representative problems of the spectrum of image-based techniques: Character reconstruction and animation as an example for image and video editing, a technique for free viewpoint rendering, and a method for 3D surface reconstruction. Each of the corresponding three main chapters places particular emphasis on the discussion of the respective requirements and on the definition of appropriate shape representations. Based on these representations, optimization techniques are presented which allow substantial improvements of existing methods and even provide a number of solutions to previously unsolved problems.

Thesis Structure and Contributions

After an introduction of relevant general concepts in image-based rendering and reconstruction in Chapter 2, a new technique for creating animated characters from images is presented in Chapter 3. The challenge addressed in this chapter is the development
of a single representation that is able to cover a variety of character types and inputs. As
basis of our approach serves a generic character template based on which we propose
new techniques for the problems of camera estimation, shape deformation and tracking,
and character reconstruction. Our method supports the creation of plausible animations
from diverse input sources such as single images or uncalibrated video of moving subjects,
with only a minimal set of prior assumptions and constraints [HDK07, HDHK09].

Chapter 4 describes a more general representation as is required for the problem of free
viewpoint rendering from images. A method is introduced which achieves the necessary
flexibility and accuracy for rendering novel views of arbitrary scenes using a particle-
based geometry representation. Distinguishing features of this method are silhouette
aware particle shapes, a new photo-consistency optimization of particle proxies, and a
pixel-accurate blending field computation for the output view generation. Moreover, the
particle proxies support an efficient implementation using fully GPU-based optimization
schemes. This allows to generate high quality output images with a faithful reproduction
of object silhouettes at real-time frame rates [HK06b, HKss].

Finally, Chapter 5 concentrates on accurate 3D object reconstruction. A new solution
to the problems of multi-view stereo and point cloud reconstruction is presented, which
is able to compute 3D surface models at a high accuracy while at the same time being
robust to degeneracies and other inconsistencies in the input data. This is achieved by
a new volumetric approach for extracting the object surface from a surface confidence
map using globally optimal graph cuts. Additionally, the question of selecting an optimal
subset of images in the context of multi-view stereo reconstruction is addressed and it is
shown that the choice of input images is an important resource for optimizing the quality
as well as the performance of image-based reconstruction techniques. Special attention
is given to representations which support computationally efficient implementations in
order to achieve an actual benefit in practical systems [HK06a, HK06c, HZK08].
1 Introduction
2 Fundamental Concepts of Imaging and Shape

This chapter provides a brief introduction to the underlying concepts of a generic image-based processing pipeline, which is the basis for the techniques presented in the subsequent chapters of this thesis. An overview of the components of this pipeline is presented in Figure 2.1. The input consists of images of real world objects or scenes. On the basis of this input, our aim is to reconstruct a representation which allows the creation of a higher-dimensional scene model and the synthesis of new output images.

**Figure 2.1:** The major components of a generic image-based application.

Fundamental for understanding the relation between the input images and the scene is a model of the image formation process. A number of successful standard models exist in computer vision and graphics research which abstract and simplify the actual physical processes involved in taking digital photos from arbitrary real world scenes. The methods presented in this thesis are based on these commonly used standard models [FP03, HZ03, MSKS04].

The basic image formation model consists of two major components: A geometric description of the imaging process and a photometric model. The geometric component describes the mapping of a point in the three-dimensional world space onto the 2D plane of the image sensor of a camera. The photometric component explains the creation of the actual image color intensities based on models of the light sources, the geometry and reflectance properties of the surfaces within the scene, or the sensor’s response function.
2 Fundamental Concepts of Imaging and Shape

The combined model of the image formation process then allows to infer and reconstruct information about the 3D shape and appearance of real world objects from a set of input pictures (see Section 2.1).

The second major issue is the representation of the reconstructed information. As already mentioned in the introduction, applications for image-based rendering often aim for a visually plausible but not necessarily physically correct scene reproduction. This means that the underlying representation has to support the generation of new images which preserve the overall appearance of the input views. However, it is not required to enable accurate simulations or measurements of, e.g., the surface geometry or scene illumination. For instance, the walking motion of the character shown in Figure 1.1 (left) is based on a 2D deformation approach instead of a full 3D simulation of the character’s body. Another example is the free viewpoint rendering technique shown in Figure 1.1 (middle), which allows to render plausible views of a scene from new camera positions. However, it reproduces only those illumination effects (such as specular highlights) which are already contained in the input views.

In contrast, data structures for accurate 3D model reconstruction from images have to support the reconstruction of a numerically precise surface geometry as illustrated in Figure 1.1 (right). Here, the focus is on the exact digitization of a specific scene property under the influence of input degeneracies and artifacts such as noise, instead of an approximate representation of all appearance attributes. Another important aspect is that the representation supports the manipulation or modification of the reconstruction before further processing. For instance, an essential requirement in Chapter 3 on character reconstruction and animation is the deformation of the reconstructed character model using motion capture data prior to the synthesis of new output images. These requirements can only be met by using data structures, which address the specific properties of a particular application scenario (see Section 2.2).

Once the reconstruction process is completed the resulting representation has to be used for rendering novel images. This requires a model of the image formation process from the chosen representation back to realistic output images (see Section 2.3).
### 2.1 Image Formation and Reconstruction

A digital image generally consists of an array of width \( \times \) height pixels, in which the pixels correspond to discretely sampled locations on the image sensor of the camera. Each pixel stores a triplet of RGB color values representing the measured irradiance or intensity which is reflected from the scene through the optics of the camera onto the image sensor. Therefore, we can represent an image as a bivariate map \( I : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}_+^3 \), which assigns each pixel \((x, y) \in \Omega\) a vector \((r, g, b) \in \mathbb{R}_+^3\). Since the physical color measurements are only available at discrete pixel positions \( \in \mathbb{N}^2 \) we assume in the following that color values at positions \( \notin \mathbb{N}^2 \) are determined by interpolation.

One common approach for modelling the image formation process and, in particular, the geometry of the mapping from a position in the 3D space of the scene to a position on the image is the so called pinhole camera model.

#### 2.1.1 Camera Model and Calibration

The geometry of the pinhole camera can be modeled by central projection: The image \( p' \) of a 3D point \( p \) is created by intersecting a ray from the center of projection (the camera center) \( c \) to \( p \) with the image plane of image \( I \). Figure 2.2 (a) is a 2D illustration of this process. Using similar triangles, the projected \( y \) coordinate \( y' \) of \( p \) in image \( I \) is given by the ratio \( \frac{y'}{z'} = \frac{y}{z} \), where the focal length \( \delta \) corresponds to the distance of the image plane to the center of projection. This projection can be modeled conveniently by matrix multiplication using homogeneous coordinates: A point \((x, y, z)\) in Euclidean 3-space \( \mathbb{R}^3 \) is represented by a 4-tuple \((wx, wy, wz, w)\), \( w \neq 0 \) in projective 3-space \( \mathbb{P}^3 \).

The projection from \( \mathbb{P}^3 \) to \( \mathbb{P}^2 \) is performed via a \( 3 \times 4 \) projection matrix \( P \) with 11 degrees of freedom (due to scale):

\[
\begin{pmatrix}
  x' \\
  y' \\
  z'
\end{pmatrix}
= P
\begin{pmatrix}
  wx \\
  wy \\
  wz \\
  w
\end{pmatrix}
= Pp. \tag{2.1}
\]

The required perspective division to compute the actual projected 2D image coordinates is achieved by de-homogenization: \( p' \leftarrow (x'/z', y'/z', 1)^T \). In this Euclidean camera...
model, the general projection $P$ can be decomposed into physically meaningful intrinsic camera parameters and an exterior orientation and position:

$$P = KR \left[ I \mid -c \right]. \quad (2.2)$$

The $3 \times 3$ rotation matrix $R \in SO(3)$ and the position $c \in \mathbb{R}^3$ represent the extrinsic orientation and the world space position of the camera center, respectively. The matrix $K \in \mathbb{R}^{3 \times 3}$ corresponds to the intrinsic calibration of the camera and can be considered as a “viewport transform”, i.e., it determines the image coordinate frame. This transform has five parameters:

$$K = \begin{pmatrix}
\delta_x & s & x_p \\
0 & \delta_y & y_p \\
0 & 0 & 1
\end{pmatrix}. \quad (2.3)$$

Parameters $\delta_x$ and $\delta_y$ represent the focal length or scale with respect to the $x$ and $y$ axis of the image coordinate frame. The parameter $s$ accounts for skew. Finally, $x_p$ and $y_p$ define the principal point of the projection, i.e., the intersection point of the perpendicular on the image plane through the camera center $c$. These parameters fully describe the mapping of a 3D scene point onto the 2D image plane in the pinhole camera model.
A variety of different approaches for the estimation of these parameters from image measurements exist, e.g., based on images of a known calibration pattern [Tsa87, Zha00], image silhouettes [Her04], prior knowledge of the scene geometry [DTM96, OCDD01], or general structure-from-motion and bundle adjustment [TMHF00, HZ03, FP03, MSKS04]. However, the image generation in real cameras is generally also subject to lens distortions and other non-linear effects. These effects have to be considered as parts of the intrinsic calibration but are not addressed in the simplified central projection model. Some calibration techniques such as [Tsa87, Zha00] also estimate those parameters. In the following we assume that image distortions which are not handled by the pinhole camera model have been compensated for in the input images in a pre-process and that the camera calibration is provided by using one of the above mentioned standard techniques. An exception from this assumption is the method described for character reconstruction and animation in Chapter 3, for which we present a new algorithm for the estimation of camera parameters from a single image of an articulated character.

The photometric component of the image formation process describes the color intensity measurements of the image sensor. This model is considerably more involved than the above geometric model since, in its most general form, it requires knowledge about the scene geometry and surface reflectance properties, the distribution and intensity of light sources, or the spectral response function of the image sensor [GW02, FP03]. A common and well established approach in computer vision and graphics research in order to simplify the involved model complexity is to assume a simple Lambertian surface reflectance for all scene elements. In this model, all surface points have diffuse, i.e., isotropic reflectance properties. This means that the light energy or radiance arriving at a surface point \( \mathbf{p} \) from a particular direction is proportional to \( (\mathbf{n} \cdot \mathbf{l})/\|\mathbf{l}\| \) and therefore only dependent on the angle between the surface normal \( \mathbf{n} \) and the light ray \( \mathbf{l} \) (see Figure 2.2 (b)). Moreover, this irradiance is uniformly scattered over all directions of the hemisphere at \( \mathbf{p} \). Consequently, the perceived brightness or color of \( \mathbf{p} \) is constant and does not depend on the actual vantage point of the camera. This allows us to analyze scene surfaces which follow the Lambertian model by using techniques for image correspondence analysis (see Section 2.1.2). Although this simple model does not handle specular or anisotropic reflectance properties, it has proven to be a sufficient approximation in practice for many images of real world scenes, and it is currently the standard model in most techniques related to multi-view reconstruction [SCD06]. Effective ap-
2 Fundamental Concepts of Imaging and Shape

proaches to handle slight deviations from this model exist which will be discussed in more detail in Chapter 4.

2.1.2 Basic Reconstruction

Using the above model of the image formation process, we can formulate a basic reconstruction technique using correspondence analysis as illustrated in Figure 2.2 (b):  

Given at least two images \( I_0 \) and \( I_1 \) of a surface \( S \), the underlying idea is to identify corresponding points \( p'_0 \) and \( p'_1 \in \mathbb{R}^2 \) in the images, which result from the respective projections of a point \( p \in \mathbb{R}^3 \) lying on \( S \). The correspondence of \( p'_0 \) and \( p'_1 \) can be evaluated by comparing their appearance in the images, i.e., we expect

\[
t(I_0(p'_0)) \approx t(I_1(p'_1))
\]

(2.4)

where \( t \) is suitable transfer function depending on the particular illumination and surface model. For instance, assuming a constant brightness of a surface point in all images, i.e., the above mentioned Lambertian surface model and identical internal camera parameters like exposure times, \( t \) is simply the identity function \([\text{KS00b}]\). To address deviations from the constant brightness assumption the transfer function \( t \) could be a conversion to a different color space or the normalization of color values within small patches around the image positions \( p'_i \) \([\text{Her04}]\). More sophisticated functions \( t \) can be employed when more information about the photometric model of the scene is available (e.g., \([\text{YPW03, HS05}]\)).

There are a variety of standard techniques for detecting explicit image correspondences, ranging from single feature detection and tracking \([\text{BM04}]\) over dense optical flow \([\text{BBPW04}]\) to complex scale and illumination invariant feature descriptors \([\text{BTV06, Low04}]\). However, the following chapters will introduce a number of new, robust and illumination invariant transfer and comparison functions for correspondence evaluation to address the specific reconstruction problems discussed in this thesis. Since these correspondence measures evaluate the consistency of the appearance of a point \( p \) in different images, it will also be referred to as the photo-consistency or surface confidence \( \phi(p) \) in the following chapters.

After two corresponding points \( p'_0 \) and \( p'_1 \) are identified in the images, the position of the 3D surface point \( p \) can be reconstructed by computing the intersection of the two viewing rays \( v_i(\lambda_i) = c_i + \lambda_i Q^{-1}_i p'_i \) for \( i \in \{0, 1\} \), where \( Q_i \) is the left \( 3 \times 3 \) sub-matrix of \( P_i \). Since these rays do not intersect in general due to imperfect calibration and fea-
ture detection, the surface point \( p \) has to be computed by minimizing a cost functional based on, e.g., the reprojection error instead [HZ03]. Our technique for character reconstruction and animation described in Chapter 3 is based on this type of image-based correspondence estimation with subsequent reconstruction.

Instead of first finding corresponding points in different images and then computing a proper 3D representative, an alternative approach is to sample a particular position \( p \) in 3-space, and to evaluate the projections \( f_i(P, p) \) in the different images, again using Equation 2.4. As illustrated in Figure 2.2 (b), a sample \( p \) lying on the surface \( S \) has a consistent appearance according to the underlying correspondence estimation, while a sample position \( q \), which is not lying on the surface, is assumed to have an inconsistent appearance since the respective viewing rays intersect the surface at different locations. This approach does not directly yield an explicit 3D model, but rather computes for each sample point a confidence value (depending on the respective image comparison measure or transfer function in Equation 2.4) which represents the probability that a point is part of or close to a surface in the scene. This 3D sampling approach with subsequent projection and correspondence analysis will be discussed in detail in Chapters 4 and 5.

There are also a variety of different approaches to image-based reconstruction as, for example, techniques for photometric stereo [HVC08]. However, the techniques presented in the subsequent chapters are based on the ideas of explicit correspondence analysis introduced in this section.

### 2.2 General Shape Representations

Given the basic image formation model and reconstruction techniques described above one requires a suitable data structure for representing the reconstructed shape and appearance of a scene. There are two major classes of shape representations which are relevant to the techniques developed in this thesis: Explicit representations such as points or polygonal meshes, and volumetric representations based on a scalar distance function defined over 3-space. The following sections provide a brief definition of these representations and discuss the relation to the basic reconstruction techniques described in Section 2.1.2.
2 Fundamental Concepts of Imaging and Shape

2.2.1 Explicit Representations

The conceptually most simple form for explicitly representing a 3D surface $S$ is to create a set of sample points or vertices $V = \{v_0, \ldots, v_{n-1}\}$ (see Figure 2.3 (a)), where each $v_i \in \mathbb{R}^3$ is a point on the surface $S$ [KB04]. To simplify the notation in the following we will use $v_i$ to refer to the vertex with index $i$ as well as its actual location in 3-space. This kind of point-based representation is, e.g., a typical output of image-based 3D digitization techniques such as structure-from-motion [FP03], multi-view stereo [KS00b], or laser scans [Bla04]. However, a disadvantage of this discontinuous, piecewise constant representation is that it has to be processed further for computing a continuous approximation of $S$.

Point clouds can be extended to a piecewise linear representation of the surface by using circular or elliptical splats in object space (Figure 2.3 (b)), where each vertex $v_i$ is augmented with a surface normal direction $n_i$ and a splat radius $r_i$ (in the circular case). A splat-based representation preserves the flexibility of purely point-based representations since it does not store any explicit topological information [KB04]. This can be an advantage for representing and rendering complex or non-static objects, e.g., for object modeling [Pau03]. In Chapters 4 and 5 we will show how splat-based rendering techniques can be extended for hardware accelerated photo-consistency and surface quality evaluation.
2.2 General Shape Representations

Many applications, however, require a consistent, closed and 2-manifold surface with information about the geodesic neighborhood of a surface point, which is not possible with discontinuous splat-based representations. This property can be achieved by adding a topological structure to the point-based surface representation, i.e., by defining explicit neighborhoods between points on the surface. Hence, a so called polygonal mesh can be generated from a set of points \( V \) by adding a set of edges \( e_{ij} \in E \) which connect pairs of vertices \( (v_i, v_j) \) and mesh faces \( f_k \in F \) so that the resulting graph \( G := (V, E, F) \) is locally homeomorphic to a disc. The faces \( F \) then define a piecewise representation of the surface. In this thesis, the mesh representations are mainly based on piecewise linear triangle meshes (see Figure 2.3 (c)) where each face \( f_k \in F \) is constructed from three edges (or, equivalently, three vertices) and hence conforms to a linear surface approximation similar to splat-based surfaces. However, the edges and faces of \( G \) now define an explicit neighborhood relationship on the surface, which is required for applications such as surface smoothing, parameterization for texture mapping, or certain surface modeling and deformation techniques. Moreover, current graphics processors (GPUs) are mainly optimized for processing triangle meshes, which makes them an efficient representation for hardware accelerated rendering.

Mesh-based surface representations can also be considered as a special case of explicit parametric surfaces which are defined as the range of a bivariate function \( f : \Omega \rightarrow S \), where a 2D parameter domain \( \Omega \subset \mathbb{R}^2 \) is mapped to a surface \( S \subset \mathbb{R}^3 \). This parametric definition also includes higher order surfaces such as NURBS (see [KB03] for a more detailed characterization). In the context of this thesis, however, we concentrate on the above surface types and on the following volumetric surface definitions.

2.2.2 Volumetric Representations

Volumetric surface representations are generally based on a scalar distance function \( d : \mathbb{R}^3 \rightarrow \mathbb{R} \) defined over 3-space, e.g., using continuous basis functions or by sampling the values of \( d \) on a regular volumetric grid \( V \) embedded in \( \mathbb{R}^3 \). The most commonly used variants are the implicit surface representations, which store the signed distance to the object surface \( S \) of every grid point, i.e., grid points outside the object get a distance value \( d > 0 \), while grid points inside get a value \( d < 0 \). The set of surface points \( S \) can be characterized as the zero level-set or kernel of the function \( d \): \( S = \{ x \in \mathbb{R}^3 \mid d(x) = 0 \} \). Figure 2.4 (a) illustrates such a signed distance function \( d \). The zero-crossings of \( d \) which
represent the intersection of the surface $S$ with the underlying grid $V$ are visualized as small circles. Instead of the kernel of $d$, it is also possible to define a surface based on a different iso-value $c$ by requiring $d(x) = c$. An explicit representation of such an implicit surface can be reconstructed by using conversion methods such as Marching Cubes [LC87]. A detailed discussion of implicit surfaces and conversion techniques between explicit and implicit representations is provided in [KB03].

One of the main problems with signed distance functions is the fact that they require reliable information about the orientation of the surface. This is difficult or impossible to achieve for noisy input data containing a significant number of outliers such as the image correspondence measures in Section 2.1.2 or point clouds. The resulting spurious zero-crossings of $d$ (Figure 2.4 (a), lower left) often lead to topological artefacts during the conversion to an explicit representation.

We employ a different type of volumetric surface representation which is based on a surface confidence map $\phi : \mathbb{R}^3 \rightarrow \mathbb{R}_+$ with a positive range $\geq 0$. This function $\phi$ can be interpreted as an unsigned distance to the surface $S$ or, alternatively, as a pseudo-probability distribution of the location of $S$ in $\mathbb{R}^3$. An actual surface can then be characterized by the minimum of an energy functional such as the integral of distance values $\int_{S} \phi(x) \, dS$. In the discrete setting in Figure 2.4 (b) this corresponds to selecting the highlighted grid nodes as surface representatives.

---

**Figure 2.4:** Volumetric surface representations.
For the problem of 3D model reconstruction this representation has several advantages over the above mentioned implicit representations. Most importantly, an unsigned distance function does not require information about the orientation of the surface, which makes this representation much more robust with respect to noise and other artefacts in the input data. Since existing iso-surface reconstruction techniques are not able to reconstruct surfaces defined by unsigned distance values, we also present a new method for extracting an explicit surface which is the global optimum with respect to the confidence map $\phi$ and the used energy functional. These issues will be discussed in detail in Chapter 5.

### 2.2.3 Relation to Reconstruction

Given the requirements of a particular application, one has to choose a shape representation which supports the respective application goals. For the image-based techniques, this also implies the question which of the two basic approaches to 3D reconstruction introduced in Section 2.1.2 (i.e., triangulation of 2D point correspondences or 3D sampling with consistency estimation) is most suitable for a particular representation. When comparing the properties of these two fundamental reconstruction approaches with the two general types of shape representations, one finds a natural mapping between image correspondence analysis and explicit representations, and the 3D sampling approach and volumetric techniques.

The first reconstruction technique identifies corresponding points in two or more images and computes 3D surface points by triangulation of viewing rays through the 2D point correspondences. Since this technique generates *explicit* surface samples, it is directly related to explicit geometry representations such as point clouds or triangle meshes described in Section 2.2.1. In Chapter 3 we will use such a 2D image correspondence estimation in order to refine the 3D vertex positions of a triangle mesh.

The second reconstruction technique creates and evaluates 3D sample positions. This does not result in an explicit surface representation but rather provides a surface confidence estimate for each sampled position. These confidence samples can be represented by volumetric approaches based on an unsigned confidence map (see Section 2.2.2) and are the basis for the techniques presented in Chapters 4 and 5. However, both chapters also address the conversion of the resulting confidence maps into explicit representations for rendering the reconstructed scenes.
2.3 Image Synthesis

The last step of the basic image-processing pipeline is the generation of new output images from the reconstructed scene representations. A number of standard rendering techniques exist in computer graphics for the basic image formation model and the representations introduced in Sections 2.1 and 2.2.

Among the explicit representations, triangle meshes are most widely supported by current GPUs and commonly used graphics libraries such as OpenGL [SA06]. They can be considered as the current de-facto standard for high quality and high performance rendering. With the programmability of today’s GPUs, however, comparably high rendering performance can be achieved for point sets [PZvBG00, ABCO’01] and the piecewise linear surface splats [ZPvBG01, BHZK05]. For volumetric data without an explicit surface geometry there exist established rendering approaches as well [EHK’04]. But since real world input scenes and objects generally consist of surfaces rather than volumetric data, methods for rendering explicit representations are more suitable in the context of this thesis. Therefore, the algorithms in the subsequent chapters used for generating new output views are based on the above mentioned standard techniques for triangle- and splat-based rendering.

A central aspect of image-based applications is the preservation of the style or photorealism of the original input images (see, e.g., the free viewpoint rendering shown in Figure 1.1). Natural scenes generally exhibit view dependent appearance changes due to non-isotropic reflectance properties. These types of effects, such as specular highlights, subtle reflections, or dynamic shadows on non-rigid surfaces, can be handled to a certain extend by reconstruction techniques based on an extended Lambertian model (see Section 2.1.2), but are nevertheless of considerable visual importance to a human observer for increasing the realism of a scene.

The above standard approaches for rendering digital 3D models are based on simulated illumination and surface reflectance [FvDFH96]. By using sophisticated models it is generally possible to render images with a high degree of realism. However, the estimation of the required model parameters from a small set of input images is a highly ill-posed problem. This makes it difficult to reproduce the appearance of the various different surface types present in most real world scenes. Hence, techniques for new view synthesis from images try to render novel output views by using the original content of the source images instead of using synthetic illumination and surface models.
Thus, for each pixel $p^*$ of an output view $I^*$, one has to determine the color contribution from all relevant input images in which the corresponding 3D point $p$ is visible. For instance, given the two input views $I_0$ and $I_1$ in Figure 2.5, the color of pixel $p_0^*$ should be weighted stronger than the color of $p_1^*$ for producing the output color of $p^*$, since $\alpha_1 < \alpha_0$, i.e., the viewing ray direction for $p_1^*$ is more similar to $p^*$. In general, these color contributions can be computed based on blending weights $\omega_i$, with

$$I^*(p^*) = \frac{1}{\sum_{i} \omega_i} \sum_{i} \omega_i I_i(p_i^*).$$

Several requirements for computing reasonable weights $\omega_i$ and for avoiding visual artefacts in the output views have been identified in [BBM'01], such as the consideration of angular deviation between viewing rays, or the necessity for temporal and spatial continuity of the reconstructed views. These weighting schemes are also the basis for the image-based rendering approaches presented in this thesis. Our emphasis in Chapters 3 and 4 is on the efficient computation of such blending fields, with a high accuracy up to pixel-level and with special consideration of discontinuities at object silhouettes.
2 Fundamental Concepts of Imaging and Shape
3 Character Reconstruction and Animation

Generating and rendering animated characters is a classical task in many computer graphics related areas with a variety of applications, ranging from 2D image and video editing tools to 3D character models for movies or virtual environments. Nevertheless, the creation of animated characters is still a complex procedure. Hence, an important problem considered in research on character creation and animation is the generation of character models directly from images or video.

On a conceptual level most of the existing image-based approaches can be classified into two major types according to the underlying representation: 2D techniques, and approaches based on a 3D character representation. Although both representations have particular strengths, they are subject to considerable restrictions and difficulties in practice.

A new approach to image-based character reconstruction and animation will be presented after the following overview and discussion of existing techniques. Our method distinguishes itself by a more general and flexible representation which allows a unified handling of the full spectrum from 2D animations over partial character reconstructions to fully animated 3D models, with input ranging from single images to uncalibrated video of moving persons.

3.1 Discussion of 2D and 3D Approaches

2D Animation Traditional 2D animation originates in the CEL animation technique [Hur15] which is based on the idea of creating continuous motion of articulated characters by drawing a character in different poses on separate transparent layers. These layers are then successively overlaid on a background image and photographed in order to
Character Reconstruction and Animation

Figure 3.1: Examples of 2D and 3D character animation. Image (a) shows a single input image of a complex character model. The technique described in this chapter allows to generate animations (b) which preserve the photorealism of the original input. A full 3D character model can be extracted from uncalibrated video showing a person from several viewing angles (c). This 3D model can be animated and embedded into a new video (d).

create an animated movie. But despite the availability of computer aided animation systems [Dis91, Too08] today, the creation of convincing high quality animations is still a tedious process.

Recent improvements in 2D image processing such as segmentation [RKB04], boundary matting [SJTS04], texture synthesis [EL99] and image completion [DCOY03] have enabled the creation of sophisticated and dynamic image animations even from a single input image. For instance, in [CGZ∗05] the authors present techniques to animate passive elements such as water and trees that are subject to natural forces like wind by segmenting the image into separate animation layers and modeling the motion of each layer with stochastic processes.

In combination with the above techniques, methods for 2D shape deformation allow the manipulation of the pose of a character in an input image. For example, in [BC02a] a character’s pose can be changed based on a set of space warping functions and user defined deformation constraints. However, for plausible articulated deformations a certain rigidity of the character’s limbs has to be preserved. One class of techniques for this as-rigid-as-possible shape manipulation triangulates the interior of the character’s silhouette and mimics rigidity by computing the deformation with respect to shape preserving
energy functionals defined on the triangles [ACOL00, IMH05]. Instead of representing
the 2D shape by a triangle mesh, these shape preserving energies can be formulated
directly on the underlying regular grid in image space [SMW06] or by using general
space deformation techniques [BPWG07]. A more three-dimensional appearance of the
2D shapes can be created, e.g., by augmenting CEL animations with texture [CJTF98],
using a simulation of 3D shadows [PFWF00], or by adding illumination and perspective
texturing effects [OCN04].

Given such a deformable shape representation, animations can be generated either
by manually provided deformation constraints [IMH05] or by using curve-based systems
[TBvdP04] which allow the user to animate characters by drawing animation paths.
Nowadays, similar techniques are becoming available in commercial solutions [Pup08].
However, these tools still require considerable manual effort and the automatic creation
of full character animations is currently not possible. One exception is [Cra08] which
allows to generate face animations with simulated depth from single images and a pre-
defined database of facial expressions.

Animation techniques for single images with essentially two-dimensional shape rep-
resentations have one main advantage over more complex representations such as 3D
models. Since 2D approaches generally work directly on the input data without inter-
mediate conversions into different representations, they easily preserve the appearance
and style of the original image content (see Figure 3.1 (a) and (b)). Moreover, due
to simple generic models and a small set of prior assumptions about the input, they
are very flexible with respect to different types of images such as photos or paintings.
But this flexibility generally also implies a number of restrictions concerning the out-
dput diversity and overall accuracy. For instance, a difficult problem is the synthesis of
(dis-)occluded textures for unconstrained animation from just a single image. Although
approaches exist for the completion of missing image content by merging several inputs
[PSK06, HE07], animation techniques processing single images based on 2D animation
are generally restricted to a single viewpoint. Hence, the creation of a plausible anima-
tion showing a character turning around is difficult to realize.

3D Reconstruction and Animation The standard approach for generating animated
3D characters (see Figure 3.1 (d)) is to create and texture a character model manually in
a 3D modeling tool [Bie08], e.g., according to photos or artwork. Using skeleton-based
deformation and animation techniques [LCF00, LCR*02, WSLG07] these models can
then be animated with captured motion of a human actor [CMU08]. Similar to the 2D case this type of model generation and animation is a complex process which generally requires a considerable amount of time and expert knowledge. Most research on image-based techniques to automate the model generation process focuses on reconstructing the pose and full shape of human characters from multiple synchronized image streams using three-dimensional volumetric or mesh-based representations (see Chapter 2).

One class of techniques reconstructs an approximate character model by computing the visual hull [Lau94] from several image silhouettes [MBR00] which can then be refined for a more faithful reconstruction [SMP03, MTHC03, SH07] or used, e.g., for multi-view body tracking and pose estimation [KBV05]. Alternatively, a pre-defined human body model can be fitted to the silhouette of a human character [HBG00, LGMT00, CTMS03, TAL07]. A similar approach is described in [BSB07] based on the SCAPE model [ASK05], which has the advantage of modeling shape changes due to, e.g., muscle contraction. Another possibility is the use of active reconstruction techniques to acquire 3D models which can then be deformed or animated using multiple input video streams [dATSS07, dAST08, PG08, VBMP08]. There are also a few techniques which process uncalibrated input images of non-rigid surfaces for reconstructing partial 3D models [TH04, HVB07]. Due to specific constraints and a focus on reconstruction, however, the application of such methods to character animation does not appear feasible at the moment.

In summary, the above methods are able to deliver high quality reconstructions, and the resulting 3D models are generally neither restricted to certain types of animations nor viewpoints like the previously discussed 2D animation techniques for single images. However, these advantages also result in a reduced flexibility with respect to the input data since most of these approaches have strict requirements such as multiple synchronized and calibrated video streams which show the complete character, precise silhouette determination, depth information, or predefined parameterized shape models. These techniques are generally well suited for the reconstruction of humans but not capable of creating animations for more general character shapes. Hence, in contrast to techniques for 2D animation which are becoming more and more available in standard image processing tools even for the non-professional user, image-based 3D character reconstruction and animation techniques are still too complex due to their immense requirements and the involved complexity. In [HDK07] and [HDHK09] we proposed corresponding solutions which alleviate many of the above mentioned restrictions.
3.2 Conceptual Overview

Most approaches discussed in the previous section are optimized for a specific target application such as 2D image deformation or 3D model reconstruction. The supported input requirements and output properties result from the underlying and often quite different shape representations. This impedes a consolidation of the different approaches in order to build a single coherent processing pipeline for creating character animations.

Instead of enforcing a strict conceptual distinction between 2D and 3D image-based animation techniques by using specific character representations, we propose an approach based on a single 3D template model called generic shape template. The main idea is to compute all required image space representations consistently from this template model and 3D motion capture data. Shape analysis and modifications performed in image space can then be transferred back to the template in order to create the actual animation model. Based on this single generic shape template, our approach has several advantages:

**Scalability** The generic shape template allows for adaptive algorithms for reconstruction and animation which scale with the number and quality of available input images. Consequently, the technique supports the creation of animations from single images over partial character reconstructions to complete animated 3D models generated from multiple input views.

**Flexibility** The proposed method alleviates many constraints of current approaches to image-based reconstruction and animation by supporting more flexible algorithms. For example, based on the shape template, we present a new solution for computing camera parameters and the character pose, an algorithm to shape tracking with proper handling of occlusions, and a technique for character pose synchronization which removes the necessity for multiple synchronized camera streams during reconstruction. This allows us to generate 3D models from a single uncalibrated video of moving persons and of a variety of other character types.

**Plausibility** The integration of original image content and captured 3D motion data with the proposed shape template enables a faithful reproduction of the visual appearance of the input data and supports realistically moving characters.

In the following we introduce the basic concepts of our approach and provide a high level overview of the involved processing steps.
3 Character Reconstruction and Animation

Figure 3.2: (a) The 3D shape template model $M$ in its generic base pose $X_b$. (b) Embedded skeleton structure with bones $i$, joint positions $x_i$, and color coded bone transformation weights for each triangle. (c) The template mesh $M$ deformed according to a new pose $X_{\tau}$ from a walking motion data set.

Input to the method are one or more images $I_t$ of a character, e.g., a photograph, paintings, or an input video showing the character from different viewing angles. Prior knowledge about the depicted character or the camera such as the intrinsic or extrinsic calibration is not required. The index $t$ indicates that the input images $I_t$ do not have to be synchronized but may be taken at different times $t$. This implies that, in the case of multiple input images, the character’s pose does not have to be identical as it is required in standard multi-view reconstruction but that the character may move moderately. Exemplary input images are shown in Figure 3.1 and in the results in Sections 3.4.3 and 3.5.4.

3.2.1 Generic Shape Template

The main idea of our approach is to derive the required image space shape representations from a single generic shape template. For creating an animated character model we need a surface representation which supports an efficient reconstruction and deformation as well as mapping view-dependent image content during the rendering of output...
3.2 Conceptual Overview

animations. However, although the shape of the template has to be deformable, the surface topology may remain static. Hence, according to the discussion of explicit and volumetric representations in Section 2.2, the most suitable data structure for the template model is an explicit triangle mesh $M$. The template which was used for the results presented in this chapter is shown in Figure 3.2 (a).

This template model is augmented with an embedded skeleton (see, e.g., [HSDK05]) and weighted vertex-to-bone associations (Figure 3.2 (b)). As discussed in Section 3.1 such a model can be deformed into different poses using skeleton-driven animation techniques [LCF00]. We use a database of captured human motion data [CMU08] in order to deform and animate $M$. This data consists of a skeleton structure that is compatible to the embedded skeleton of $M$ and a sequence of motion frames in the form of skeleton poses $X_\tau$, indexed by parameter $\tau$. Each pose $X_\tau$ consists of a set of transformations $T_i$ and joint positions $x_i$ for every bone $i$ of the skeleton. Figure 3.2 (c) shows the template mesh $M$ deformed into $M_\tau$ based on a frame $\tau$ of forward walking motion.

By using this template model a view-dependent shape representation $S_t$ of the character in a reference image $I_t$ can be computed by projecting and fitting the 3D template $M$ to image $I_t$ (see Figure 3.3). This step requires information about the character's pose in $I_t$ and a corresponding camera projection matrix $P_t$. Section 3.3.1 describes a method for estimating $P_t$ and an approximate character pose $X_{t0}$ for image $I_t$ from user selected skeleton joint positions (see Figure 3.3 (a)). The deformed and projected model $M_{t0}$ shown in Figure 3.3 (b) then provides an estimate of the visible surface parts and the pose of the character in $I_t$ but generally does not match the overall character shape exactly. Hence, in a second step the projection is aligned to the character’s shape in the image, resulting in the actual 2D shape representation $S_t$ of the character (see Figure 3.3 (c)). This process is described in detail in Section 3.3.2. These projected and aligned 2D shapes $S_t$ are the basis for the subsequent reconstruction and animation of an output character model.

3.2.2 Reconstruction and Animation

As discussed in Section 3.1, in the case of a single input image $I_0$ it is generally not possible to infer a three-dimensional model from the character’s 2D shape $S_0$. Nevertheless, we will show in Section 3.4 that plausible animations can be generated by deforming $S_0$ with motion capture data directly in image space. We will introduce a perspectively
correct technique for as-rigid-as-possible shape manipulation which allows the animation of the 2D character shape with simulated 3D effects (see Figure 3.5).

With multiple input images, e.g., in the form of a video, the projection and fitting procedure could be executed for several reference views \(I_i\) and corresponding 2D shapes \(S_i\) in order to reconstruct the 3D shape of the depicted character, similar to standard multi-view reconstruction (see Section 3.1). However, the challenge is that one of the central goals of our proposed technique is the possibility to generate an animated model from a single video showing a moving character. In Section 3.5 we propose a method for \textit{pose synchronization} which is based on a deformation of the character’s 2D shapes \(S_i\) and the template \(M\) into an
3.3 Shape Template Fitting

For a given reference image $I_t$, the first necessary processing step in order to extract the corresponding 2D character shape $S_t$ is the estimation of a camera projection $P_t$ and a matching pose $X_{t(0)}$ in the motion data (see Section 3.3.1). $P_t$ and $X_{t(0)}$ can then be used for projecting and aligning the template model to the character’s shape in image $I_t$ as described in Section 3.3.2.

identical pose, and which then allows us to transform the above problem into a multi-view reconstruction setting. The main contributions, illustrated in Figure 3.4, are a technique for character shape tracking with proper occlusion handling, the above mentioned synchronization of the character’s pose, and the subsequent refinement of $M$ in order to create a final animation model $M^*$. 

Figure 3.4: (a) After the shape alignment in image $I_t$, the character’s shape is tracked in the input video to another image $I_r$ resulting in a deformed shape $S_r$. (b) In order to compensate for articulated movement, the tracked shapes and the template $M$ are synchronized into a common pose $X_\tau$ from which a refined mesh $M^*$ can be computed (c).
3 Character Reconstruction and Animation

3.3.1 Camera and Pose Estimation

The computation of the camera projection can be regarded as a classical camera calibration problem (see Section 2.1.1). However, estimating both the character pose and the camera projection from a single image without prior knowledge is a highly ill-posed problem.

There are a number of dedicated algorithms for reconstructing the pose and viewing parameters for articulated figures. [Tay00] presents a solution for reconstructing articulated objects from point correspondences for orthographic camera models which, however, leads to ambiguous configurations. [PC04] solves the same problem but accounts for projective foreshortening effects of a simplified skeleton model. A variety of other methods for recovering 3D human body poses from single images, video, or range scans are discussed in [Gav99, MG01, RAPK06]. There are also automatic methods for pose estimation such as [AT06] but they generally require segmented input images or other prior knowledge about the depicted figure and hence do not comply with our overall goals. Additionally, the user’s degrees of freedom and the flexibility for creating animations from various different types of input might be restricted by those automatic methods.

In [HDK07] we proposed a new skeleton-based estimation of the camera and pose parameters which requires only simple user interaction and exploits 3D motion capture data for the combined estimation of the camera and model pose. The estimation is based on manually selected 2D joint positions \( x'_i \in \mathbb{R}^2 \) in a reference image \( I_t \) for each skeleton joint \( i \) of the template model (see Figure 3.6). This manual user interaction generally takes less than a minute to complete, and it leads to superior results in poses which are ambiguous for automatic human pose estimators or which are difficult to estimate due to occlusions. Additionally, this approach provides more degrees of freedom to handle uncommon poses, perspective deformed figures as in photographed paintings, and a larger variety of different character types (e.g., Figure 3.6 (b)) than automatic methods which are often optimized to human characters.

Based on these 2D joint positions \( x'_i \) in \( I_t \) we have to find a frame \( \tau(t) \) of the motion capture data and a camera projection model \( P_t \) which provides the best mapping from the 3D pose \( X_{\tau(t)} \) and its corresponding 3D joint positions \( x_i \) to the user selected 2D joint positions \( x'_i \). This formulation is similar to the problem of camera estimation from point correspondences, with one additional free parameter for the 3D pose.
3.3 Shape Template Fitting

(a) Human. (b) Scarecrow.

Figure 3.6: User selected joints in an input image and the computed shape template pose and projection for two different characters.

Camera Projection

Assume for the moment that we know the pose $X_t(\tau)$ which provides an exact fit to the character’s pose in image $I_t$, i.e., there exists a camera projection $P_t$ for which $\forall i x_i' = P_{t,i}x_i$. For estimating the projection $P_t$ from these world to image correspondences there are a variety of linear and non-linear algorithms [TMHF00, HZ03] available which minimize, e.g., the re-projection error $E(P_t)$ from 3D to 2D:

$$E(P_t) := \sum_{i,j} \left( P_{t,j}x_i/P_{t,3}x_i - x_i' \right)^2,$$

with index $j \in \{1, 2\}$ referring to the $j$-th row of $P_t$ and $x_i'$, respectively. Such an unconstrained projective camera provides an optimal fit for projecting the skeleton joints of $X_t(\tau)$ to $I_t$. However, since these methods often do not impose any constraints on $P_t$, the resulting projection does not correspond to a geometrically plausible Euclidean camera model which is required for reconstruction and animation.

Instead of an unconstrained matrix $P_t$, we would like to compute a parameterized projection $P_t = KR[I|\mathbf{c}]$ (see Equation 2.2) consisting of an intrinsic calibration $K \in \mathbb{R}^{3\times3}$, an extrinsic right-handed orientation $R \in SO(3)$, and a world space camera center $\mathbf{c} \in \mathbb{R}^3$ as described in Section 2.1.1. To impose the necessary non-linear orthonormality constraints on $R$, we parameterize the extrinsic orientation by a sequence
of axis rotations $\mathbf{R}(\alpha, \beta, \gamma)$. This yields a parameterized projection matrix consisting of eleven unknowns: The intrinsic parameters for scale $\delta_x, \delta_y$, the skew $s$, and the principal point $(x_p, y_p)$, three rotation angles $\alpha, \beta, \gamma$, and the camera center position $c_x, c_y, c_z$.

Unfortunately, the unconstrained optimization of the intrinsic calibration $K$ can still lead to undesired distortion effects during reconstruction or animation, since one generally requires an (unconstrained) linear starting solution of $P_t$. Based on this solution it is not clear how to impose any meaningful constraints on $K$ during the subsequent non-linear optimization. Moreover, the optimization has to be robust to the generally quite inaccurate 2D joint positions selected by the user and to imperfectly matching character poses $X_{\tau(t)}$ in the 3D data base. Hence, we have to constrain the optimization process to yield a robust convergence to a reasonable estimate of $P_t$ despite the mismatch between the user input and the 3D motion data.

We found that a plausible camera model $P_t$ with the desired properties can be estimated by fixing the elements of $K$ except for the focal length during the optimization, and by providing a proper starting solution for the extrinsic calibration instead of an unconstrained linear solution. Reasonable assumptions for the intrinsic data of most cameras are a zero skew factor $s$, the principal point $(x_p, y_p)$ at the image center, and scaling factors $\delta_x$ and $\delta_y$ based on the image resolution, a unit aspect ratio, and a typical field of view (e.g., similar to an OpenGL projection matrix [SA06]). Thus, we set:

$$\delta_x = -\delta_y = \frac{\triangle f \text{width}}{2 \tan(\theta/2)}, \ s = 0, \ x_p = \frac{\text{width} - 1}{2}, \ y_p = \frac{\text{height} - 1}{2},$$

(3.2)

with $\theta = \pi/8$ and $\triangle f = 1$. The factor $\triangle f$ allows us to adjust the predefined field of view $\theta$ during the non-linear optimization process. The extrinsic data is initialized with $\alpha = \beta = \gamma = 0$ and a camera center at a reasonable distance from the root of the 3D skeleton model, e.g., simply a multiple of the bounding box diagonal $d$ of the model $c = (0, 0, 5d)^T$.

To compute an optimized projection matrix $P_t(\triangle f, \alpha, \beta, \gamma, c)$ based on the remaining 7 free parameters we use an iterative Levenberg-Marquardt [HZ03] solver to minimize the re-projection error $E(P_t)$ with the above starting solution. Since this initial solution corresponds to a geometrically plausible camera model, and because the optimization process optimizes this model using the above parameterization of $P_t$, the resulting camera is guaranteed to preserve all required intrinsic and extrinsic constraints. This approach converges robustly to the desired optimum without the necessity for any pa-
3.3 Shape Template Fitting

Parameter adjustments. Moreover it showed to be insensitive to changes to the initial parameters, e.g., the field of view \( \theta \) or the camera center \( c \).

Model Pose

For the above camera estimation we assumed the correct model pose to be known. Now, to find the best fitting 3D pose \( \mathcal{X}_{(d)} \) contained in our motion data base for the current user selection in image \( I_{(d)} \), we couple the camera and model pose estimation into a single optimization problem with eight degrees of freedom. We apply the above algorithm to
Character Reconstruction and Animation

all poses contained in the motion data, and keep the pose resulting in the minimal re-projection \( E(P_t) \) error (see Figure 3.7). In contrast to other pose estimation techniques, this allows us to drastically reduce the search domain for valid poses to one degree of freedom, since we only want to find the best solution in our existing sequence of poses. For a typical 3D motion sequence consisting of about 500 to 2000 frames the optimization procedure takes less than one second to compute.

It should be noted that this algorithm does not solve the general camera calibration problem, but provides a specific solution aiming at reconstructing an approximate but “plausible” camera model. Although the algorithm is quite robust with respect to the inevitable discrepancies between the user selection and the 3D data, it might fail to converge to a proper solution in cases where the user defined pose is too different from any pose within the selected 3D motion data sequence. In such cases, the user has the possibility to de-activate model joints and to choose a specific, fixed field of view. But as our experiments in Sections 3.4.3 and Section 3.5.4 show, our camera and pose estimation works robustly even on characters with a shape different from a human.

3.3.2 Template Projection and Fitting

Based on the camera projection \( P_t \) and the best matching model pose \( X_{\tau(t)} \) for image \( I_t \), the 2D reference shape \( S_t \) is created by first deforming the template model \( M \rightarrow M^{\tau(t)} \) according to \( X_{\tau(t)} \) and then projecting \( M^{\tau(t)} \) into \( I_t \) (see Figure 3.8 (a)). The triangles of this projected shape serve as an initial guess which triangles of the template model \( M \) are actually visible in \( I_t \) and hence relevant for representing the character.

In order to properly handle occluded surface parts such as limbs during animation and reconstruction, the reference shape \( S_t \) does not only consist of the visible parts of the projection of \( M^{\tau(t)} \), but stores all front-facing triangles with an additional camera space depth value for each vertex. Hence, \( S_t \) is effectively a triangle mesh with connected layers at different depths corresponding to the limbs of the depicted character (see Figure 3.8 (d)). This layered representation and the extra depth information is the key property for many subsequent processing steps described in later sections, such as detecting and resolving occlusions during shape tracking, 3D reconstruction, and animation.

The projection step provides a coarse fit of the 3D template to the character, but a significant mismatch generally remains. Assuming that the visibility of regions on the
3.3 Shape Template Fitting

character surface does not change significantly with respect to the projected template model, this mismatch can be resolved in two steps: First, we modify the pose of $S_t$ by fitting it to the user selected joints $x'_i$ (see Figure 3.8 (b)). Then, the shape of $S_t$ is aligned with the silhouette of the character as shown in Figure 3.8 (c).

In order to create a plausible pose deformation of $S_t$ we employ the as-rigid-as-possible (ARAP) shape manipulation technique [IMH05] mentioned in Section 3.1. In the first step the user selected joints $x'_i$ are used as deformation constraints. To ensure that the limbs of a character are truly rigid and that the shape deformation is mainly restricted to triangles located at joints, we place additional constraint vertices along each bone of the underlying skeleton and integrate these vertices to the tessellation of $S_t$. Since the mesh layers of $S_t$ (i.e., the different body parts) are conjoined, e.g., at the shoulders, this deformation properly retains the impression of a connected body.

After the pose adaptation, the boundary vertices of $S_t$ have to be aligned to the character’s silhouette. Although semi-automatic approaches exist, those techniques are often error prone, especially with respect to the deficiencies and problems occurring in videos of moving people, e.g., frequent (dis-)occlusions or video compression artifacts. Previous work such as [AHSS04] has shown that manual curve-based editing is a practical

Figure 3.8: This figure shows the central steps of the template projection and fitting procedure and the resulting layered shape representation.
and efficient tool for this task. Hence, we utilize a similar editing metaphor and let the user match the shape boundaries manually. The non-boundary vertices of $S_t$ are then repositioned automatically using an additional ARAP deformation step with the boundary vertices as deformation constraints. This is crucial in order to redistribute the inner triangles within the adapted shape while preserving their perspectively distorted aspect ratio (see Figure 3.8 (b) and (c)). The result of the template projection and fitting step is the final shape representation $S_t$ in image $I_t$. For the subsequent reconstruction and animation we have to distinguish two cases: One or multiple input images.

3.4 Single Input Views

Given only a single input image $I_0$ and a 2D shape $S_0$ computed as described in the previous section, it is generally not possible to compute an exact 3D model for animation. Moreover, the visible texture information available from a single view is generally very restricted due to occlusions. Hence, the animation of a partially reconstructed and textured 3D model using general motion data including, e.g., rotations, would lead to artifacts since originally occluded regions are likely to become visible. Therefore, in the case of a single input image we propose a technique which animates the character shape $S_0$ directly in image space, textured with the original image content [HDK07].

3.4.1 Texture Completion and Shape Initialization

For an unconstrained animation, occluded regions on the character’s body and on the remaining background image have to be completed. The layered representation of the character shape shown in Figure 3.8 (d) allows us to decompose $S_0$ by a front-to-back depth-peeling algorithm. For each layer including the image background we complete dis-occluded texture regions using a method for image completion [PSK06]. Poisson-matteing [SJTS04] is used to extract an alpha matte for each layer (see Figure 3.9).

The initial model fitting is concluded by applying a final ARAP manipulation step $S_0 \rightarrow S^*$, transforming the character’s image pose as defined by the user selected joints into the best matching 3D pose $X_{t0}$ by using the projected joint positions as vertex constraints, similar to Section 3.3.2. The resulting shape $S^*$ is the initial starting shape used for the subsequent animation (see Figure 3.9 (c)).
3.4 Single Input Views

(a) Input image.  (b) Completed background.  (c) Initial $S^*$.  

Figure 3.9: Exemplary texture completion and shape initialization.

3.4.2 As-Similar-As-Possible Shape Deformation

Using the ARAP shape deformation technique the textured shape representation $S^*$ could easily be animated by constraining the joint vertices to the 2D positions obtained by sequentially projecting consecutive poses $X_{\tau}$ into the image using the estimated camera projection $P_0$. However, since the ARAP approach aims for a rigid preservation of the original 2D triangle shapes and hence ignores perspective scaling and distortion effects, unrealistic deformations such as thinning or thickening of the character’s limbs may occur when they are changing their orientation relative to the image plane (see Figure 3.10). Our solution to this problem is a generalization of the ARAP approach to an as-similar-as-possible (ASAP) technique which properly considers perspective changes of the triangles in $S^*$ due to the projection of 3D motion.

The original ARAP approach [IMH05] for transforming a source shape subject to a set of vertex position constraints consists of three consecutive processing steps $S^* \rightarrow S^I \rightarrow S^F \rightarrow S^D$, which try to preserve the shape of the original triangle faces $f \in S^*$ (see Figure 3.11). In a first step an intermediate shape $S^I$ is computed for the vertex constraints using a Laplace-based deformation [SCOL*04]. Since this step does not prevent an arbitrary scaling of the triangles $f \in S^*$, a second scale adjustment step rigidly fits each face $f$ in a least squares sense to the corresponding face in the
intermediate shape using translation and rotation only. This results in a disconnected set of triangles $S^T$. This set is then converted into the final deformed shape $S^D$ by computing a weighted average of corresponding vertex positions in $S^T$. The details of this technique are described in [IMH05].

For the ASAP shape deformation we still perform this sequence of deformation steps for every animation frame and pose $X_\tau$. But instead of rigidly fitting the original triangles $f \in S^*$ to the intermediate shape $S^I$ to obtain $S^F$, we estimate their perspective distortion and fit the distorted triangles $f'$ to $S^F$. By this method we eventually generate a deformed shape $S^D$ whose faces are as similar as possible to the perspective distorted triangles $f'$.

To estimate the perspective distortion of a triangle $f \in S^*$, we exploit the 3D information contained in the motion data as illustrated in Figure 3.12. With each bone $b = x_i - x_j$, defined by two neighboring joints in a pose $X_\tau$, a local coordinate frame $L$ is attached which changes according to the bone’s movement. This change of the bone’s
3.4 Single Input Views

Figure 3.11: The first step $S^* \rightarrow S^I$ of the ARAP shape manipulation deforms the shape according to a set of vertex constraints but does not preserve the triangle scales. The second step $S^I \rightarrow S^F$ fits the original triangles of $S^*$ into the intermediate shape $S^I$. The final deformed shape $S^F \rightarrow S^D$ is computed by minimizing the difference between these fitted triangles and their corresponding triangles in $S^I$.

orientation provides the necessary information to compute the triangle’s perspective distortion. Let $b_0$ be a bone’s orientation in the initial pose $X_{\tau(0)}$. Its local frame $L_0$ is defined by three vectors $l_x = b_0 \times (c - x_i)$, $l_y = b_0$, and $l_z = l_x \times l_y$, with $c$ being the camera center (see Section 3.3.1). Likewise, we define the local frame $L_\tau$ for the same bone as it is oriented in the target pose $X_\tau$. We first consider triangles in $S^*$ which are uniquely associated with a single skeleton bone $b_0$. Every 2D vertex $v$ of such a triangle is un-projected to 3D by mapping it to a point $v_0$ on the plane spanned by the vectors $l_x$ and $l_y$ of $b_0$’s local frame $L_0$. Then, $v_0$ is expressed in local coordinates with respect to $b_0$, i.e., $\tilde{v}_0 = L_0^{-1}(v_0 - x_i)$. During animation, the 3D position of a vertex in another pose $X_\tau$ of the motion data is obtained by transforming its local coordinates $\tilde{v}_0$ back to global coordinates $v_\tau = L_\tau \tilde{v}_0 + x_i$, and then projecting this point back to the image plane $v'_\tau = P_0 v_\tau$. Applying this procedure to all three vertices of a triangle $f$ yields the correctly distorted triangle shape $f'$ for the ASAP deformation. For triangles near a skeleton joint which are associated with several bones in the original shape template $M$ (see Section 3.2.1), we apply the same procedure for each associated bone and compute the weighted average position in 3D. Finally, instead of using a planar proxy for the triangles associated with a bone, the depth information which was stored during the template projection step with
Figure 3.12: Illustration of the projective triangle correction. (a) Each face \( f \in S^* \) is projected from image space into the local coordinate system \( L_0 \) of the corresponding bone in the initial pose \( X_{\tau}(0) \). (b) The projectively deformed triangle \( f' \) is generated by updating \( L_0 \rightarrow L_\tau \), and re-projecting the triangle vertices back into the image.

every vertex of \( S_0 \) (see Section 3.3.2) can be exploited to simulate a curved surface. As demonstrated in Figure 3.10, this effectively avoids cardboard effects during animation at grazing viewing angles and results in a more three-dimensional appearance.

To simplify the original ARAP approach [IMH05], we compute the rigid fitting step \( S^I \rightarrow S^F \) by a closed form solution for the optimal rotation, which minimizes the squared error \( \sum_i \| p_i - q_i \|^2 \) over the vertices \( p_i \in f' \) and the vertices \( q_i \) of the corresponding triangle in \( S^I \). First, we translate \( f' \) into the center of gravity of the vertices \( q_i \). Then we compute the optimal 2D rotation angle \( \psi \) for \( f' \) using [Hor87]:

\[
\tilde{R} = \sum_i \left( \langle p_i, q_i \rangle, \langle p_i, q_i^\perp \rangle \right), \quad (\cos \psi, \sin \psi) = \frac{\tilde{R}}{\|\tilde{R}\|} \quad (3.3)
\]

This fitting step is computed for each triangle, and the resulting deformed shape \( S^F \rightarrow S^D \) is computed as in [IMH05] by a weighted averaging of the unconnected triangle vertices.

3.4.3 Animation and Rendering

The final animation is generated by simply updating 2D joint positions \( x'_i = P_0 x_i \) for subsequent poses \( X_\tau \) from the 3D motion sequence. These positions are used as constraints to compute an intermediate shape \( S^* \rightarrow S^f \). Projectively corrected triangles
3.4 Single Input Views

(a) Painting of Napoleon.  (b) Scarecrow model.  (c) Painting of a Skater.

Figure 3.13: Input images and frames from animations generated with the proposed technique. The painting of Napoleon (a) and the Skater (c) have been animated with forward motion. The Scarecrow (b) performs a jumping and waving motion.

computed by the above algorithm are used to generate the final deformed mesh $S^I \rightarrow S^F \rightarrow S^D$ which is then rendered in real-time using textured OpenGL triangle meshes with enabled alpha blending [SA06] for the boundary matting. Triangle depths for proper occlusion handling are available from the depth data stored for each vertex of $S_0$ as described in Section 3.3.2.

New animations can be generated simply by exchanging the 3D motion. All previously computed steps do not have to be re-computed, so that applying a different motion to a character generally is a matter of seconds. Input images and output renderings of image animations generated with this approach are shown in Figures 3.13 and 3.14. These results show that it is possible to animate a variety of different character types
3 Character Reconstruction and Animation

Figure 3.14: More input images and animation frames. The Meerkat (a) and Philipp (b) perform the Lambada and the Chicken Dance, respectively. A complex animation of a standing statue of Engels taking a seat besides Marx, which involves a significant deformation of the original shape, is shown in (c).

containing even complex details such as straw or fur from just a single input photograph or painting with a wide range of 3D motions.

3.5 Multiple Input Views

With multiple input views $I_n, t \in [0, n]$, e.g., in the form of an input video, more information about a character’s actual 3D shape and appearance can be derived from the images. As mentioned in Section 3.2 the difficulty in our problem setting is that we aim for generating an animated character model from a single uncalibrated video showing a
3.5 Multiple Input Views

moving person. This type of input data violates fundamental assumptions of standard multi-view model reconstruction, such as all images showing the character in an identical pose with accurate point correspondences between the images.

The main insight presented in this section is that this problem setting can effectively be transformed into a multi-view reconstruction setting with multiple synchronized cameras by a pose synchronization of the character and subsequent 3D reconstruction [HDHK09]. The major steps to achieve this are a new mesh-based shape tracking approach which tracks the deformation of the character’s shape over a continuous sequence of images, and which enables our system to establish dense surface correspondences with proper handling of occluded surface parts. The resulting sequence of shapes is synchronized with the template model into a common character pose which allows for a subsequent refinement of the 3D shape of the template model (see Figure 3.4).

More formally, given a reference shape $S_t$ for an image $I_t$ from the set of input views the process of pose synchronization and reconstruction works as follows: First, the shape $S_t$ is tracked in 2D image space over short subsequences of images $\{I_s, \ldots, I_t, \ldots, I_r\}$ in the input video, as long as the visible triangles of $S_t$ are consistent with the visible surface parts of the character. Due to the one-to-one correspondence of the triangles of $S_t$ to a subset of triangles of $M$, the resulting tracked shape sequence $S_t = \{S_s, \ldots, S_t, \ldots, S_r\}$ captures the deformation of these triangles for different viewing positions (see Figure 3.15). Under the assumption that the character has an identical pose in all images (as in classical multi-view stereo) it would be possible to compute a 3D model by casting and intersecting rays from the camera centers through corresponding vertices of the shapes in $S_t$ (see Section 2.1.2). However, the triangles are also subject to deformation caused by articulated motion of the character. Moreover, the pose $X_{\tau(t)}$ and the deformed template $M^{\tau(t)}$ computed in Section 3.3 are generally just an approximation to the true pose of the character in the images (see Figure 3.6). These facts prevent a straightforward refinement of the 3D vertex positions of $M$.

The main idea of our pose synchronization is to compensate for the deformation caused by articulated motion and the pose mismatch. The aim is to bring all shapes $S_j \in S_t$ and the 3D template model $M$ into a common pose in order to refine the vertex positions of $M$. Assume we have computed skeleton-based camera parameters for all images as described in Section 3.3.1 and we know a motion frame $\tau$ from the motion data whose projection is most similar to the poses represented by all shapes $S_j \in S_t$. We then mimic a synchronized multi-camera setup by deforming each shape $S_j \in S_t$ as-rigid-as-possible
3 Character Reconstruction and Animation

to a shape $S^j_t \in S^j_t$ so that the shapes in $S^j_t$ are a representation of the character in an identical pose $X_t$ for different viewing positions. This step is illustrated in Figure 3.4 (b) for the two shapes $S^j_t$ (Figure 3.3 (c)) and $S^r_t$ (Figure 3.4 (a)). By deforming the 3D template model $M \rightarrow M^*$ according to this common pose as well, its 3D shape can be refined using basic image-based reconstruction as described in Section 2.1.2, i.e., we simply compute a new 3D position for each vertex by a triangulation of the viewing rays through corresponding 2D vertex positions of the shapes $S^j_t$. By combining partial reconstructions from multiple shape sequences we can eventually update all parts of the template model which are visible in the input video to a new output model $M^*$ as shown in Figure 3.4 (c). The following sections describe these steps in detail.

A shape sequence $S^j_t$ is initialized with its reference shape $S^r_t$ as described in Section 3.3. The next step then consists of tracking the deformation of $S^j_t$ over a sequence of images.

3.5.1 Shape Tracking

For tracking the motion of objects through a sequence of images a variety of solutions such as feature tracking [BM04], correspondence estimation [Low04, BTV06], and optical flow [BBPW04] have been proposed. However, in the context of this work these techniques have a number of significant disadvantages. For instance, tracking rectangular windows centered, e.g., at the mesh vertices has the drawback that it is difficult to handle occlusions or vertices at the silhouette of a shape $S^j_t$. Techniques for correspondence estimation have similar problems and often do not create dense and reliable matches for untextured regions. This is a particular problem for the quite limited character size at standard video resolutions. However, we need such dense correspondences in order to track shape deformations from one image to another. Due to regularization, methods for dense optical flow often have problems at foreground/background discontinuities as well, and the integration of the character segmentation resulting from the initial model fitting step (see Section 3.3) is not straightforward.

Although there are solutions which address some of these problems, an important requirement in our problem setting is that we have to keep track of the complete limbs of a character even if they are only partially visible in an image, since they might become dis-occluded eventually. Most existing techniques either do not support these requirements or do not consider all of the available information about the character segmentation and occlusions. We therefore propose a layered, mesh based tracking
3.5 Multiple Input Views

Figure 3.15: Example of a tracked shape sequence $S_t = \{S_s, \ldots, S_t, \ldots, S_r\}$. The bottom row shows the distortion of a selected triangle $f_k$. During the mesh tracking this deformation is approximated by an affine mapping $A_k$.

approach which exploits the depth information in the triangle meshes $S_t$ in order to resolve occlusions and to keep track of partially occluded limbs.

Given two successive images $I_i$ and $I_{i+1}$ and a shape $S_t$ in image $j$, our goal is to compute a displacement field attached to the vertices of $S_t$, i.e., a displacement $d_i$ for each vertex $v_i$ of $S_j$. The vertices $v'_i$ of $S_{j+1}$ then become $v'_i := v_i + d_i$. Each triangle face $f_k$ of $S_j$, together with the respective transformed face $f'_k$ of $S_{j+1}$, defines an affine transformation $A_k : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ between the images $I_j$ and $I_{j+1}$ (see Figure 3.15). We formulate the matching process as a global optimization problem that minimizes the sum of triangle errors. The per-triangle error for each pair $(f_k, f'_k)$ of corresponding triangles is computed by integrating over the squared intensity differences of the respective image.
areas in $I_j$ and $I_{j+1}$. The desired displacement field then is a minimum of the objective function

$$E_{\text{data}} = \sum_{f_k \in S_j} \sum_{p \in \Omega_k} \left( I_j(p) - I_{j+1}(A_k(p)) \right)^2,$$

(3.4)

where $\Omega_k \subset \mathbb{N}^2$ denotes the set of image pixels covered by triangle $f_k$ in image $I_j$. To alleviate the distorting effect of image noise and to prevent convergence to a false local minimum, smoothness of the resulting displacement field is enforced by an additional error term

$$E_{\text{smooth}} = \sum_{i \in V(S_j)} \omega_i \sum_{j \in N_i} \omega_{i,j} \left\| d_i - d_j \right\|^2,$$

(3.5)

which imposes rigidity on the tracked mesh. $V(S_j)$ denotes the set of vertices of the shape $S_j$ and $N_i$ denotes the 1-ring neighbors of vertex $i$. We chose the standard chordal weights $\omega_{i,j} := \left\| v_i - v_j \right\|^{-1}$, $\omega_i := \sum_{j \in N_i} \omega_{i,j}$. The complete objective function then is

$$E = E_{\text{data}} + \lambda E_{\text{smooth}},$$

(3.6)

which is minimized using a standard Levenberg-Marquardt minimization procedure to determine the vertex displacement field between pairs of successive images. In order to reliably match large motions of the character, a multi-resolution matching approach is applied. The shape meshes for coarse resolution images are generated using a variant of the iterative remeshing approach of [BK04a], adjusted to correctly preserve shape boundaries and with an appropriate target edge length. In our experiments we found two resolution levels (i.e., the original images and one coarser resolution) to be completely sufficient.

Although this approach is formulated using conventional image and smoothness energies, the combination of projected shapes $S_t$ and mesh-based tracking is the key to resolving the complex occlusions in the input videos (see Figure 3.16): The generation of the initial shape $S_t$ by projecting the 3D template mesh to 2D image-space provides per-vertex depth information in addition to the image positions. During the rasterization of all triangles $f_k$ in order to obtain the pixel sets $\Omega_k$, this depth information is taken into account such that each triangle is assigned truly visible pixels only. If a triangle is completely occluded (because, e.g., it lies on the characters torso and is covered by an arm) it hence is assigned no pixel at all. For the objective function $E$ this has the desirable effect that $E_{\text{data}}$ is zero for the respective triangles. Only the regularization
3.5 Multiple Input Views

Figure 3.16: Four frames of a tracked shape sequence showing a person from different viewing angles. Red regions correspond to detected occlusions, e.g., on the person’s left arm, the leg, and on the torso.

term $E_{\text{smooth}}$ is non-zero for the involved vertices, resulting in a plausible transformation of the occluded parts of the shape $S_i$. Since the rasterization is performed in every image of the shape sequence, eventually dis-occluded triangles are correctly recognized and only then add a meaningful contribution to $E_{\text{data}}$. The choice of $\lambda$ depends on the orders of magnitude of the energies $E_{\text{data}}$ and $E_{\text{smooth}}$. However, it showed to be rather insensitive to the actual image data, so that we could keep it constantly set to $\lambda = 2$ throughout all of our experiments.

After tracking $S_j \rightarrow S_{j+1}$ the sub-pixel accurate surface matching resulting from this approach effectively corresponds to an exact (stereo) correspondence estimation. The 2D skeleton joint positions are updated as well by pulling them along the computed displacement field. Besides the possibility to handle occluded surface parts, the tracking automatically preserves the segmentation of the character and the background, and it is straightforward to integrate prior deformation information. For example, in cases where the tracking fails (e.g., for large displacements) the user can provide hints to the system simply by moving skeleton joints.
This tracking loop is iterated both forwards and backwards through the video as long as the reference shape $S_t$ is reasonably consistent with the visible surface areas, i.e., until the viewpoint change with respect to the character in image $I_t$ (and hence the amount of (dis-)occluded surface area) becomes too large. Although automatic error thresholds could be formulated based on, e.g., the color mismatch of erroneously tracked triangles, this is again an error prone process similar to the boundary snapping in Section 3.3.2. Hence, we give the user interactive control when to abort the tracking process. The result is a shape sequence $S_t = \{S_s, \ldots, S_t, \ldots, S_r\}$, tracking the character’s deformation from the reference image $I_t$ through adjacent frames in the input video.

Depending on the number of available different viewpoints in the video with respect to the character, one or more shape sequences are generated starting at different reference frames. In order to reconstruct the 3D shape of the character, the next processing step mimics a temporal synchronization of the images by a pose synchronization of the tracked shapes. This compensates deformation due to articulated movement.

### 3.5.2 Pose Synchronization

In order to synchronize the poses of shapes $S_j \in S_i$, each $S_j$ has to be deformed such that it corresponds to the 2D projection of a common 3D skeleton pose $X'$ in its respective view $I_j$. Thus, in a first step, camera models $P_j$ are computed using the procedure described in Section 3.3.1. Since the 2D skeleton joints are pulled along with the shapes during the mesh tracking, one generally has to do only minor adjustments to the joint positions in order to get a stable calibration. The common pose used for the synchronization can then be found by evaluating the joint reprojection error of each possible pose with respect to all shapes $S_j \in S_i$ similar to Equation 3.1.

The actual synchronization $S_j \rightarrow S'_j$ is then performed by an additional 2D ARAP deformation step as in Section 3.3.2 with the projected 3D skeleton joints of the common pose as deformation constraints. This step is again motivated by the assumption that the overall visibility of surface triangles remains valid due to continuous character and camera movement while the perspective changes of the surface triangles have been properly captured by the shape tracking. Therefore, the 2D ARAP deformation into the common pose is a reasonable approximation to the corresponding 3D pose change (see Figure 3.17). The result is a synchronized shape sequence $S_i \rightarrow S'_t$. In practice the synchronization is done only for the reference shape $S_i$ and the two shapes $S_s, S_r$. 

46
3.5 Multiple Input Views

Figure 3.17: Example for the pose synchronization. The top row shows shapes $S_1$, $S_t$, $S_r$ of a shape sequence $S_t$ and the template $M$ before the synchronization. In the bottom row these shapes and the model have been synchronized to a common pose $X^*$. Pose changes are visible around the legs and the arms.

at the boundaries of the tracked video interval, since they generally have the largest baseline between the reconstructed camera centers and hence result in the most stable 3D reconstruction.

Besides this intra-shape sequence synchronization, there is an additional synchronization issue between overlapping shape sequences $S_i$ and $S_j$ generated from different reference images $I_i$ and $I_j$, respectively (see Figure 3.18). Although the shape tracking per sequence produces pixel accurate correspondences, it is not ensured that a vertex of the template model $M$ ends up at exactly corresponding surface points in both sequences, since the boundary snapping and shape tracking are done for each shape sequence indi-
Character Reconstruction and Animation

The primary effect of this mismatch are ghosting artifacts when merging the texture information from different shape sequences into a single model. This problem is addressed by computing interpolated 2D vertex positions for each shape similar to the concept of epipole consistent camera weighting in Unstructured Lumigraph Rendering [BBM’01]. Given several shape sequences \( S_i \) which have overlapping shape tracking domains in an image \( I_k \), and let \( v_i \) be a corresponding 2D vertex position in the respective shapes \( S_{i,k} \in S_i \). The updated vertex position \( v^\star \) is computed by a weighted contribution from each shape sequence \( v^\star = \sum \omega_i v_i \), based on the temporal distance of their respective reference frames to the current image \( I_k \):

\[
\omega_i = \begin{cases} 
\infty, & i = k \\
\left( \frac{1}{i-k+1} \right)^{\beta}, & \text{else}
\end{cases}
\]

The final weight is normalized by \( \omega_i = \tilde{\omega}_i / \sum \tilde{\omega}_i \). For shape sequences which have image \( I_k \) as their reference image (\( \tilde{\omega}_k = \infty \)) we use the original vertex positions in \( I_k \) and set all other weights \( \omega_i = 0 \). The exponent \( \beta \) controls the influence of shapes which are distant to the reference image and was set to \( \beta = 3 \) in our experiments. All vertex positions \( v_i \) are then replaced by the vertex position \( v^\star \).

Obviously, the mismatch between different shape sequences becomes larger with increasing deviation of the depicted character to the 3D template model \( M \). Hence, it is of relevance mainly for non-human shapes such as the Scarecrow example (see Figure 3.15). However, by initializing the vertex positions in a new shape sequence with tracked vertex positions from a previously generated sequence, it is possible to compute partial reconstruction with pixel accurate surface points even for more complicated models. For the
3.5 Multiple Input Views

Figure 3.19: Refinement of the template model $M^r$ (a) without (b) and with (c) pose synchronization of the shapes in Figure 3.17.

3D reconstruction this mismatch is less relevant since the ray intersection and merging of partial reconstructions explained in the following already compensates this effect.

3.5.3 Model Refinement

With the synchronized shapes $S^r_j \in S^r$ and the corresponding camera projection $P_j$, the character’s shape represented by a sequence $S^r$ can be used to update the template model $M$. First, the template is deformed into the common pose $M \rightarrow M^r$ (see Figure 3.17). Then, the 3D positions of its vertices are refined by simple triangulation of the corresponding viewing rays through the 2D vertices of the shapes in $S^r$ as described in Section 2.1.2.

Because a single shape sequence carries information only about the vertices visible in its reference image, each sequence contributes only a partial model update. However, since the vertex positions are refined in the local frame of $M^r$, each partial reconstruction can also be animated by changing the target pose. In order to compute the final output model $M^r$, all partial 3D reconstructions are deformed from their respective pose $X$. 
back into the base pose $\mathbf{X}_b$ (see Figure 3.2). The model update is then finalized by averaging the positions of all reconstructed vertices into the final model $\mathbf{M}^\star$.

The effect of the pose synchronization on the model refinement is shown in Figure 3.19: Without the synchronization the ray intersection computes wrong depth estimates for the mesh vertices due to the pose differences between the 2D shapes and the template. This causes severe shape distortions such as the person’s left knee bending backwards (b). Moreover, the refined vertices are not properly repositioned with respect to the coordinate system of the template model, which results in self intersections of the original template surface and the refined vertices. These intersections become even more problematic when merging different partial reconstructions. The pose synchronization resolves these problems (c) and the vertex positions of the template can be updated properly to match the person’s shape. A reconstruction example of a more complex shape is shown in Figure 3.20.

### 3.5.4 Animation and Rendering

The reconstructed model $\mathbf{M}^\star$ is now ready for animation, since the original bone weights associated with every vertex (see Figure 3.2 (b)) with respect to the underlying skeleton
3.5 Multiple Input Views

do not have to be recomputed. Although this may lead to problems in the case of drastic deviations of the reconstructed model from the original template, even for the quite complex Scarecrow model the original vertex weights are still fine. This is due to the fact that the actual association of vertices to the limbs and torso does not change globally.

View-dependent surface textures for $M^\star$ are retrieved from the character aligned shapes $S_j$ in the input images $I_j$. The visibility of a triangle in a given image can be computed from the shapes $S_j$ similar to Section 3.5.1 since the shape’s depth information is preserved throughout the whole process. For each visible triangle the texture coordinates for an input image are given by the 2D vertex positions $v^\star$ (see Section 3.5.2). For view-dependent texture mapping, a pencil of directions $p_j$ from the triangle center to the camera centers $c_j$ of the input images is stored. During animation and rendering we transform this pencil with the local coordinate frame of its associated triangle, and then compute a weighted contribution from each relevant input image $I_j$, which depends on the angle between the directions $p_j$ and the direction to the virtual camera. As in Equation 3.7 the used weighting is again similar to the weighting function proposed in the ULR framework [BBM‘01] (see Section 2.3). The effect is that, given a new viewpoint, we can render each triangle with view-dependent texture mapping as similar as possible to its original appearance in the input video. These view-dependent textures provide a dynamic, much more realistic reproduction of the appearance of folds and shadows on the character than simple static texturing. Due to the underlying 3D representation it is also straightforward to add global illumination and shadowing effects. A simple way for integrating the model into a new scene is to perform the character based camera estimation from Section 3.3.2 on a character already existing in the scene.

Figures 3.21 to 3.24 show a number of different character animations created with the proposed system, ranging from moving people acquired with a standard video camera over more complex shapes such as the scarecrow model to hand-drawn figures. All reconstructions are based on the template model shown in Figure 3.2.

The Doro example was created from a video recorded with a hand-held camera, showing her turning around on the spot. The turning motion involved a significant amount of movement and rotation of the limbs. The full body reconstruction is based on five shape sequences: three overlapping sequences of the front, and two sequences of the back. Frames of resulting animations with new motions are shown in Figure 3.21. The Felix example in Figure 3.22 was generated from three shape sequences, resulting in a
reconstruction of the frontal part. Although this does not allow for full 360 degree views, it is possible to animate the model with motions involving restricted rotations of the limbs and torso.

The Scarecrow model was captured using a turn-table and reconstructed with three shape sequences. Due to the texture-rich surface and only minor illumination changes, the tracking procedure worked fully automatic over the whole sequence despite the complicated shape. Some of the features such as the straw-hair could not be captured because these structures deviated too strong from the template mesh. However, the reconstructed 3D mesh is a faithful reproduction of the tracked surface parts (see Figure 3.20). Simple shape inflation for simulating 3D effects [PFWF00] or the original

Figure 3.21: The Doro example. The top row shows three images from the input video sequence and a front view of the reconstructed model in a new pose. The bottom row shows an animation of the model with jumping and walking motion overlayed on a video.
3.5 Multiple Input Views

Figure 3.22: One of three reference views of the Felix example and two frames from an output animation with walking motion.

Figure 3.23: Output animation of the reconstructed Scarecrow model with jumping and waving motion.

depth values of the template model instead of using the proposed shape tracking and pose synchronization would not be able to reproduce the concavities at the head or the trousers. Frames of a jumping and waving animation are shown in Figure 3.23.

Finally, the Paperman example in Figure 3.24 was generated from three hand-drawn images of the figure in slightly different poses and reconstructed from a single shape sequence. Obviously, an accurate correspondence estimation is problematic due to the hand-drawn and hence quite inconsistent shape and texture between the images. In order to produce a more three-dimensional shape, we mapped a logo onto a sphere and
pasted images of this sphere as seen from different viewpoints to the input images. The shape tracking was performed at a coarser resolution in order to filter the fine-scale texture inconsistencies. Although the reconstructed model (apart from the belly) is quite flat, our method allows to generate 3D animations and rendering effects even for such hand-drawn pictures. Similar to the single image case described in Section 3.4 the motion data for generating new animations can be easily exchanged without having to recompute the reconstruction.

3.6 Discussion

In this chapter we motivated the use of a generic character template as a shape representation for image-based character reconstruction and animation. The main advantage of this approach is its flexibility to handle a wide range of different characters and input types, ranging from single images to uncalibrated video of moving persons.
For single input images, from which a 3D reconstruction of the character is not possible, the generic shape template allows to generate a proper view-dependent image space representation which respects the complex occlusions and topological changes that have to be considered when creating animation shapes for different viewpoints. For these 2D shapes, we presented an approach to create plausible deformations from motion capture data with simulated perspective effects by using the depth information from the original 3D template model for a more realistic appearance. A natural restriction of the single image approach is the problem that motions involving a complete turn of the character cannot be animated, since this would involve a smooth on-the-fly switch between 2D shape templates with different topology. Moreover, it would be necessary to complete a considerable amount of missing texture information. However, motions involving slight rotations can be easily handled since the ASAP deformation of the shape requires only projected 2D joint positions instead of the full bone orientations of the underlying skeleton. As a consequence, the single image approach can realistically preserve even complex character silhouettes as for the Scarecrow model.

Given more input views our method can be used to reconstruct the actual 3D shape and full texture of the character. One of the key contributions here was a layered mesh-based tracking approach which establishes the required dense correspondences throughout the image sequence and which properly handles the complex occlusions occurring with moving characters and changing viewpoints. Moreover, we showed that articulated movement can be compensated by a pose synchronization step, which enables a subsequent update of the original 3D template model to match the shape of the character. On the one hand, a disadvantage of this explicit textured 3D mesh is the loss of support for complex silhouettes. On the other hand, however, the major advantage of the resulting model is that it supports unconstrained animations from 3D motion data.

Our approach based on a single generic shape template alleviates a number of restrictions discussed in Section 3.1, such as constrained multi-view capture setups, missing support for general character shapes, or a rigging and skinning process (e.g., a proper computation of vertex weights for animation) which has to be performed individually for each single model.

Nevertheless, there remain interesting problems for future research. For example, although the reconstruction quality of this method scales with the number and quality of the input images, there obviously remain potential sources of error such as an inaccurate boundary snapping or joint positioning by the user, or the approximative camera
estimation. Hence, it is difficult to compete with the high reconstruction quality possible in controlled studio setups [BSB'07]. Although the manual steps involved in our system provide increased flexibility to handle a variety of different types of input data, it would nevertheless be interesting to integrate automatic techniques for pose estimation and silhouette detection for human characters in common poses, in particular for persons under articulated motion [GEJ'08]. However, the currently most time consuming part is the shape tracking described in Section 3.5.1, which takes about 30 to 40 minutes to complete for a single shape sequence consisting of 30 to 60 video frames. We expect significant speedups of the overall pipeline with optimized implementations using, e.g., CUDA [Buc07]. As another example, the triangle sizes of the current generic shape template used in all presented examples are optimized for standard video resolution. A dynamic tessellation and refinement for higher input image resolutions would allow to reconstruct character models with a more detailed surface. Finally, too strong non-rigid movement in an input video that cannot be explained by articulated motion (e.g., clothes) can obviously lead to artifacts during the reconstruction and should be addressed in future work.

Nevertheless, the proposed character representation based on a generic shape template allows to generate plausibly animated models from a variety of different character types, while the photo-realistic texture of the original input images is preserved.
4 Interactive Free Viewpoint Rendering

The mesh-based shape template introduced in the previous chapter has been designed as a specific representation for animated articulated character models. However, for the image-based reconstruction and the rendering of more general scenes, the generation of a dedicated shape template for objects and scenes of any type is impractical. One example for this is the field of free viewpoint rendering. Here, the basic idea is to synthesize novel output views of an arbitrary, unstructured scene from a given set of input images (see Figure 4.1). For these applications one either requires a dense sampling of the input view space using a large number of images [LH96] which can only be acquired with specific and often significantly constrained capture setups, or one needs a geometric scene proxy to merge content from a sparser set of input images [GGSC96, PCD∗97, BBM∗01]. In order to achieve a high rendering quality and versatility for general scene types, this chapter focuses on the latter case of image-based rendering using geometric proxies.

First approaches for the generation of explicit geometric scene proxies have been based on voxelized scene representations [SD97], merging of multiple depth maps [KPV98], or layered depth images [SGwHS98]. But despite a large body of research in the field of stereo vision, the accurate and robust generation of geometry solely from a set of input images is still a difficult problem [Mid08a, Mid08b]. Therefore, many techniques for free viewpoint rendering either assume that the necessary depth information is available using, e.g., z-cams or active stereo techniques based on structured light, or they employ rectified stereo from image pairs and therewith are restricted with respect to the camera setups and scene types. Moreover, an important issue during the proxy generation is an exact representation of object silhouettes in order to prevent rendering artifacts at depth discontinuities. Due to the involved issues, many methods are optimized for a particular application domain such as free viewpoint video of human actors [MBR∗00, CTMS03] with controlled setups and specifically designed object proxies [ABB∗07, WWG07], or
continuous video with small inter-frame displacements [ZJWB08]. More recently, point-based approaches have shown great potential for 3D video recording and real-time rendering [WLG04, WWCG07].

Beside the proxy generation, the second central problem is the actual synthesis of a new view from the given scene representation. Merging the contributions from all relevant input images in order to estimate the final color of an output pixel generally involves significant computational effort. Because of the ill-posed nature of the scene proxy generation and visibility estimation, sophisticated filtering and optimization techniques are often inevitable in order to achieve an acceptable output quality. Thus, some methods restrict themselves mainly to smooth view interpolation between images [VBK02, ZKU’04], or reduce the accuracy of the corresponding camera blending field [BBM’01]. Instead of an actual fusion of content from different images, some approaches focus on sophisticated navigation and rendering algorithms for exploring large collections of photos [SSS06]. However, for reconstructing fine-scale view-dependent illumination effects of complex surfaces, it is often necessary to consider the colors from a large number of input views simultaneously. Correspondingly, many methods still require significant offline processing [FWZ05], often combined with the need for storing huge amounts of data. All these approaches are capable of producing high quality synthesis results, but generally require computationally intensive pre-processing. Hence, an increasing number of techniques such as [YPYW04, WVVV05, GV07] has presented solutions to exploit the programmability of current GPUs for implementing efficient real-time view synthesis. In
[HKss] we presented an approach to interactive free viewpoint rendering which alleviates many of the above mentioned issues and which is the main topic of this chapter.

4.1 Conceptual Overview

Our main objective is a generic and flexible scene representation, which enables a fast generation of geometric proxies from images without posing too many constraints on the camera configuration or the scene. Furthermore, this representation should allow to combine content from the input views efficiently into new photo-realistic output images for novel viewing positions and camera parameters. In this context, scene representations based on points, splats, or particles have shown a high flexibility in terms of geometry representation and GPU-based processing (see Section 2.2.1). This chapter describes an integrated generic and scalable system that uses particle-based scene proxies and addresses all relevant processing steps. The main features of the proposed approach are:

Versatility Our generic particle-based scene representation supports the creation of geometric proxies from a set of input views for arbitrary scene types and complexities. The particle shapes are sensitive to depth discontinuities at object silhouettes and allow a proper handling of occlusions in the input images. The proxy generation employs a new photo-consistency estimation and optimization technique for multi-view stereo which supports unconstrained camera configurations.

Accuracy The proposed method achieves a high accuracy with respect to the reconstruction of the three-dimensional scene geometry as well as the scene appearance. Geometric accuracy is ensured by the above mentioned multi-view stereo formulation for volumetric scene elements, silhouette aware sampling in the input images, and an efficient continuous proxy optimization procedure. In the rendering phase a pixel-accurate camera blending field is computed for a photo-realistic reproduction of the input view properties.

Efficiency By decoupling the proxy resolution from the output view sampling, the algorithm provides an adjustable trade-off in terms of quality vs. speed. In particular, smooth per-pixel blending fields can be computed from relatively coarse geometric proxies. Moreover, the particle-based scene representation supports an implementation which executes all processing steps fully on the GPU, resulting in high performance rendering with unconstrained and interactive 3D user navigation.
Figure 4.2: Overview of our processing pipeline. From a set of input images $I_j$ loaded to the GPU, a pre-process computes a proxy $P_j$ for each input view by a photo-consistency optimization and subsequent regularization and noise removal. The output view synthesis then combines all proxies $P_j$ into a view-dependent depth map $D^\star$ and computes color contributions from all input views in order to render the final output image $I^\star$ in real-time.

In order to process more general types of scenes instead of characters as in the previous chapter, some of the prior assumptions with respect to the scene geometry have to be alleviated. To compensate the additional degrees of freedom we switch from an uncalibrated setting to scenarios with multiple calibrated and synchronized cameras. Accordingly, the input to this method is a set of calibrated images $I_j$ showing a static scene from different viewing positions. Our goal is then to synthesize a new output image $I^\star$ from a new vantage point which reproduces the photo-realistic appearance of the original input views.

For each output pixel in $I^\star$ two properties have to be determined: First, one has to compute which set of input images actually influences the pixel’s appearance, i.e., which input images are seeing the same 3D scene point as the output view. Secondly, the color contributions of these images to the pixel’s final color have to be estimated. Accordingly, the proposed algorithm proceeds in two main phases: The computation of proxies for the input views, and the actual synthesis of new output views (see Figure 4.2).
4.1 Conceptual Overview

4.1.1 Input View Proxies

The first phase is a pre-process that extracts a partial geometric scene proxy from each input view $I_j$. The key idea is to regard each input view camera center $c_j$ as an emitter of a structured particle cloud $P_j$ which is generated by uniformly sampling pixels in the corresponding reference image $I_j$ at a desired resolution (see Figure 4.3). The basic definition of a particle $P \in P_j$ is given by a tuple $P := (u, v, r)$. Parameter $u \in \mathbb{R}_+$ is the particle’s distance to the camera center $c_j \in \mathbb{R}^3$ of its respective input image $I_j$, $v \in \mathbb{R}^3$ is its associated viewing direction from the camera center through the image plane at a particular pixel $(x, y)$, and $r \in \mathbb{R}_+$ is the particle’s radius in object space. Then, for each particle, the distance $u$ has to be computed where its corresponding viewing ray $v$ intersects the scene surface $S$.

A common approach in stereo reconstruction for solving this problem is to sample the viewing rays at discrete positions, and to estimate the photo-consistency at each sample position by comparing color differences of projections in image-space, similar to the original voxel-coloring method [SD97], the commonly employed plane-sweep stereo [YPYW04], or related labeling and segmentation techniques based on graph cuts [BVZ01, VTC05]. However, these standard approaches have two major restrictions. Due to the discrete representation, higher geometric resolutions come at the expense of correspondingly increased processing times, which is true for most stereo methods using discrete depth labels. Moreover, although approaches using isotropic image space footprints for the consistency estimation work well for continuous surfaces without a too strong perspective distortion between the comparison images, they lead to wrong estimates at oblique surface parts or at object silhouettes since the consistency computation is then performed over fore- and background pixels simultaneously.

Our proxy generation addresses these problems in several respects. We propose a new photo-consistency estimation for general volumetric scene elements, which is ro-
bust with respect to perspective distortion by integrating consistency values over a 3-dimensional domain rather than screen- or estimated object-aligned 2-dimensional patches (see Section 4.2.1). Moreover, the consistency estimation is based on silhouette aware, anisotropic particle shapes which have their image footprint properly aligned to scene discontinuities in order to handle object silhouettes and occlusions in the input images (see Section 4.2.2). Finally, a particle-based view-space parameterization enables the use of efficient continuous optimization and regularization techniques rather than a simple discrete sampling of the view-space, and extends standard passive stereo to robust reconstruction of smooth proxies from arbitrary numbers and configurations of cameras (see Section 4.3).

Each computed particle cloud $P_j$ then represents a geometric proxy of the scene as seen from its reference input image $I_j$ and basically encodes which view captures which part of the scene.

### 4.1.2 Output View Synthesis

The second phase proceeds with the actual interactive view synthesis. From the set of proxies it can be determined whether an input image $I_j$ potentially contributes to the desired output image $I^*$ by a reprojection of the corresponding particle cloud $P_j$ into this new view. However, one has to consider the fact that particles from different proxies could project onto the same output pixel, but actually correspond to entirely different scene points. Therefore, this second phase consists of the following steps.

First, all partial scene proxies $P_j$ generated in the first phase are merged into an intermediate, view-dependent scene proxy or depth map $D^*$ which represents the scene geometry as seen from the desired output view $I^*$ (see Section 4.4.1). All particles of a particular input view proxy $P_j$ which lie in the spatial vicinity of $D^*$ then correspond to a scene point visible in the input image $I_j$ that is also visible in the output view and contributes to the output color information. Thus, we compute the correct output colors by projecting the output view pixels over the proxy $D^*$ into the corresponding input image $I_j$. The output view $I^*$ is then generated by a weighted accumulation of the contributions of all relevant input images. This approach, which is described in Section 4.4.2, effectively corresponds to a pixel-accurate blending field.
4.2 Particle Photo-Consistency

The most fundamental processing step for building a scene proxy is to estimate whether a scene element located at a particular position in 3-space coincides with the unknown surface as seen in a set of \( n \) input images \( I_j \). The basic idea is that only those regions containing the surface have a consistent appearance in the images \( I_j \), while other regions project to incompatible image parts (see also Section 2.1.2).

In the previous chapter we already presented a consistency measure used for mesh-based shape tracking (Equation 3.4). In this section we discuss a more general class of consistency measures commonly used in 3D object reconstruction and view synthesis. The scene elements in this chapter are represented by particles \( P \) as defined in Section 4.1.1. The difference in appearance, the so-called photo-consistency \( \phi(P) \) of a particle \( P \), can be measured by comparing image patches \( \Omega_j \subset \mathbb{R}^2 \) resulting from projecting \( P \) into images \( I_j \) (see Figure 4.4). Here we assume that \( P \) is visible in all comparison images. The issue of occlusions will be handled in subsequent sections.

The original photo-consistency estimation proposed in [SD97, KS00b] assumes perfectly Lambertian and well textured surfaces under constant illumination conditions. Under these assumptions \( \phi(P) \) is computed by projecting \( P \) into each image \( I_j \) and measuring the resulting color variation. Surface points are expected to have a low color variation, since they project to corresponding image regions \( \Omega_j \), while points not lying on the surface project to inconsistent colors (see Section 2.1.2). More formally, one first computes the average color \( c_j \) of all pixels \( p_{kj} \in \Omega_j \) and then applies a transfer function \( f \) to the variance of the different colors \( c_j \):

\[
c_j = \frac{1}{|\Omega_j|} \sum_k I_j(p_{kj}), \quad \tau = \frac{1}{n} \sum_j c_j, \quad \phi(P) = f \left( \frac{1}{n} \sum_j (c_j - \tau)^2 \right). \tag{4.1}
\]

This variance-based photo-consistency measure supports efficient computation and unconstrained camera setups. However, it is quite sensitive in practice to non-Lambertian surfaces, weak texture, and varying illumination. Despite these restrictions this method is still widely used because of its computational efficiency and the often acceptable quality, e.g., for time-critical new view synthesis applications [SSH03, LMS04, SP05]. Since then the original approach has been subject to several improvements for handling transparency and more complex illumination effects [BV99, BDC01, YPW03].
Figure 4.4: Consistency estimation. (a) Sampling of an object aligned planar patch. (b) Wrong sampling due to misaligned patches. (c) Rectangular image-space comparison window. (d) Mismatch between simple comparison window and projectively warped window (dashed).

In more recent work with a focus on accurate image-based model reconstruction, photo-consistency is commonly evaluated by comparing the intensity functions of image patches $\Omega_j$ by using more sophisticated measures such as sum-of-squared-differences (SSD, Equation 3.4) or normalized cross-correlation (NCC) \cite{GW02, HS04, VTC05}. Suppose we approximate the unknown surface $S$ intersecting a volumetric region by a planar surface patch as illustrated in Figure 4.4 (a). The respective intensity functions can be compared using NCC by placing $m$ object space samples $p^0_j$ to $p^{m-1}_j \in \mathbb{R}^3$ on this patch, and evaluating their respective image space projections $p^k_j$ and $p^k_l \in \mathbb{R}^2$, $0 \leq k < m$ in two images $I_j$ and $I_l$. Since $S$ is unknown one generally computes an approximate solution by doing a pixel-wise comparison of simple, rectangular windows $\Omega_j$ instead (see Figure 4.4 (c)):

$$c_j = \left( \ldots, I_j(p^k_j) - c_j, \ldots \right)^T, \quad \hat{c}_j = \frac{c_j}{\|c_j\|}, \quad \phi(P) = f (\hat{c}_j^T \cdot \hat{c}_l), \quad (4.2)$$

with $p^k_j \in \Omega_j$, $m = |\Omega_j|$ and the average color $c_j$ and $f$ as defined in Equation 4.1.

This method strongly reduces potential color ambiguities and accounts for changes in illumination due to the involved normalization step. But despite these advantages there remain several open issues with this approach.

Since the NCC is computed for pairs of image patches only, one has to combine results for more than two images to compute the actual photo-consistency $\phi$ in a multi-view stereo setting, e.g., by averaging the NCC for all image pairs \cite{VTC05} or by using a
4.2 Particle Photo-Consistency

single reference image [HS04]. But more importantly, one of the main problems of the above approach is the fact that pixels $p^k_j$ and $p^k_l$ in images $I_j$ and $I_l$ might not correspond to the same surface sample in object space (see Figure 4.4 (d)). Hence, image-aligned patches provide acceptable results only for medium baseline, epipolar-aligned images, while setups with arbitrary camera configurations are difficult to handle. As also mentioned in [HS04], the more sophisticated planar model-aligned patches provide valid results only if the approximation is not too far away from the true object surface (e.g., Figure 4.4 (b)). In both cases projective warping can introduce a considerable matching error.

Another important aspect besides the quality of a photo-consistency measure is its efficiency. Computation times of up to several hours are common even in recent NCC-based work due to the high computational complexity [Mid08a]. Although this could be considered acceptable for high quality model reconstruction, it is less desirable for interactive free viewpoint applications. Partially hardware accelerated implementations for consistency estimation and related techniques have been presented in a number of works [PD98, CMS99, ESC99, SBS02, YP03, LMS04]. However, those techniques either have conceptual limitations in their general applicability to arbitrary input, or they have restrictions concerning the accuracy of the results.

In [HK06b] we proposed an efficient GPU accelerated consistency measure based on a new, color normalized volumetric supersampling approach, for which it was shown that the color integration of samples projected from a 3D domain rather than using object- or image-aligned 2D patches is less biased in those cases where the surface orientation is not known in advance.

4.2.1 Volumetric Supersampling

To overcome the aforementioned problems we propose to create consistent object-space samples $p^k \in \mathbb{R}^3$ such that the matching error does not depend on the quality of the current surface approximation or view alignment but only on the size of the tested volumetric region, i.e., the size of a particle $P$ or, equivalently, the resolution of the voxel grid in volumetric reconstruction techniques [SD97, KS98b].

Photo-consistency can be considered as an unsigned pseudo-distance function $\phi : \mathbb{R}^3 \rightarrow \mathbb{R}_+$ defined in continuous 3-space where the scene to be reconstructed is embedded.
Interactive Free Viewpoint Rendering

(see Section 2.2.3). This function is expected to vanish for points lying exactly on the surface $S$, to have small values in the vicinity of $S$, and to have larger positive values (which, however, do not necessarily increase for larger distances) everywhere else. If we do not have any reliable information about the exact location and orientation of $S$ within a given volumetric region, the best consistency indicator that we can check is to simply integrate the function $\phi$ over the entire interior of the region. The value of this integral is expected to be relatively small in those parts of the scene volume that are intersected by the surface. Obviously the integration of $\phi$ has to be done numerically, i.e., by supersampling the considered particle at sub-particle resolution (see Figure 4.5).

In our particle-based setting we therefore uniformly distribute $m$ equally weighted samples $\{p^k : \|p^k - p\|^2 \leq r^2\}$ within the volume enclosed by a particle $P$ with radius $r$ at position $p$ and compute the colors for each of these samples separately by projecting them into the respective input images. For the sample distribution we use a variant of the Sobol quasi-random sequence generator [PFTV92] because of its properties in terms of incremental sample distribution. This approach effectively eliminates the matching problem between the different images and samples even for completely unconstrained camera positions.

To preserve the illumination invariance of the NCC-based approach we apply a similar normalization step to the color vectors $c_j \rightarrow \hat{c}_j$ as in Equation 4.2. The complete photo-consistency $\phi(P)$ of a particle is computed by summing up the variation of each sample’s color $\phi(p^k)$ over all images $I_j$:

$$
\phi(P) = \sum_k \phi(p^k), \text{ with }
$$

$$
\phi(p^k) = \text{VAR}_j(\omega_j \hat{c}_j^k) = \sum_j \omega_j (\hat{c}_j^k)^2 - \left( \sum_j \omega_j \hat{c}_j^k \right)^2.
$$

The contribution of each image $I_j$ can be weighted using a per image weight $\omega_j$ (with $\sum_j \omega_j = 1$) to prioritize images taken from near-by viewing positions in order to respect effects such as blurring at oblique viewing angles. To consider the full three channel color

![Figure 4.5: Particle super-sampling.](image)
space instead of just one intensity channel, one simply accumulates the resulting variance for each channel. Since $\phi$ measures the color variation, values close to 0 correspond to a high photo-consistency while larger values correspond to an inconsistent appearance.

This general approach to computing the photo-consistency by volumetric supersampling and projection is robust to different camera and surface configurations and can be implemented efficiently on the GPU (see Section 4.5). Moreover, Chapter 5 will show how this approach can be utilized for higher order surface approximations in high quality surface reconstruction as well.

4.2.2 Silhouette Aware Sampling

As motivated above, simple isotropic image footprints may result in wrong photo-consistency estimates at object silhouettes since fore- and background pixels are evaluated together (see Figure 4.6 (a)). This problem has been addressed in previous work by using modified or shiftable 2D correlation windows for improving object boundaries and to handle occlusions [KSC01, HIG02]. A more sophisticated solution are adaptive weights to control the influence of single pixels within correlation windows [YK06]. However, due to the use of windows defined in image space, these techniques have difficulties to handle arbitrary camera configurations and the resulting scale changes or perspective distortions between images.

In our more general multi-view stereo setting supporting arbitrary volumetric elements, such modified 2D correlation windows are not suitable. Instead, these issues have to be addressed by modifying the shape of these elements, i.e., the 3D particles $P$. Simple anisotropic shapes such as ellipsoids alleviate, but do not resolve this problem (see Figure 4.6 (b)). Moreover, one has to compensate the compression along one axis by stretching the footprint along the silhouette in order to keep the footprint area large enough.

Therefore, in order to allow for a sharp segmentation, we modify the above formulation using spherical 3D particles by intersecting them with a silhouette aligned feature plane.
This plane has a normal orthogonal to the associated viewing ray of a particle such that the intersected particle projects to a hemi-circle in image space as shown in Figure 4.6 (c). This particle shape has several important advantages: Firstly, it can be aligned to object silhouettes such that either only foreground or only background pixels are taken into account during the consistency estimation. Secondly, the sampling on the object itself remains rather isotropic, without incorporating additional color samples from oblique and therefore less reliable surface areas.

A problem is that the actual object silhouettes are unknown and that they cannot be reliably estimated from a single image alone by using, e.g., color gradients. With multiple input images and our photo-consistency estimation described in the previous section, however, we can generally expect the color variance of a particle to be higher for wrong silhouette alignment due to the mixture of fore- and background pixels, so that a correct feature orientation can be detected by measuring $\phi$.

Hence, we add an additional parameter $d$ to the tuple defining a particle $P = (u, v, d, r)$, where $d \in \mathbb{R}^3$ is the feature plane orientation orthogonal to the particle’s viewing ray $v$. The set of samples for the photo-consistency estimation of $P$ is then defined as $\{p^k : \|p^k - p\|^2 \leq r^2 \land d^T(p^k - c_j) \leq d^T c_j\}$, so that only samples are generated which lie in one side of the sphere intersected by the plane $d^T(p^k - c_j) = 0$, where $c_j$ is the camera center of the image $I_j$ emitting $P$. The particle orientation is then included into the overall optimization process described in Section 4.3.2.

### 4.3 Proxy Generation

Given the photo-consistency measure described in the previous section, the problem of computing a scene proxy can now be solved by finding the optimal distance $u$ of a particle $P$ on its respective viewing ray, which minimizes $\phi(P)$.

Figure 4.7 shows exemplary plots of $\phi$ for a viewing ray through a close (red) and distant (blue) scene point (a). The dotted lines are the corresponding epipolar lines in the comparison image, and the thickened line segment represents the sampled interval on the respective viewing ray $v$. The graphs show the values of $\phi$ for sample positions $u$ along $v$ for three view-space parameterizations. The scales of the $u$ axes differ due to the different parameterizations, but effectively represent the same interval on $v$. 

68
4.3 Proxy Generation

Figure 4.7: Consistency plots of a near and a far scene points using three different view space warping approaches.

Uniformly distributed sample positions \( u \) in view-space result in a highly non-uniform sampling in image space due to the perspective projection. Hence, minima of distant scene points with \( u \rightarrow \infty \) appear significantly stretched (see Figure 4.7 (b)). Moreover, 3D particles with a constant radius have a footprint of decreasing size in image space for increasing distance, so that their overall color variation also decreases. These two facts lead to a distortion of \( \phi \) without a proper consideration of the image sampling rate and render consistency optimization routines based on uniform discrete sampling unstable, because minima of \( \phi \) are more difficult to locate.

Most standard passive stereo methods have to address these problems by using rectified stereo image pairs and comparison windows at discrete pixel disparities in image space. Another solution commonly employed in related work (e.g., [YPYW04]) are projectively corrected parameterizations of the view-space. However, as shown in Figure 4.7 (c), this leads to exactly the opposite effect compared to (b): close minima are overly stretched while distant minima are compressed considerably. These significantly
changing widths of the desired minima of $\phi$ and the correspondingly inhomogeneous sampling properties again render the efficient optimization of the particle positions $u$ unnecessarily difficult. The following section introduces a new particle-based parameterization of the view-space which addresses the above problems and allows for efficient continuous optimization and filtering techniques.

### 4.3.1 View-Space Parameterization

So far each particle had an associated radius $r$ which, however, was not specified in detail yet. Here we present a derivation of the particle radii which results in a natural parameterization of the 3D space and which relates the particle sizes to the scene approximation quality.

The resolution of an image $I_j$ is directly related to the precision of a 3D scene proxy that can be reconstructed from that image. Intuitively, each pixel or image space window spans a viewing cone with the corresponding camera center $c_j$ (see Figure 4.8) which integrates visual information over a part of the scene. The diameter of that cone determines the depth precision or detail that can be achieved, since an intersection of this cone with the scene geometry covers smaller scene regions close to the camera, and larger regions with increasing distance.

Thus, for a correct integration of the image resolution into our problem formulation, the particle primitives have to be associated with viewing cones rather than viewing directions. This implies that a particle at a distance $u$ has to have a radius $r$ corresponding to the radius of the associated viewing cone at that distance. Consequently, it is the number of pixels covered by a particle’s projected footprint that stays constant rather than its 3D radius. From these observations we can derive a parameterization for the particle parameters $u$ and $r$ based on the footprint radius $r_s$ of its viewing cone.
4.3 Proxy Generation

The radius \( r \) is computed from the screen-space radius \( r_s \), the camera’s focal distance \( f \), and the distance \( u \) of a particle by

\[
r = \frac{r_s}{f}.
\]  

(4.4)

Defining ratio \( \alpha := \frac{r_s}{f} \) (i.e., \( r = \alpha u \)), two “neighboring” particles at \( u_0 \) and \( u_1 \) under consideration of their size change within the viewing cone are related by (see Figure 4.8)

\[
u_1 - \alpha u_1 = u_0 + \alpha u_0 \iff u_1 = \frac{1 + \alpha}{1 - \alpha} u_0.
\]  

(4.5)

Recursive application of this relation yields an exponential function for \( u \). Writing \( \lambda = \frac{1 + \alpha}{1 - \alpha} \), and given a fixed, but arbitrary position \( u_0 \) of the first particle \( P_0 \), the position of any other particle \( P_i \) is given as

\[u(i) = \lambda^i u_0.\]  

(4.6)

Instead of considering only particles \( P_i \) at discrete positions \( i \), this equation obviously holds for particles \( P_k \) at continuous parameter space positions \( k \in \mathbb{R} \) as well. The particle radius at \( u(k) \) is \( r(k) = \alpha u(k) = \alpha \lambda^k u_0 \). Inversely, given a particle position \( u \), the corresponding parameter value is computed as

\[k(u) = \log_\lambda \frac{u}{u_0}.
\]  

(4.7)

To simplify notation in the following, we additionally introduce a distance norm based on this log-space parameterization which measures distances in terms of particles:

\[
\|P(u, \psi) - P(u_j, \psi)\|_P = |k(u_i) - k(u_j)|.
\]  

(4.8)

This parameterization yields several significant advantages: The only relevant free parameters in order to compute reasonable sampling step sizes along viewing rays are the screenspace radius \( r_s \) of the corresponding viewing cone, and a reference position \( u_0 \). Using Equation 4.6, a uniform sampling with respect to \( k \) results in a non-uniform sampling along the viewing ray, which properly considers the necessary particle size change. The resulting sampling is not necessarily uniform in all comparison images during the photo-consistency estimation. However, a single parameterization resulting in uniform sampling with more than one comparison image is obviously not possible. Hence,
in the context of our problem formulation and design goals supporting an arbitrary number of input views, this parameterization seems to be the best possible choice.

The positive effect of this log-space parameterization using a uniform sampling of parameter $k$ is shown in Figure 4.7 (d): Minima for close and distant points have identical shapes, independent of the domain boundaries of $u$. This significantly simplifies corresponding optimization routines for finding these minima and allows for uniform step sizes. Moreover, given two extremal interval boundaries $u_0$ and $u_e$ along a ray which are known to contain the scene’s region of interest, the number of necessary sample positions within this interval can be automatically derived from Equation 4.7. Finally, as will become clearer in the following sections, the distance norm in Equation 4.8 allows for a straightforward definition of outlier particles, noise, and corresponding filter operators. Due to this parameterization, the particle radius $r$ depends only on $r_s$ and can be removed from the particle definition.

### 4.3.2 Optimization

In order to produce high quality rendering results it is inevitable to compute discontinuity preserving, but otherwise smoothly varying depth estimates for the particles. Hence, one generally needs very robust optimization techniques and additional smoothness terms in order to solve this problem. In related work one finds a variety of approaches which range from exhaustive but effectively discrete sampling up to complex non-linear energy minimization procedures. Unfortunately, most of these approaches are computationally highly expensive and one of the main sources for the considerable processing times in image-based rendering. Moreover, the required artificial smoothness energies often introduce an undesirable bias.

Instead of a combined optimization of $\phi$ and a smoothing regularization of the particle positions, we split these two tasks into subsequent processing steps: First, an optimal position and orientation is computed for each particle independently. Then we apply discontinuity preserving filter operations in order to compute the final proxy. Formally, the optimization problem can be described by the following expression

$$\arg \min_{u, \gamma} \phi(P(u, v, d(\gamma))),$$

(4.9)

i.e., for each particle we have to find the optimal viewing ray position $u$ and feature direction $\gamma$ by rotating $d$ around the rotation axis defined by the viewing ray $v$. 

72
4.3 Proxy Generation

Figure 4.9: Illustration of the discontinuity alignment at the claw and pillar of the Bahkauv statue (a). (b) shows silhouette aligned particles (solid) and arbitrarily oriented non-silhouette particles (dashed). Close-up renderings of a point cloud proxy generated from (a) are shown in (c) and (d). With isotropic particle shapes (c) a band of background particles is wrongly associated with the foreground around the silhouette, resulting in significant rendering artefacts. With silhouette aware particle shapes (d) the outlines are reproduced much more faithfully. The proxy quality and continuity of the reconstruction is shown for the floor and the pillar in a side view (e) and top view (f). These types of curved and oblique surfaces are generally difficult to reconstruct with methods based on discrete depth sampling.

The estimation is started with a uniform sampling of particles $P_i$ at discrete sample positions in the previously defined log-space and a set of uniformly sampled feature directions $\gamma_j \in \{0, \frac{\pi}{4}, \frac{\pi}{2}, \ldots\}$ by a stepwise rotation of the particle orientations during the optimization, i.e., we compute $\forall u \forall \gamma_j \phi(P(u, v, d(\gamma_j)))$. The only user-defined input parameters to this procedure are two search interval boundaries $u_0$ and $u_e$, and the desired footprint size $r_s$. The required number of sample positions on $v$ is then given by Equation 4.7. After this global sampling at discrete particle positions $P_i$, a subsequent continuous optimization is performed by using the current optimum as a starting position for incremental interval refinement (golden section search, [PFTV92]). The initial interval for this search is simply defined as $[P_{i-2}, P_{i+2}]$, corresponding to an offset of two particles in each direction along the viewing ray with respect to $P_i$. During this phase the previously computed feature direction remains fixed for each particle.

Figure 4.9 (a) and (b) illustrate the automatic alignment of the particle shapes to discontinuities at object silhouettes. In non-feature regions, the particle directions have
4 Interactive Free Viewpoint Rendering

no influence on the overall consistency estimation. The resulting proxy and silhouette quality from different vantage points with isotropic and feature aware particles is shown in (c) and (d), respectively. Similar to the particle positions, we explicitly do not want to enforce any artificial smoothness on the direction field. Moreover, a particular advantage of this optimization is the fact that it does not depend on a pre-analysis of silhouettes based, e.g., on image gradients, which is generally an error-prone process.

The proposed approach effectively corresponds to a feature aware, continuous stereo optimization, resulting in faithful reconstructions of discontinuities as well as of curved and oblique surfaces as shown in (e) and (f). All relevant parameters such as the convergence threshold for the golden section search can be provided intuitively and independent from the underlying scene using the distance norm $\| \cdot \|_P$ defined in Equation 4.8. For example, in our implementation this convergence threshold is set to 1% of the particle radius.

4.3.3 Regularization and Filtering

During the above optimization, an image-based regularization is achieved by using a particle footprint of several pixels. Moreover, the theoretical sensitivity of the photo-consistency measure to textureless regions can be alleviated by dynamically increasing the particle size, until a sufficient color variance in the reference image $I_j$ is achieved [HK07]. Our experiments showed, however, that a small footprint of $r_s = 5$ is generally sufficient. Nevertheless, due to the feature aware sampling and proper separation of fore- and background regions, the footprint size can be easily increased without sacrificing sharp object silhouettes in particular for the more dominant silhouettes of foreground objects.

For those cases in which background regions visible in reference image $I_j$ are occluded in the comparison images, a proper depth cannot be computed. This results in a small number of outlier particles. Moreover, the purely image-based regularization exhibits a certain amount of remaining noise as well (see Figure 4.10). As motivated in the previous section we do not want to bias the initial particle optimization. Instead, we propose a simple set of three subsequent spatial filters which effectively handle outliers and measurement noise as a post-optimization process. A nice property of these filters is that they can again be formulated intuitively in terms of $\| \cdot \|_P$. 

74
4.3 Proxy Generation

Figure 4.10: Regularization and outlier removal. (a) A reference image $I_j$ from the Middlebury Temple data set [Mid08a]. (b) A cut out of the corresponding particle cloud $P_j$ directly after the photo-consistency estimation without any explicit regularization and (c) after the regularization and filtering step. For an improved visualization we converted $P_j$ into a triangle mesh by creating a vertex from every particle and connecting neighboring particles.

Assuming that the scene consists of piecewise continuous surfaces, we first reposition isolated outlier particles $P \in P_j$ emerging from local minima of $\phi$, which lie in otherwise sufficiently smooth surface regions. Due to the structured nature of the particle cloud resulting from a regular sampling of the input image $I_j$, this can be achieved by a uniform filter kernel which counts the number of neighbors of $P$ within a certain distance $\tau$:

$$s(P) = \# \{P': P' \text{ is in } 8 \text{-neighborhood of } P \land \|P' - P\|_P < \tau \}. \quad (4.10)$$

If this smoothness measure has a value $s(P) \leq 3$, we move $P$ into the center of gravity of its neighbors. This filter $F_S$ has two major effects: Particles with a stable proximity remain fixed, while isolated outlier particles are moved towards the surface defined by their neighbors. The remaining outlier particles without a sufficiently smooth support are removed by a filter $F_O$ similar to $F_S$. We again estimate the support of a particle using Equation 4.10, and reject all particles with $s(P) \leq 3$ instead of repositioning them. Parameter $\tau$ depends on the particle density of $P$ (the particle sampling of $I_j$), but proved to work robustly for all our experiments with $\tau = 0.1$.

Finally the remaining measurement noise is removed by a last weighted smoothing operator $F_W$. We compute a weighted average of neighboring particles by taking their distance into account, so that depth discontinuities are preserved. The weight of each
neighbor particle $P'$ is simply computed as $w(P') = 1.0 / \max(\|P' - P\|_F, 0.1)$. Since the filters $F_S$ and $F_O$ already converge to a stable solution after a few iterations, the overall proxy quality is not sensitive to changes of the involved parameters. An example for a particle cloud $P_j$ before and after optimization ($P_j \leftarrow F_W \circ F_O \circ F_S(P_j)$) is shown in Figure 4.10. These steps finalize the phase for generating the input view proxies.

4.4 View Synthesis

For computing a new view of the scene, one could in principle follow two obvious approaches. The first would be a forward splatting approach similar to layered depth images [SGwHS98]. However, this approach has the significant disadvantage that the output view sampling and quality would depend solely on the geometric complexity of the scene proxies. Our aim is to decouple the geometric proxy resolutions from the output view sampling, so that high quality views can be synthesized even from proxies with low geometric complexity.

A second possibility would be to utilize the same optimization as for the input views, but with the particles defined with respect to the output view similar to [FWZ05]. However, this approach has restrictions as well. The optimization procedure for input proxies $P_j$ has one fixed color constraint for each particle in its originating view $I_j$. This is no longer true for the unknown output view. Although this may seem a subtle problem, the photo-consistency estimation becomes significantly more ambiguous and ill-posed in practice. The second problem is that it cannot be determined a priori, for which viewing ray positions a particle is actually visible or occluded. This results in significant rendering artifacts in particular for output views distant to any input view. Since our input proxies are generated from nearby input views, the occlusion problem can be effectively handled using the silhouette aware particles and our filter scheme.

For computing a new view $I'$ of the scene, we instead propose the following approach, which computes pixel-accurate color contributions by backward warping and which achieves the desired decoupling of the proxy resolutions from the output view resolution.
4.4 View Synthesis

4.4.1 View-Dependent Output Proxy

The first step consists of computing a depth proxy \( D^\star \) for the output view \( I^\star \) by a view-dependent merging of the generated input proxies. We first splat each proxy \( P_j \) into the output view and store the resulting depth maps \( D_j \). These depth maps then have to be merged into a single depth map \( D^\star \). A corresponding operator has to robustly remove remaining outliers in the depth maps \( D_j \) and compute a reliable depth value for each output pixel. The benefit of merging multiple depth maps for increased accuracy and robust outlier removal has been discussed in the context of 3D object reconstruction in [MAW+07]. Our problem setting, however, requires outlier resistant filtering with visibility validation as well as support for an efficient real-time implementation.

Simple filtering by computing, e.g., the median value of all depth maps \( D_j \) is not sufficient (see Figure 4.11): Assume that each depicted particle belongs to a different proxy \( P_j \), and that (a) is an outlier particle. The particles located at positions (b) and (c) represent valid scene points. Computing the median would wrongly choose a depth value from the occluded set of particles (b, right), because this scene part has been observed by more cameras. The correct scene point visible in the output view, however, should be located at position (c, green). This visibility problem can be properly resolved by the following 2-phase filter.

An iterative, GPU-friendly median filter can be implemented by partitioning the search space into two sections, and counting the number of elements within each section [VKG03]. This procedure is iterated by subdividing the interval containing the median (Figure 4.11, right bars). Our modified filter has the additional prioritized rule to subdivide the section closer to the camera, if at least \( n \) cameras "vote" for this part of the scene, i.e., there are at least \( n \) particles (Figure 4.11, left bars). If, after enough interval subdivisions, the number of particles within the investigated intervals is less than \( n \), the filter proceeds and computes the median depth value of the remaining particles. This approach robustly eliminates remaining outlier particles and correctly resolves occlusions.

Figure 4.11: Filtering of depths. The bars visualize the different interval refinement strategies for merging depth maps.
Figure 4.12: Example for the view-dependent proxy generation. (a) Two reference images $I_0$ and $I_1$. (b) The resulting particle clouds $P_0$ and $P_1$ splatted into the output view. (c) The corresponding depth maps $D_0$ and $D_1$. The merged depth map $D^*$ is shown in (d, left) and the output image $I^*$ after the color reprojection in (d, right).

between different scene parts projecting to the same pixel by taking the closest depth value that is reliably seen by enough cameras. Moreover, it results in a continuous proxy $D^*$ even for rather sparse input proxies $D_j$. The parameter $n$ basically depends on the noise-level (i.e., the expected number of outliers) and the number of input cameras. However, in all our experiments we consistently used $n = 3$. The complete view-dependent proxy generation is illustrated in Figure 4.12.

4.4.2 Color Estimation

Given $D^*$ the next task is to compute the “blending field”, i.e., to accumulate the actual color contributions from the input images to each output pixel. This can again be evaluated efficiently by a forward splatting approach similar to the previous section.

Each particle proxy $P_j$ is splatted into the output view. We then test for each covered output pixel $(x, y)$, whether the depth value $d(x, y)$ of the covering splat lies within a narrow band around the computed depth map $D^*(x, y)$ of the output view. Each “activated” pixel $(x, y)$ covered by such a splat then has to receive some color contribution.
4.5 Efficient GPU-based Implementation

A nice feature of the described procedure is that it can be implemented completely on the GPU using OpenGL [SA06], GLSL shaders, and multiple render targets based on floating point framebuffer objects and textures. We use a combination of several vertex, geometry, and fragment shaders. Most steps described here follow the general approach in GPGPU processing: The underlying idea when transferring an arbitrary algorithm to the GPU is to exploit the possibility to execute a custom program for each generated vertex and fragment independently and in parallel instead of using the standard 3D rendering pipeline. Because of the floating point support of current GPUs even quite

Figure 4.13: Output pixel color estimation by reprojection. (a) shows the activation of put view pixels. In (b) the output colors for the activated pixels are gathered from the input views by reprojection.
complex input data can be processed by encoding it in the color channels of one or more textures. By simply drawing a screen-sized quad one generates width $\times$ height fragments on which a custom algorithm is executed. This means one effectively runs this algorithm on the texture encoded input data width $\times$ height-times in one single rendering pass. The output data of the algorithm can then directly be used for further processing or accessed by reading it from the framebuffer. In our rendering pipeline the CPU is used only for high level control of the algorithm.

The first important data structure is a single set of width $\times$ height vertices which corresponds to the proxy particle clouds $P_j$. For efficiency reasons, these vertices are stored in a vertex buffer object, and hence have to be transferred to the GPU only once at the beginning of the process. The density of $P_j$ can be chosen independently of the actual input image resolution, which is an important aspect for the scalability of our method. All further particle data, i.e., $u$, $v$, and $d$, is stored in textures of size width $\times$ height, such that one has a one-to-one correspondence between particles at position $(x, y)$ in $I_j$ and their texture data. These textures are entirely processed on the GPU. For instance, the initialization of a texture storing the viewing rays $v$ can be performed by loading the corresponding camera projection matrix of $I_j$ to the GPU. A fragment program then computes and writes $v$ to this texture. When rendering and processing the particles $P_j$ they retrieve their data from these textures. The algorithms described in the following are implemented as multi-pass rendering approaches with alternating rendering targets.
Consistency Estimation  Compared to window-based NCC, our photo-consistency estimation introduces an additional overhead since, for each of the samples $p^k$ of a particle $P \in P_j$, the projections into a set of comparison images $I_i$ have to be computed (see Section 4.2.1). However, this overhead can be compensated by exploiting the recently introduced geometry shaders which support the generation of the samples $p^k$ directly on the GPU. Hence, we render the corresponding particle list $P_j$ and spawn for each $P \in P_j$ a set of $m$ 3D samples $p^k$ using a geometry shader. Each spawned sample then writes its 3D position serialized into an output texture $T_p$, so that a single particle is represented by a sequence of $m$ texels (see Figure 4.14). After the serialization, a fragment $f$ is generated for every $p^k$, and the comparison images $I_i$ and projection matrices are transferred to the GPU. All intermediate results for computing $\phi$ as in Equation 4.3 are then loaded from and stored to textures. The main loop of the consistency estimation consists of the following steps:

1. Projection Pass:
   
   foreach fragment $f$ and image $I_i$ do
   
   Compute sample color $c^k_i := I_i(p^k_i)$ by projecting $p^k$ into image $I_i$.
   
   Compute camera weight $\omega_i$.
   
   Store color and weight in a texture $T_c(f) := (c^k_i, \omega_i)$.
   
2. Normalization pass:
   
   foreach fragment $f$ and image $I_i$ do
   
   Loop over all samples $c^l_i, 0 \leq l < m$ and normalize $c_i \rightarrow \hat{c}_i$.
   
   Store normalized color and weight $T_{nc}(f) := (\hat{c}^k_i, \omega_i)$.
   
3. Accumulation pass:
   
   foreach fragment $f$ and image $I_i$ do
   
   Get $(\hat{c}^k_i, \omega_i) := T_{nc}(f)$.
   
   Add $\omega_i (\hat{c}^k_i)^2, \omega_i \hat{c}^k_i$, and $\omega_i$ to the accumulation buffer $T_a$.
   
4. Variance computation:
   
   foreach fragment $f$ do
   
   Loop over the color and weight data stored in $T_a$ and compute variance
   
   $\phi(p^k) = \sum_\omega_i (\hat{c}^k_i)^2 - \left(\sum_\omega \omega_i \hat{c}^k_i\right)^2$ according to Equation 4.3.
   
   Compute final sum $\phi(P) = \sum_k \phi(p^k)$ and store it in $T_\phi$.

Algorithm 4.1: GPU-based photo-consistency

Since we have three color channels per image we accumulate the $3 + 3 + 1$ values computed in step 3 in two output buffers using multiple render targets.
**Optimization** The central data structures for the GPU-based uniform and golden section search are two textures, one for storing the particle position during the optimization and one for storing its consistency values and the feature orientation. For instance, for the implementation of the section search, one has to store a 4-tuple \((u_0, u_l, u_r, u_e)\) for each particle: 2 interval boundaries and 2 sample positions on the viewing ray at which we evaluate the consistency. Hence, we store this tuple in the four channels of a floating point texture. Secondly, we have to store the estimated photo-consistency values at positions \(u_l\) and \(u_r\), and an update flag \(f\) indicating for which of these two positions the consistency has to be recomputed in the next iteration. The current feature orientation \(\gamma\) (see Equation 4.9) is stored in the remaining texture channel, resulting in a tuple \((\phi_l, \phi_r, f, \gamma)\). The GLSL-shader then uses these two textures as input, evaluates \(\phi_l\) and \(\phi_r\) as described above, and computes new sample and boundary values for the next iteration based on Equation 4.6 and Equation 4.7.

The convergence of the optimization routine can be checked by rendering all \(P \in \mathcal{P}_j\) into a single output pixel. Rasterized fragments corresponding to converged particles are discarded, while non-converged particles create some arbitrary output value. This single output pixel for controlling the optimization process is actually the only data transferred back from the GPU to the CPU in our entire pipeline. Finally, the implementation of the filter operators is straightforward by again rendering a single quad into an \(\text{width} \times \text{height}\) output buffer, with correspondingly activated fragment programs. Each fragment first estimates the stability \(s(P)\) using a series of texture lookups and then performs the corresponding filter update.

**View Synthesis** The splatting approach described in Section 4.4.1 is straightforward, since it is basically needed only for the generation of depth maps \(D_j\) and for "activating" fragments in the output view during the color estimation. Hence we employ a standard splatting approach [BHZK05]. The depth maps \(D_j\) are generated by rendering each particle cloud \(P_j\) into the output view, and storing the resulting depth values in a floating point texture. The filter for merging the depth maps \(D_j\) into \(D^*\) is implemented on the GPU using an extension of the bi-sectioning approach described in [VKG03]. Computing the color contribution of an image \(I_j\) to an output pixel again uses standard splatting. Each generated fragment first checks, whether its depth value is in the proximity of \(D^*\). If yes, it is projected over \(D^*\) into \(I_j\) and the resulting weighted color value is added to a color accumulation buffer. A final normalization pass produces the final output image.
4.6 Results and Discussion

We tested our method with different data sets and scene types. The experiments were performed on a Nvidia GeForce 8800 graphics card. Even though accurate 3D reconstruction of the scene geometry is not our primary target here, we evaluated the numerical accuracy of the proxy generation with the Middlebury multi-view stereo data sets [SCD’06, Mid08a]. For both models, the Temple and the Dino, we computed 11 particle proxies from a subset of cameras of the “full” data sets, and combined the resulting point clouds into a single model. Figure 4.15 shows renderings and the achieved accuracy and completeness for these models. These results are quite competitive considering the fact that we used a much lower number of input views (11 proxies / reference images and 5 additional comparison images for each proxy) than other multi-view stereo algorithms for creating these models. The primary focus of our method is, however, the efficient creation of sufficiently accurate, silhouette preserving geometry proxies for new view-synthesis rather than high-quality model reconstruction.

The camera configurations for the other examples ranged from small base-line stereo with only three input images (Figures 4.1 and 4.18) to over 40 images (Figures 4.16 and 4.17). The number of proxies \( P \) ranged from three for the Semper statue to 20 proxies for the Monkey. For each corresponding reference image \( I_j \), we used between two and four additional comparison images for estimating \( \phi \). The proxy generation generally takes a few minutes, depending on the desired proxy resolution, the number

![Image](image_url)
Figure 4.16: Monkey example. (a) Two exemplary input images. (b) New output view with camera parameters similar to the input views. (c) Top view with an increased field of view which significantly differs from the input views. (d) Side view showing the faithful reproduction of the curved head. (e) View-dependent illumination effects on the fur and the nose.

of samples per particle, and the depth of the scene, since the number of sampling steps during the uniform optimization is derived from the particle size. For instance, for the Semper statue shown in Figures 4.1 and 4.18 with a high input image resolution of $3072 \times 2048$, we used $300k$ particles for each of the three proxies with 50 samples per particle, corresponding to a $7 \times 7$ window size in standard NCC based approaches. The resulting computation time was 180 seconds per proxy. For the Bahkauv statue in Figure 4.17 with 40 images at $720 \times 576$ and $100k$ particles, our method needs 35 seconds per proxy. The Monkey example in Figure 4.16 with 56 images of $654 \times 490$ pixels and $60k$ particles took 20 seconds per proxy. For comparison, our CPU reference
4.6 Results and Discussion

Figure 4.17: Bahkauv statue. (a) The first and last image of the input video sequence. (b) Renderings from new positions and with different camera parameters.

implementation needed several hours for the same data. This offline process is fully automatic without requiring any user intervention.

The actual free viewpoint rendering runs with several frames per second at resolutions of 800 × 600 or higher. The performance depends mainly on the number and density of the proxies \( P_j \) and the resolution of \( D_r \). However, even for high numbers of input views or particles as for the Monkey or Semper example, our method easily reaches 20 frames per second or higher. If required, a higher performance can be achieved by downsampling the proxy and depth map resolution. Due to the per-pixel blending this is possible without sacrificing too much visual quality.

As shown in the examples, an advantage of this method is that even output views with camera parameters which deviate significantly from all input images are able to preserve the photo-realistic appearance of the scene (see Figure 4.16 (c) and (d)). View-dependent specular highlights and other subtle illumination effects are visible in Figure 4.16 (e). However, too strong reflections or dominant specular highlights like on the large glass window in Figure 4.18 (a) are problematic for the consistency estimation based on color variances.

Regions which are occluded and not visible in at least 2 input views result in white spots in the output renderings (e.g., Figure 4.18). However, this is not a restriction of our method, but rather due to the possibility for unconstrained user navigation outside the hull spanned by the input views. In contrast, many previous techniques do not allow for extreme viewpoint changes with respect to the original input views. The input images
of the Bahkauv statue were captured with a hand-held camera in a difficult scene with many depth discontinuities and fine features. Despite these difficulties we are able to render new views of acceptable output quality. However, for these scene types it would be desirable to capture a larger number of images.

Since our method recovers all information solely from raw images, we generally require more input views than, e.g., methods with given depth information or those based on active stereo. However, since we can efficiently process a large number of images, this is not a real disadvantage but rather allows us to reconstruct even subtle view-dependent effects. Nevertheless, active methods could be easily integrated into our system. Future work should address improvements such as alpha boundaries for rendering scenes with complex silhouettes such as the Monkey example. Moreover, in order to achieve a more faithful silhouette reconstruction also for very coarse proxies, the splatting approach in Section 4.4.1 should be adapted to render the anisotropic, oriented splat shape computed during the optimization. We would also like to investigate the applicability of our method to free viewpoint video with combined spatio-temporal filtering or image completion for applications such as free viewpoint television for sports events or movies [MP04]. Finally, while the rendering phase is real-time capable, it would obviously be desirable to improve the reconstruction phase for true online processing of video streams.
5 High Quality Model Reconstruction

This chapter focuses on the reconstruction of high quality 3D models. Here, our main concern is the accuracy of the reconstructed surface geometry, rather than a visually plausible reproduction of the scene appearance as in the previous two chapters.

In Section 2.1.2 we classified existing approaches for image-based object reconstruction into two major categories. One class of techniques first generates a point cloud from point correspondences based on, for example, structure-from-motion, small baseline stereo depth maps, or laser scans. This 3D point cloud then has to be converted into a consistent surface representation (e.g., a triangle mesh) for further processing. The second class of approaches computes the object surface directly from the input images without an intermediate point-based representation.

In previous research, these different approaches, i.e., the reconstruction of a proper surface from 3D points or directly from a set of input images, have generally been considered as two separate problems. In this chapter we show that, by choosing an appropriate problem formulation and surface representation, both problems are closely related. This allows us to devise a single generic approach for both surface reconstruction problems. In addition, we show that the accuracy and efficiency of image-based reconstruction techniques depends considerably on the actual selection of input views, and we propose an automatic algorithm for choosing an optimal subset of images for reconstruction. In order to motivate our work, the following section provides a brief discussion of related approaches.

5.1 Discussion of Existing Techniques

**Surface Reconstruction from Images** Seminal work on multi-view stereo (MVS) reconstruction has been introduced in [SD97, KPV98, KS00b]. One problem in common with these methods and most related techniques based on the concept of voxel-coloring
is their inability to impose geometric constraints such as spatial coherence of the reconstructed surface (Figure 5.1 (b)). Nevertheless, more recent methods based on the robust computation of depth maps [GCS06, SFV06], feature matching and patch expansion [STV03, FP07], or splat-based surface growing [HK07] with an additional post-process to extract a consistent object model [KBH06] have presented multi-view stereo reconstructions of high accuracy [Mid08a].

For the direct reconstruction of a proper manifold surface from images, different methods based on deformable surfaces have been proposed using, e.g., level-set methods [FK98], stochastic refinement of the visual hull [IS03], or the integration of 3D and 2D data into a single optimization process [LQ03]. Reconstructions of considerable quality using high resolution images have been presented in [HS04]. Although it is possible to reconstruct a consistent object surface with these gradient-based methods, they often have a large computational complexity and they are generally sensitive to local minima in their respective target functional.

Recently, research on combinatorial energy minimization has shown that globally optimal solutions to related discrete segmentation and reconstruction problems can be found efficiently by reformulating them into a maximum flow / minimum cut problem of a spatial graph structure. First applications to computer vision included a solution
5.1 Discussion of Existing Techniques

to the voxel occupancy problem from visual hulls [SVZ00] and discrete labeling by iterated graph cuts for disparity-based stereo [BVZ01]. An extension to minimal geodesic contours and surfaces for arbitrary Riemannian metrics and an analysis of the types of energy functionals which can be minimized using graph cuts has been presented in [BK03b] and [KZ04], respectively. First applications to volumetric multi-view stereo were proposed in [PSQ04, SP05, VTC05]. The approach presented in [VTC05] is closest related to our work [HK06a] described in this chapter and will be discussed in more detail in Section 5.4. These works have initiated a large body of work on graph cuts for MVS reconstruction [LBI06, LPK07, SMP07, VHTC07].

Surface Reconstruction from Point Clouds Instead of reconstructing a consistent 3D surface directly from a set of input images, a large number of existing image-based reconstruction techniques creates a point cloud which then has to be converted in a post-process into a closed and manifold surface. Examples include the problem of merging several laser scans [Bla04] or small baseline stereo depth maps (see Section 4.3), or the previously mentioned work on MVS such as [HK07]. Most related work on surface extraction from such point clouds can be classified into the following approaches.

Voronoi-based techniques [ABK98, ACK01, BC02b, DG03] reconstruct a mesh directly from the input samples, with the advantage of computing output meshes with a complexity in the order of the input data, and with good results for data sets with known sampling density. Wrapping approaches [BMR∗99] provide a good local feature preservation. However, point clouds generated from images are typically non-uniformly sampled and may contain a considerable amount of noise and outliers. From this data it is difficult for the above mentioned approaches to guarantee the reconstruction of a globally optimal, watertight surface. Improvements with respect to these problems have been achieved in [MAVdF05, SFS05]. However, it is generally difficult to guarantee the reconstruction of a smooth and manifold surface, especially in the presence of noise and for varying sampling densities.

Methods based on deformable models for point cloud reconstruction have been presented in [EBV05, SLS∗06]. Here, the problem of computing a watertight surface is solved by incrementally deforming an initial mesh along an energy field induced by the point cloud, similar to optimizing energy functionals based on photo-consistency measures as in MVS. However, as already mentioned in the above discussion on deformable models, these methods are not guaranteed to find the globally optimal surface. More-
over, they have the potential problem of creating overly smoothed surfaces since it may be difficult to find appropriate surface tension parameters.

Most related to our work are volumetric approaches which reconstruct the unknown surface as the zero level-set of a signed distance function [HDD’92, CL96, CBC’01, ABCO’01, DMGL02, OBA’03, OBS04], e.g., using Marching Cubes [LC87]. These methods often rely on an accurate normal orientation and fairly uniform sampling densities of the input point clouds, which are both requirements generally not met by real world data sets. Additionally, they can be quite sensitive to noise, outliers, or misaligned surface patches, where they tend to introduce topological artifacts such as handles, bridges or isolated surface components due to spurious zero crossings of the signed distance function (see Figure 5.2). These artifacts are a fundamental problem of methods which extract the zero level-set of a signed distance function. As a consequence these techniques generally require subsequent post-processing [ESV97, GW01, NT03, WHDS04] for artifact removal or general mesh repair [BNK02, Ju04, BPK05]. Reconstruction techniques which address these issues have been presented in [FCOS05, Kaz05, KBH06, SLS’07]. Because of these intrinsic difficulties, recent research has focused on surface reconstruction from unstructured, unoriented points [ACSTD07] without the requirement for dense sampling and accurate surface normal information.

In [HK06a] and [HK06c] we have shown that multi-view stereo and surface reconstruction from point clouds are closely related and can be solved using a single approach based on an unsigned volumetric confidence map and graph cuts.

5.2 Conceptual Overview

The key observation for solving both reconstruction problems in a unified manner is that both can be formulated in terms of a surface confidence map $\phi : \mathbb{R}^3 \rightarrow \mathbb{R}_+$ which assigns each point in 3-space a pseudo-likelihood of being intersected by the unknown object surface $S^*$ (see Section 2.2.2). We interpret $\phi$ as an unsigned distance function (see Figure 5.2), i.e., it is expected to have values close to zero for points lying on the surface $S^*$, and larger values for points at an increasing distance to $S^*$. In Section 4.2.1 a corresponding representation of $\phi$ for multi-view stereo was introduced based on the notion of photo-consistency. As will be described in the following, a similar unsigned distance function can be derived from the 3D point samples of an input point cloud (see
5.2 Conceptual Overview

Figure 5.2: The fundamental problem of surface representations based on a signed distance function $d$ are the frequent sign changes and zero crossings caused by local inconsistencies (e.g., unreliable normal estimations) of the input data. This generally results in reconstructions of undesirably high genus with significant topological artifacts (upper left). In contrast, our approach for surface reconstruction is based on an unsigned confidence map $\phi$, which gracefully handles data inconsistencies (lower left). (b) shows an exemplary reconstruction of a point cloud with a considerable number of noisy and misaligned surface point samples, for which our method produces a smooth and closed genus zero reconstruction without any artifacts.

Also [PMG04]). Based on such a confidence map, a surface $S$ can be characterized by an energy functional of the form

$$E(S) = \int_S \phi(x) \, dS + a \int_S dS ,$$

(5.1)

which represents a weighted sum of the integrated surface confidence and the area of the surface. The desired optimal surface $S^*$ should minimize this functional (subject to certain constraints to exclude degenerate solutions) for a maximally confident and smooth approximation to the true object surface. We find this surface $S^*$ by a multi-resolution discretization and minimization of Equation 5.1 using hierarchical graph cuts. Our approach has several important characteristics which are listed in the following.
5 High Quality Model Reconstruction

Correctness Our method finds the globally optimal (discrete) minimizer of the surface energy in Equation 5.1 by computing the minimum cut of a spatially embedded graph structure. A proper mapping of the discretized confidence map \( \phi \) to the graph edge weights is ensured by a new octahedral graph structure. Our multi-resolution approach enables an iterative improvement of visibility and shape priors, and supports reconstructions at high spatial resolutions.

Robustness Due to our problem formulation based on an unsigned rather than a signed distance function our method can process data without any normal information and it is immune to noisy and non-uniform input. In contrast to deformable surfaces our surface energy minimization based on graph cuts is robust to local minima of the energy functional. Moreover, the hierarchical approach results in a reduced computational complexity which allows us to process large input data sets, i.e., complex point clouds or hundreds of input images for a more robust handling of difficult object shapes and imperfect acquisition conditions.

Efficiency Because of our hierarchical approach a high spatial resolution and reconstruction accuracy can be achieved significantly faster compared to computing the solution directly at the volumetric target resolution. Moreover, the discrete energy optimization by computing the min-cut of our regular graph structure can be performed in low-order polynomial time complexity [BK04b]. Finally, further computational steps such as the photo-consistency estimation (see Section 4.2) can be efficiently performed on the GPU.

Our algorithm for high quality surface reconstruction consists or four central ideas which are outlined in the following.

5.2.1 Volumetric Confidence Map

For the representation of the confidence map \( \phi \) we use a discrete volumetric grid \( \mathcal{V} \) where each voxel \( v \in \mathcal{V} \) gets assigned a confidence value \( \phi : v \rightarrow \mathbb{R}_+ \) (see Figure 5.3). For multi-view stereo reconstruction we have already shown in Section 4.2.1 that these confidence values can be estimated from a set of input images \( I_j \) by computing the photo-consistency of a volumetric scene element. In this case these elements are defined by the voxels \( v \in \mathcal{V} \) instead of particles. For input in the form of point samples \( p \) of a point cloud \( \mathcal{P} \) this map \( \phi(v) \) can be interpreted as the unsigned distance \( \| v - p \|_2 \) of a voxel center \( v \) to the closest point sample position \( p \).
5.2 Conceptual Overview

Figure 5.3: This figure illustrates the graph based surface reconstruction in 2D. For the case of an input point cloud \( \mathcal{P} \), (a) shows an exemplary sparse set of occupied voxels containing samples of \( \mathcal{P} \). From these voxels, we compute a crust \( \mathcal{V}^{\text{crust}} \) and, for all voxels \( v \in \mathcal{V}^{\text{crust}} \), a confidence value \( \phi(v) \) by volumetric diffusion (b). In the multi-view stereo setting \( \phi \) is computed directly from the input images based on the photometric consistency of a voxel (darker colors indicate higher confidence values). Typically, this confidence map has many local minima and maxima due to non-uniform and noisy sampled input points, or color ambiguities, image noise, and other error sources in the MVS setting. We embed a spatial graph structure \( \mathcal{G} \) within the voxel grid with small edge weights (green) for high confidence voxels and vice versa (c). The interfaces to \( \mathcal{V}^{\text{ext}} \) and \( \mathcal{V}^{\text{int}} \) are connected to a graph source \( s \) and a sink node \( t \), respectively. Computing the min-cut of this graph then yields the desired surface \( \mathcal{S}^* \) (d).

Instead of computing \( \phi \) for all voxels \( v \in \mathcal{V} \) we first compute an initial geometric scene proxy from the input data and then constrain the computation of \( \phi \) to a dilated crust \( \mathcal{V}^{\text{crust}} \subseteq \mathcal{V} \) around this proxy. As we will show in later sections, this allows us to achieve a higher efficiency as well as an improved accuracy. Moreover, this approach prevents degenerate solutions during the minimization of Equation 5.1. The computation of \( \mathcal{V}^{\text{crust}} \) and the confidence map \( \phi \) is described in Section 5.3.
5.2.2 Energy Minimization based on Graph Cuts

Given such a set of confidence weighted voxels we want to extract a subset of voxels $S^* \subseteq V_{\text{crust}}$ which represents a closed and smooth 2-manifold surface with maximum confidence, i.e., which minimizes the discretization of the energy functional given in Equation 5.1:

$$E(S) = \sum_{v \in S} \phi(v) + \sum_{v \in \partial S} a.$$  \hspace{1cm} (5.2)

Methods for iso-surface extraction from implicit functions [LC87] are not suitable to reconstruct a surface represented by such a confidence map, since they depend on the zero-level of a signed distance function rather than the unsigned confidence values of $\phi$. However, previous work [BK03b, KZ04] has shown that similar types of combinatorial optimization problems involving the minimization of certain energy functionals can be efficiently solved by transforming them into a max-flow / min-cut problem of an embedded spatial graph $G$.

A graph $G := (N, E)$ consists of a set of nodes $N$ and undirected graph edges $E$ connecting pairs of nodes. With each edge $e \in E$ an edge weight $\omega_e \geq 0$ is associated. Two of the nodes in $N$ are designated terminal nodes, namely the graph source $s \in N$ and the graph sink $t \in N$. A $s - t$ cut is then defined as a subset of edges $C \subseteq E$ where the nodes $s$ and $t$ are separated in $G(C) := (N, E \setminus C)$. The cost of a cut is defined as the sum of weights of the edges in $C$:

$$E(C) = \sum_{e \in C} \omega_e.$$  \hspace{1cm} (5.3)

Such a cut $C$ is called a min-cut, if its cost $E(C)$ is minimal among all possible $s - t$ cuts. It is known from combinatorial optimization literature [FF62, GT88] that the globally optimal solution to this problem can be found efficiently with low-order polynomial time algorithms [BK04b].

The underlying idea of our method is to encode our consistency map $\phi$ in terms of edge weights of such an embedded graph $G$, and to connect the two terminal nodes of the graph to the interface between $V_{\text{crust}}$ and an outer volumetric component $V_{\text{ext}}$ and an inner component $V_{\text{int}}$, respectively (see Figure 5.3 (c)). The minimum cut $C$ through $G$ then defines a set of surface intersected voxels $S^*$ which corresponds to the desired closed and manifold surface minimizing Equation 5.2. This relation between graph edges, voxels and confidence values, and the actual construction of $G$ are described in Section 5.4.
5.2 Conceptual Overview

5.2.3 Hierarchical Approach

For high volumetric resolutions the computational complexity for generating the unsigned distance function and the graph cut computation become impractical. In MVS one of the dominating computational factors is the consistency estimation (see Section 4.2). For non-uniformly sampled point clouds it is generally difficult to estimate an optimal volumetric grid resolution such that holes in sparsely sampled areas can be efficiently detected and closed without losing details in densely sampled regions. By the integration of the confidence estimation and the graph-based surface extraction into a hierarchical framework these problems can be effectively resolved.

As mentioned above, our method first computes an initial surface proxy at a low volumetric resolution, dilates this proxy to a crust $V_{\text{crust}}$, and then computes a surface approximation $S^\star$. The basic idea of our hierarchical approach is to use an adaptive volumetric grid (e.g., an Octree [SW88]), and to refine this grid only in the vicinity of this surface $S^\star$. Hence, given a surface approximation $S^\star_l$ at a volumetric level $l$, we define the new crust $V_{\text{crust}}^{l+1}$ at the next level $l+1$ by first refining the voxels $v \in S^\star_l$ and then applying a number of morphological dilation steps at this higher resolution level. Within this new crust, a new confidence map and surface approximation $S^\star_{l+1}$ is computed. These steps are iterated until the desired target resolution is reached. The details of this algorithm are explained in Section 5.5.

5.2.4 Surface Mesh Extraction

Once the desired target resolution is reached the minimum cut $\mathcal{C}$ representing the surface $S^\star$ has to be converted into a triangle mesh for further geometric processing. At this stage one could apply existing iso-surface reconstruction techniques such as [LC87]. This however would imply a conversion of the cut surface into a signed scalar field, and the resulting level-set surface would not necessarily respect the surface topology defined by the graph cut edges $\mathcal{C}$ anymore. In Section 5.6 we derive a simple and efficient algorithm for the manifold extraction directly from the geometric interpretation of our graph embedding and the computed min-cut edges of $\mathcal{G}$.
Figure 5.4: Illustration of the initial object proxy generation from segmented input images. The volumetric grid on the left represents a single slice (red line in images $I_j$ and $I_k$) through the volume $V$. The initial crust $V_{\text{crust}}$ is generated by carving away all voxels projecting outside of the object silhouette in the input images. Although the resulting visual hull often represents a good first approximation of the object shape, the resulting proxies can be subject to wrong surface estimates such as the additional pillar in the middle of $V_{\text{crust}}$.

5.3 Surface Confidence Estimation

As mentioned in Section 5.2.1 we estimate the confidence values $\phi(v)$ for all voxels within a crust $V_{\text{crust}}$. The process of creating an initial crust at the starting level $l_0$ and the computation of $\phi$ from multiple views as well as from point clouds is described in the following two sections. We will drop level indices $l$ for a simplified notation, since all voxels $v \in V_{\text{crust}}$ are at the same refinement level.

5.3.1 Image-based Surface Confidence

For the image-based confidence estimation we basically apply the technique for photo-consistency estimation described in Section 4.2.1. However, in a multi-view stereo setup with many different vantage points it is generally more involved to estimate whether a voxel is visible or occluded in the available input images $I_j$. Thus, the generation of a coarse initial voxel crust $V_{\text{crust}}$ is not only required for the embedding of the graph $G$, but also for computing visibility information for the photo-consistency estimation.
5.3 Surface Confidence Estimation

Figure 5.5: Using our particle-based stereo technique an improved initial surface approximation $V_{crust}$ can be generated by carving away all voxels lying “in front” of the computed depth maps (dashed line). This approach also captures concave object regions which cannot be recovered from the image silhouettes.

**Crust Generation** We can generate this initial object proxy from the visual hull [Lau94] using segmented input images [RKB04], or by using small baseline stereo (see Section 4.3). As we have shown in [HK06b] a voxelized visual hull can be computed efficiently on the GPU by testing whether a voxel projects onto the fore- or background in the segmented input views (see Figure 5.4). This step is basically identical to the projection pass in Algorithm 4.1.

Although such an initial proxy computed from image silhouettes is often sufficient and has been used successfully in related work on MVS, a better surface approximation can be achieved by employing small baseline stereo. After computing a set of depth maps from different viewing positions, the voxelized geometry proxy $V_{crust}$ is generated by carving away all voxels lying “in front” of each of the depth maps (see Figure 5.5). Instead of simple carving one could apply more sophisticated visibility voting schemes for a more robust handling of outliers in the depth maps. But due to the sufficient quality of our particle-based stereo technique this carving approach produced reliable results in our experiments.

Such a proxy then allows to compute, whether a voxel is visible or occluded in a particular view $I_j$. For the visibility estimation, we first estimate approximate surface normals on the outer proxy surface $\partial V_{crust}$ (see Figure 5.6) and then propagate this information
inwards through the remaining voxels of $V^{\text{crust}}$. For each voxel $v \in \partial V^{\text{crust}}$ we compute the k-nearest neighbor voxels $N(v) \subset \partial V^{\text{crust}}$ and fit a regression plane to the center points $v_i$ of the voxels $v_i \in N(v)$ (see Figure 5.7). The normal $\mathbf{n}$ of $v$ is estimated by computing the center of gravity $\bar{v} = \frac{1}{|N(v)|} \sum v_i$ and solving the overdetermined homogeneous linear equation system $\forall v_i : \mathbf{n}^T (v_i - \bar{v}) = 0$ using SVD [PFTV92]. A consistent outward orientation can be enforced by traversal of all voxels in $\partial V^{\text{crust}}$. The number of neighbors $k$ simply controls the smoothness of the normal field. We typically set this value to $k = 25$ to minimize discretization artifacts due to the voxelization. Finally, the computed normals are propagated inwards through $V^{\text{crust}}$ by a morphological
5.3 Surface Confidence Estimation

(a) Depth map of backfacing voxels.  (b) Visibility estimation by depth comparison.

Figure 5.8: For computing the visibility of a voxel $v$ in an image $I_j$ we first compute a depth map of all backfacing voxels of $\partial V^{\text{crust}}$ in image $I_j$ (a). The visibility of each voxel $v \in V^{\text{crust}}$ can then be evaluated by a depth comparison of the projection of $v$ and the corresponding entry $v_b$ in the depth map (b).

The basic idea to compute the visibility of a voxel in an image $I_j$ is then to render an appropriate occlusion surface as seen from the corresponding viewpoint [HK06b] (see Figure 5.8). This is an operation which can be performed efficiently on a modern GPU by exploiting splat-based rendering techniques [KB04]. The difficulty lies in choosing the proper occlusion surface for computing the visibility of all voxels $v \in V^{\text{crust}}$, since the thickness of $V^{\text{crust}}$ is generally more than one voxel. However, this problem can be effectively solved using only the backfacing boundary of $V^{\text{crust}}$ with respect to $I_j$, which can be identified using the computed surface normals. We set the projection of the rendering system according to the calibration data of $I_j$ and render all backfacing $v \in \partial V^{\text{crust}}$ by replacing each voxel with a screen-aligned quad located at the center of the voxel. This is achieved by sending one point primitive per voxel to the GPU. In order to render a closed surface, the projected screen-size of each rendered primitive is computed in a vertex shader [BK03a], conforming to the size of its corresponding voxel in the volumetric grid. The result is a dense depth map (see Figure 5.8 (a)) of all outer boundary voxels on the backside of $\partial V^{\text{crust}}$ as seen from image $I_j$. The visibility for all $v$ in image $I_j$ can then be computed efficiently by a simple depth comparison (see Figure 5.8 (b)).

As mentioned in Section 5.2.2 our graph cut algorithm requires a common boundary of $V^{\text{crust}}$ with an exterior $V^{\text{ext}}$ and interior $V^{\text{int}}$ volumetric component for connecting the erosion operator on the voxel grid and simple averaging of normals from those directly 26-connected voxel neighbors which were deleted in the respective previous erosion step.
5 High Quality Model Reconstruction

(a) Voxel and surface sampling.  
(b) Photo-consistency $\phi$ for the Leo and the Temple model.

**Figure 5.9:** The image-based consistency estimation is based on the same spatial supersampling (a, top) as the particle-based approach described in Section 4.2.1. Due to our hierarchical approach which computes iteratively improved surface proxies, we can additionally employ non-planar surface approximations at higher volumetric resolutions (a, bottom). This is necessary as soon as the projected footprint of a single voxel has a size in the order of just a few pixels and hence does not provide a sufficient integration domain for the photo-consistency estimation. (b) shows two cut outs of the photo-consistency volume computed for the Leo and the Temple. The consistency maximum representing the surface location is visible as a thin black band. However, noise and ambiguous regions are apparent as well.

Graph terminal nodes during the surface computation. The exterior interface $\partial V^{\text{crust}}$ is given by the above proxy computation. However, the computed normal field allows for a simple algorithm to determine the interior component as well by an approximation to the inner medial axis of $V^{\text{crust}}$. For each voxel we estimate a normal cone by collecting the normals of all 26 neighboring voxels and label each voxel as $V^{\text{int}}$ if the opening angle of this cone lies above a threshold of $\pi/2$. The actual choice of this threshold does not have a significant influence on the results, since we basically just want to find discontinuities in the normal field (see Figure 5.6 (b)).

**Voxel and Surface Supersampling** Based on the approximate surface normals and the visibility estimation, the photo-consistency of a voxel is computed using the approach
presented in Section 4.2. We supersample each voxel $v$ (see Figure 5.9), project the samples into the input images where $v$ is visible, and compute the normalized color variance as in Equation 4.3. For accurate consistency estimates with distinct extrema a sufficient parallax between the input views is required. However, the influence of images from too oblique viewing angles should be suppressed due to the significantly reduced texture resolution and accuracy in these regions. Hence, we weight the contribution of each input image $I_j$ to Equation 4.3 by a Gaussian weight $\omega_j(\lambda) = e^{-(\lambda-\mu)^2/w}$.

$\lambda$ corresponds to the angle between the surface normal $n$ and the direction from the center $v$ of voxel $v$ to the camera center $c_j$ of image $I_j$, and is computed as $\lambda = n^T (c_j - v) / \| c_j - v \|$. $\mu$ represents the angle of the prioritized viewing direction with respect to the surface normal. Finally, $w$ controls the width of this bell-shaped weighting function. For the reconstructions shown in this work we set $\mu = 0.866$ corresponding to approximately $30^\circ$ deviation from the surface normal direction and $w = 0.05$. An example plot of the weighting function for different values of $\lambda$ is shown in Figure 5.10. Prior to computing $\phi$ using Equation 4.3 the weights are normalized such that $\sum_i \omega_i = 1$.

The voxel supersampling approach provides a robust consistency measure as long as the projection of a voxel covers at least a few pixels in the input images. However, if the object space voxels are too small relative to the pixel resolution of the images this method tends to become unstable due to alias errors, e.g., when applying bilinear interpolation of color values (see Figure 5.11 (a)). Thus, we have to enlarge the integration domain in this case by adding neighboring voxels. In our iterative optimization setting with a sequence of improved object proxies $V^{\text{crust}}_l$ at different resolution levels $l$, we are in fact able to use non-planar surface sampling for the consistency estimation. Similar to the normal estimation step above, we again compute the $k$-nearest neighbors of a voxel $v \in \partial V^{\text{crust}}$. Then, instead of supersampling only $v$, we create samples for each of the $k$ neighbor voxels as shown in Figure 5.9 (a) and then compute the photo-consistency. The same procedure is repeated for the remaining voxels by an iterated erosion of $V^{\text{crust}}$. While this is conceptually similar to the patch-based NCC (see Section 4.2), we can exploit a non-planar surface approximation in contrast to planar patches using NCC. Again, the
5 High Quality Model Reconstruction

Figure 5.11: Reconstruction of the Leo head at a high volumetric resolution where the projected footprint of a single voxel has an area in the order of less than 10 pixels. (a) With such a small integration domain the voxel supersampling causes small oscillations and artifacts due to the image noise and color interpolation errors. (b) The surface sampling shows a much better reconstruction quality in terms of surface detail and smoothness, e.g., at the ear or the top of the Leo’s head.

The matching problem is implicitly avoided by the projection of 3D sample positions. This approach results in smooth surface reconstructions even at high volumetric resolutions relative to the resolution of the input images (see Figure 5.11 (b)). Our algorithm switches from the voxel to the surface sampling approach for volumetric resolutions, where a single voxel projects to less than $5^2$ pixels.

5.3.2 Confidence Diffusion for Point Clouds

For input in the form of a point cloud $P$ the crust $V_{\text{crust}}$ and the confidence map have to be estimated from the point samples. Initially, we insert each 3D sample $p \in P$ into the volumetric grid $V$, resulting in a sparse set of occupied voxels (see Figure 5.3 (a)). As mentioned above the probability or confidence that a voxel $v$ is part of the unknown surface represented by the point cloud can be approximated by an unsigned distance function $\phi$ over $V$ (see also [PMG04]). To compute $\phi$ in the vicinity of the input point samples we apply several steps of a morphological dilation operator to the 6-neighborhood of occupied voxels. This dilation generates the crust $V_{\text{crust}}$. The distance function $\phi(v)$ for each $v \in V_{\text{crust}}$ is computed by volumetric diffusion (see Figure 5.12).
5.3 Surface Confidence Estimation

Figure 5.12: In analogy to Figures 5.6 and 5.9 this figure shows the input point cloud of the Buddha model (left), the resulting volumetric crust $V^{crust}$ (middle), and the computed surface confidence map $\phi$ (right).

**Crust Generation**  For the graph-based surface computation we have to ensure that the computed crust is watertight (i.e., 6-connected), again with an interface to an exterior $V^{ext}$ and interior $V^{int}$ component (see Figure 5.12, middle). The number of necessary dilation steps for computing this crust can be computed robustly with a simple heuristic. By flood-filling unoccupied voxels from the outer boundaries of $V$ we can easily determine the current number of different volumetric components separated by $V^{crust}$. Initially, there is only one (outer) component. This number increases during the dilation process as the crust grows, and eventually drops down again to one component when the “interior” of the point cloud $P$ is full of occupied voxels. $V^{int}$ is then simply defined by the voxels conquered during the last dilation steps. This repeated flood-filling and dilation process is computationally irrelevant in our overall hierarchical setting, since we generally start at low volumetric resolutions of $64^3$ or $128^3$.

For incomplete point clouds covering only a part of the surface of an object as in Figure 5.25 (a) or objects with relatively thin, elongated features and non-uniform sampling density (b) it is sometimes not possible to compute a proper interior component $V^{int}$. In
these cases our algorithm employs the approximation to the medial axis of the dilated
crust as described in the previous section.

**Volumetric Diffusion** For the computation of $\phi$ we first assign distance values $\phi(v) = 0$
to voxels containing surface samples $p$, and $\phi(v) = 1$ for the remaining voxels in $V_{\text{crust}}$.
The diffusion is then simply performed by iterative averaging over the 6-neighborhood $N(v) \in V_{\text{crust}}$ of a voxel $\phi(v) = \frac{1}{|N(v)|} \left( \phi(v) + \sum_{u \in N(v)} \phi(u) \right)$ while keeping $\phi(v) = 0$
fixed for voxels containing surface samples (see Figure 5.12, right). The overall algorithm
is not very sensitive to the number of diffusion steps. In fact, a valid surface can already
be computed after the initialization of $\phi$ without any diffusion. However, the surface
becomes smoother with more diffusion steps. Moreover, as we will see in Section 5.6, the
unsigned distance values allow for a confidence weighted mesh smoothing of the extracted
mesh. For our results we used three diffusion steps for all reconstructed models.

Optionally, one can compute the initial confidence values from the sample density
within a voxel instead of setting all occupied voxels to $\phi(v) = 0$. However, the current
approach shows a better handling of non-uniformly sampled regions. Similarly, keeping
$\phi(v) = 0$ fixed for voxels containing surface samples instead of including them in the
diffusion process preserves fine details more faithfully.

### 5.4 Graph Construction and Surface Computation

For the computation of the surface via graph cuts, the central question is the actual
graph structure and the mapping of the confidence values $\phi(v)$ of each voxel to the graph
edges. Previous work discussed in Sections 5.1 and 5.2.2 has shown how graph cuts can
be used to solve the problem of computing proper *segmentation* surfaces in-between the
voxels of a volumetric scene representation. This segmentation-based formulation has
been employed in [VTC05] for high quality MVS reconstruction. However, there exists
a subtle but important issue in using these segmentation approaches for volumetric 3D
object reconstruction.

The confidence map $\phi$ integrates, for a specific region in space, the likelihood of
being *intersected* by the surface. Hence, it is well defined only for a proper integration
domain, e.g., geometrically valid entities such as voxels. The above segmentation based
approaches generate a graph structure which associates graph nodes with voxel centers
5.4 Graph Construction and Surface Computation

and graph edges with voxel faces. This poses the question of properly defining the edge weights of the embedded graph. If we compute the confidence values for the voxels, these values have to be re-mapped to the graph edges, e.g., by taking the average consistency of two face connected voxels. This, however, is equivalent to applying a low-pass filter to the voxel consistency values and hence reduces the effective resolution of the reconstruction.

In [VTC05] the surface confidence (in the form of photo-consistency) is computed for the graph edges instead of voxels. But while the projection of a voxel and hence the respective integration domain for the photo-consistency estimation is well defined (see Section 4.2), there is no such definition for projecting graph edges into images and computing their confidence values. A simple integration of point samples over 1D edges or 2D faces would introduce a directional bias in the 6-connected grid, and the geometrical interpretation becomes even more unclear if one wants to extend this approach to larger voxel neighborhoods to allow for better surface integral approximations [BK03b]. Although these approaches are well suited for problems where voxels have to be segmented into different classes by a contour or surface at voxel boundaries, they do not support the extraction of a proper surface intersecting the interior of voxels. Additionally, it is not clear how the weights of the corresponding graph edges should be defined based on a confidence value computed per voxel.

The following section describes a graph structure which establishes a proper relation between voxel confidence values, the embedded graph, and the reconstructed surface by representing each voxel by its dual octahedral subgraph.

5.4.1 Octahedral Graph Structure

Our goal is to find the set of surface voxels \( S^* \subseteq \mathcal{V}^\text{crust} \) which are intersected by the unknown object surface by minimizing the discretization of the surface energy functional defined in Equation 5.1. Within the voxel grid \( \mathcal{V} \) consisting of cubical voxels with square faces, the crust \( \mathcal{V}^\text{crust} \) properly separates the interior component \( \mathcal{V}^\text{int} \) of \( \mathcal{V} \) from the exterior component \( \mathcal{V}^\text{ext} \) (e.g., Figure 5.6), since \( \mathcal{V}^\text{crust} \) is a face-connected (6-neighborhood) set of voxels. Suppose now we have an arbitrary closed surface \( S \) which intersects \( \mathcal{V}^\text{crust} \). For each voxel in \( \mathcal{V}^\text{crust} \) we can label its faces as interior or exterior depending on which side of \( S \) they lie. Faces that are intersected by the surface are labeled as interior by default. The important observation now is that if we want to separate the interior faces
Figure 5.13: Because of the duality of the voxel and the octahedron, a cut through the octahedral subgraph corresponds to a split of the voxel faces into an exterior and an interior component (a), (c). Simple configurations with four or six cut-edges correspond to locally planar cut surfaces (b) while complex configurations with eight cuts correspond to locally curved cut surfaces (d).

from the exterior faces for a single voxel, we have to cut along a sequence of edges of the voxel (see Figure 5.13).

Based on this observation we build the following graph structure. For each voxel face in $V^\text{crust}$ we define a node in the graph $G$. Within each voxel $v$ we connect the six nodes corresponding to the six faces in an octahedron-fashion and assign the confidence value of the voxel plus a surface area constant

$$\omega_e = \phi(v) + a$$

(5.4)

to all twelve edges. Due to the duality of the cube (voxel) and the octahedron, we have a one-to-one correspondence between the edges of both. Hence, the above voxel cut (along edges) which separates interior from exterior faces is equivalent to a graph cut in the octahedron which separates the corresponding nodes (see Figure 5.13). The global graph $G$ embedded in $V^\text{crust}$ consists of the sub-graphs of all voxels. The graph source $s$ is connected to all nodes whose associated faces lie at the interface to the exterior component $V^\text{ext}$, while all nodes at the interior component $V^\text{int}$ are connected to the sink $t$ (see Figure 5.3).

Computing the minimum $s - t$ cut of this graph yields a set of cut edges $C$ which minimizes the sum of edge weights according to Equation 5.3, and which defines a
closed surface that separates the graph into two components by splitting those voxels 
\( S^* \subseteq V^\text{crust} \) which are most likely intersected by the true object surface.

The geometric smoothness of the resulting cut surface is enforced by two aspects 
of the graph embedding: First, within each voxel, a configuration which cuts as few 
octahedron edges as possible is preferred (since they all have the same weight). These 
preferred configurations cut either four or six edges and correspond to planar cuts while 
more complex cuts with eight or more edges correspond to curved configurations (see 
Figure 5.13). Moreover, since the cost of a separating cut is the sum of the weights of 
all edges that are split, the optimal solution might rather contain inconfident voxels if 
this leads to a better global solution than summing over a larger number of voxels with 
better confidence values.

Since the edges are embedded fairly uniform in the voxel space, the sum over all 
edge weights \( \phi(v) + a \) can be considered as a decent, discretely sampled surface integral 
approximation of the surface confidence and area. Therefore a cut through this 
graph minimizes the discretization in Equation 5.2 of the surface energy functional in 
Equation 5.1 up to a constant scaling factor which depends on the voxel size.

The trade-off between local confidence and global surface area minimization can be 
controlled to a certain extend by applying a suitable transfer function \( \phi'(x) = \phi(x)^s \) 
with a smoothness factor \( s > 0 \), and by changing the surface area constant \( a \). Due 
to our hierarchical approach, different choices of these parameters can be evaluated 
by the user with only short delays. However, in our experiments we kept these two 
parameters constantly set to \( s = 4 \) and \( a = 10^{-5} \) for all resolution levels and experiments.

This graph-based surface reconstruction supports our initially formulated requirements 
concerning the minimized energy functional, the definition of edge weights, and the 
extration of surface-intersected voxels \( S^* \).

5.5 Hierarchical Crust Refinement

As described in Section 5.2.3 we use an iterative hierarchical approach that starts at 
a coarse volumetric crust \( V^\text{crust}_l \) with \( l = l_0 \), from which we compute an initial surface approximation \( S^*_l \). This volume is then refined to the next higher grid resolution and 
dilated to a new crust \( V^\text{crust}_{l+1} \). We iterate this process until the desired target resolution 
is reached (see Figure 5.14). Assuming a reconstruction accuracy up to one voxel at
the respective previous level, a fixed distance crust generated from four dilation steps proved to be sufficient in our experiments.

A first description and analysis of hierarchical, banded graph cuts based on 6-connected grids for fast segmentation has been presented in [LSGX05]. The fact that the surface extraction at level \( l + 1 \) is constrained by the crust computed from the respective surface \( S_l \) on the previous level implies that one does not compute the globally optimal solution with respect to the target resolution anymore, and that it is possible to miss fine surface details not contained within the fixed distance crust. A solution to this problem has been proposed in [LB07]. It is, however, important to realize that the quality of the globally optimal solution depends on the definition of the surface energy functional (see Equation 5.1) and, in particular, on the computation of the confidence map \( \phi \).

In the following two sections we will show that our iterative hierarchical approach can improve not only the efficiency but the overall quality and accuracy of the reconstructed surface as well.

5.5.1 Iterative Visibility Update

As mentioned in Section 5.3.1 the computation of \( \phi \) in the MVS setting depends significantly on a correct visibility estimation of the voxels in the input views. Although there exist statistical approaches to handle occlusions as well [VHTC07], an explicit geometric scene proxy can help to resolve visibility issues more reliably and efficiently.

Despite the fact that an iterative surface computation based on deformable models can lead to wrong surface estimates due to local minima of \( \phi \) (see Section 5.1), corresponding methods have the advantage of supporting an iterative improvement of the visibility information. In contrast, methods based on graph cuts compute a globally optimal surface, but do not provide the possibility to update visibility information during the optimization phase (see also the discussion of level sets and graph cuts in [Vog06]).

In our iterative setting we can re-estimate and improve the visibility in an incremental manner by recomputing the surface normals and occlusions at each iteration step. Moreover, we can employ the non-planar surface sampling for the photo-consistency estimation as described in Section 5.3.1. So despite loosing the property of computing the globally optimal solution of Equation 5.1, our multi-resolution approach actually leads
to a significantly more reliable computation of \( \phi \) and correspondingly more accurate MVS surface reconstructions in practice.

### 5.5.2 Hole Filling and Detail Preservation

For non-uniform point clouds containing sparsely sampled regions as well as fine details, the complexity of computing \( \phi(v) \) in a sufficiently dilated crust \( \mathcal{V}^{\text{crust}} \) and the cut computation directly at the volumetric target resolution becomes impractical. Starting at a low volumetric resolution allows for an efficient generation of a proper initial crust even for point clouds containing large gaps. As mentioned above, a straightforward approach using banded graph cuts implies the danger of losing fine details [LSGX05] if the corresponding point samples are not contained inside the crust. In our application setting, where the explicit data samples of the input point cloud are available, we can effectively avoid these problems while preserving the advantages of a hierarchical approach.

As in the previous section, we compute a surface approximation \( S^*_l \) within a crust \( \mathcal{V}^{\text{crust}}_l \), and then compute a new, thinned crust \( \mathcal{V}^{\text{crust}}_{l+1} \) on the next higher resolution level by refining and dilating the surface voxels \( v \in S^*_l \). To preserve fine details represented by point samples of \( \mathcal{P} \) outside of this crust, we re-insert the corresponding input samples as occupied cells into the volumetric grid at the new resolution \( l + 1 \) and dilate these cells until they merge with \( \mathcal{V}^{\text{crust}}_{l+1} \) (see Figure 5.14). Our experiments showed that a fixed number of three dilation steps is generally sufficient for the merging process. The cut computation then automatically includes these voxels for the surface extraction, and fine details are preserved.

For point clouds the number of dilation steps required to compute the thinned crust \( \mathcal{V}^{\text{crust}}_{l+1} \) basically depends on the amount of noise of the input data \( \mathcal{P} \). Without noise two dilation steps at level \( l + 1 \) would be sufficient assuming that the low frequency parts of the surface have been reconstructed up to voxel accuracy at level \( l \). As mentioned above our experiments with real data showed that four dilation steps proved to be a good choice. However, since all original data points outside of the crust are re-inserted and dilated in the higher resolution grid anyway, the choice of this parameter is mostly of importance for the computational performance. An unnecessarily thick crust would include too many voxels which do not contribute to the surface extraction at all, while a too thin crust would lead to a higher number of samples outside of the crust, such that a higher number of additional dilation steps is necessary for the merging process.
5 High Quality Model Reconstruction

Figure 5.14: Surface reconstruction for a point cloud of the Buddha model from the Stanford 3D Scanning Repository (see Figure 5.12). Starting at a volumetric resolution of $64^3$ this image sequence shows a hierarchical reconstruction up to $512^3$. On each level the images show the reconstructed mesh and the crust of consistency values in alternating order. Re-inserted detail samples are shown in red. Note that reconstructed thin features and the genus of the model do not depend on the respective approximation on the previous levels, but automatically adapt to the current resolution. Our algorithm reduces the original genus of the input model from $>100$ to 10. A few additional holes are introduced in regions, where the point samples of opposite surface sheets lie very close compared to the volumetric resolution, e.g., as shown in the lower right close up.
The proposed algorithm computes a closed surface representation even from strongly non-uniformly sampled point clouds. Large gaps are effectively closed with a reasonable surface due to the surface tension in Equation 5.1, and fine details are preserved due to the iterative point insertion and dilation. To summarize, the hierarchical reconstruction consists of the following three phases. Given a crust \( V^\text{crust}_l \) at resolution \( l \):

1. Surface confidence estimation
   - Multi-view Stereo
     - Compute normals and visibility for each voxel \( v \in V^\text{crust}_l \).
     - Compute the photo-consistency \( \phi(v) \) of all \( v \in V^\text{crust}_l \).
   - Point Clouds
     - Insert the points in \( P \) as occupied voxels \( v \) into \( V_l \).
     - Dilate these voxels until they merge with \( V^\text{crust}_l \).
     - Compute \( \phi(v) \) for all \( v \in V^\text{crust}_l \) by volumetric diffusion.

2. Graph-based surface extraction
   - Create \( G \) consisting of octahedral subgraphs for each \( v \in V^\text{crust}_l \) with edge weights according to Equation 5.4. Nodes at the boundaries of \( V^\text{crust}_l \) are connected to the respective terminal node of \( G \).
   - Compute the min-cut of \( G \), resulting in a set of surface intersected voxels \( S^*_l \) and cut edges \( C \).
   - If \( l \) is the target resolution then terminate and extract the final surface mesh (see Section 5.6).

3. Volumetric refinement
   - Refine the resolution of the surface voxels \( S^*_l \) and set \( l = l + 1 \).
   - Compute the new crust \( V^\text{crust}_{l+1} \) by dilation and proceed with step 1.

5.6 Mesh Generation

The final step of the reconstruction algorithm is the generation of a polygonal representation from the set of surface voxels \( S^* \) and cut edges \( C \). As mentioned before, existing
Figure 5.15: The computed min-cut of $G$ splits the voxels in $S^\star$ into exterior and interior faces (a). The set of cut-edges $C$ defines a loop of split-edges for each voxel, corresponding to a non-planar polygonal face (b). In the dual mesh, the non-planar polygonal faces are defined by split-corners which are shared by at least three surface voxels. The corresponding mesh is extracted by placing a single mesh vertex at the center of each voxel, and visiting the voxels associated with a split-corner by cycling over shared split-edges (c). The resulting polygonal mesh is shown in (d).

methods for iso-surface extraction require a conversion of the graph cut edges into a scalar field, which might have negative implications with respect to the accuracy and the topology of the reconstructed surface.

Our solution to this problem exploits the duality of the octahedron and the cube described in Section 5.4: A cut through the octahedron edges in $G$ can be interpreted as a split along the cube edges, which separates inside and outside faces (see Figure 5.15 (a)). By this, the global graph cut through the voxels $S^\star$ defines a closed and 2-manifold polygonal mesh $M$ with non-planar faces. The vertices of $M$ lie on the voxel corners and the mesh edges coincide with voxel split-edges (see Figure 5.15 (b)). However, although one could create a proper triangle mesh directly by triangulating either the set of inside or outside faces, or the polygonal faces defined by the loop of split-edges, these approaches would not generate a minimal number of vertices and triangles. Since the confidence values are estimated per voxel, we instead extract the polygon mesh $M'$ which is dual to $M$, i.e., vertices are located in the voxel centers and the non-planar polygonal faces correspond to voxel corners.

This polygonal mesh $M'$ can easily be generated by visiting all $2 \times 2 \times 2$ blocks of voxels. For each block $B$ the center voxel corner corresponds to a polygon face of $M'$.
if the block contains at least three voxels from $S^*$ (see Figures 5.15 (c) and 5.16). The edges for a polygonal face of $M'$ are enumerated by cycling through the voxels of the $2 \times 2 \times 2$ block. Since every voxel in $S^* \cap B$ has exactly two split-edges incident to the corner in the center of $B$, the ordering is given by split-edges that two neighboring voxels have in common. The following pseudo-code describes the procedure:

```
foreach $2 \times 2 \times 2$ block $B$ with at least three voxels in $S^*$ do
    pick a starting voxel $v$ from $S^* \cap B$
    pick a split-edge $e$ in $v$ adjacent to the center voxel corner $c$ of $B$
    repeat
        find the second split-edge $f$ in $v$ adjacent to $c$
        find the neighbor voxel $w$ from $S^* \cap B$ sharing split-edge $f$ with $v$
        generate a polygon edge from $v$ to $w$
        $v \leftarrow w, e \leftarrow f$
    until the first voxel is reached again
```

This algorithm works correctly because each voxel from $S^* \cap B$ has exactly two split-edges adjacent to the center corner and no more than two voxels share a common split-edge. A consistent orientation of the faces can be propagated by mesh traversal. The polygonal faces may be converted into triangle fans afterwards and decimated if required.

The resulting mesh is watertight but shows grid artifacts because the mesh vertices were initially placed at the voxel centers (see Figure 5.17). However, our surface confidence map $\phi$ computed for each voxel can be applied to the mesh vertices accordingly.
Figure 5.17: A mesh generated from a point cloud of a sphere. The upper sector shows the mesh extracted directly from the cut edges with voxel grid discretization artifacts. The lower sectors show the vertex distribution and reflection lines on the mesh after a number of smoothing steps. Since the meshing algorithm creates exactly one vertex per surface voxel the resulting mesh has a uniform vertex distribution and a quite regular mesh structure.

We can exploit this information for a confidence weighted smoothing algorithm, which allows for error bounded surface smoothing in confident surface areas, such that only the grid artifacts are removed, while less confident or noisy parts of the mesh can be smoothed stronger. We implement this algorithm by applying an iterative bi-Laplacian smoothing operator [DMSB99] for each vertex $v$:

$$v \leftarrow v - \frac{1}{d} \Delta^2 v, \quad d = 1 + \frac{1}{n_v} \sum_j n_{v,j}$$

with $n_v$ and $n_{v,j}$ being the valences of vertex $v$ and its $j$-th one-ring neighbor. The surface confidence values $\phi(v)$ prescribe how much every vertex is allowed to deviate from its original position during smoothing. We stop the movement of a vertex if $\delta p < \delta v (\phi(v) + 1)^s$ is violated. $\delta p$ is the difference between the original and the smoothed vertex position, $\delta v$ represents the voxel size, and $s$ allows for emphasized smoothing in inconfident regions. For all presented results, however, setting $s = 1$ proved to be sufficient.

This algorithm computes smooth output meshes while preserving the original surface approximation quality and the exact topology of the computed cut surface.
5.7 Reconstruction Results

Figure 5.18: Reconstruction of the Bahkauv statue from only 27 images. (a) The images were captured using a simple hand-held video camera on a roughly circular path. (b,c) Despite the specular surface and other illumination artifacts our method allows us to reconstruct 3D meshes of fairly high quality. (d) Some regions with a significant deviation from the Lambertian model in the input images lead to slightly noisy surface properties. (e) A textured and relighted version of the model.

5.7 Reconstruction Results

For the experimental evaluation we applied our algorithm to several self-generated as well as public data sets for multi-view stereo and point cloud reconstruction. All experiments were performed on a 3.2 GHz Pentium P4 with 2 Gb of main memory.

5.7.1 Multi-view Stereo

We captured video streams of the Warrior, the Leo, and the Dragon model with an uncalibrated turn-table setup and an image resolution of 1024 × 768. The Bahkauv statue was captured using a hand-held video camera with a resolution of 720 × 576. The
The Leo model (b) was reconstructed using 86 images captured with an uncalibrated turn-table setup (a). (c) Occluded parts of the model like the belly or the feet of the Leo are smoothly closed by the computed cut surface. (d) Textured and relighted versions of the model.

effective image size of the objects was often significantly smaller because of the necessity to include calibration objects within the images (see, e.g., Figure 5.19). The images were calibrated and segmented with standard techniques as those discussed in Section 2.1.1 and Section 3.1.

The Bahkauv statue shown in Figure 5.18 was reconstructed from only 27 images. Despite the difficult acquisition conditions with non-Lambertian, weakly textured surface properties and other illumination artifacts we achieve a quite acceptable reconstruction with less than 10 minutes overall computation time. Previous non-hierarchical approaches generally report reconstruction times in the range of one to several hours for comparable input complexity and reconstruction quality on similar reference systems.

For the reconstruction of the Leo model in Figure 5.19 we used 86 images. Although a quite acceptable reconstruction can already be obtained with about 30-40 images, we used a higher number of images to better reconstruct details such as the Leo’s ears. With less cameras they became slightly truncated due to their relatively small image size and slight errors in the camera calibration.
5.7 Reconstruction Results

Figure 5.20: The dragon model is particularly difficult to reconstruct because of the fine details such as the claws and the partially specular and transparent surface areas. We used 141 images (a) to increase the robustness of our technique and succeeded in reconstructing a fairly detailed model (b). However, our essentially Lambertian photo-consistency measure leads to visible artifacts, e.g., for the fire which is made from colored glass and hence causes complex light scattering effects. (c) A relighted and textured version of the model.

The Dragon model in Figure 5.20 represents a very complex reconstruction in terms of object structure and surface properties. We used 141 input images for increased robustness and started the reconstruction at a volumetric resolution of $256^3$ to preserve fine details such as the claws in the initial surface proxy.

Figure 5.21 shows a reconstruction of the Warrior model at a high resolution of $1024^3$. Despite the relatively low image resolution in comparison to the volumetric resolution our method is able to reconstruct fine surface details. For ground truth evaluation we computed the Hausdorff distance [ASCE02] of our reconstruction to a laser scan of the Warrior, resulting in a very low mean (max) error of 0.1% (1.9%) with respect to the bounding box diagonal.
5 High Quality Model Reconstruction

Figure 5.21: The warrior model was reconstructed from 71 images (a) at a high volumetric resolution of $1024^3$. (b) The resulting mesh has a complexity of 1570 K vertices. (c) The color coded reconstruction error with respect to a laser scanned reference model. The maximal errors occur at the concavities on the left arm and the hair.

Table 5.1 presents quantitative results of the performance and complexity of the reconstructed models, including the number of input images, timings, and the resulting mesh size. Our experiments show that the photo-consistency estimation is the dominating computational factor. This fact underlines the benefit of our hierarchical approach, since it significantly reduces the number of processed voxels and hence keeps the computation times acceptable even for high volumetric resolutions.

Additionally, our algorithm was evaluated with the “full” input data sets provided by the Middlebury MVS evaluation. Renderings of the reconstructed models and the numerical evaluation of the reconstruction correctness and surface completeness are shown in Figure 5.22. An in-depth explanation of these thresholds is provided in [SCD’06]. Our method achieves highly competitive results in terms of the surface quality and is, at the same time, among the fastest techniques evaluated in [Mid08a].
5.7 Reconstruction Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Bahkauv</th>
<th>Leo</th>
<th>Warrior</th>
<th>Warrior</th>
<th>Dragon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Images</td>
<td>27</td>
<td>86</td>
<td>71</td>
<td>71</td>
<td>141</td>
</tr>
<tr>
<td>Level $l_0$</td>
<td>7(128$^3$)</td>
<td>7(128$^3$)</td>
<td>7(128$^3$)</td>
<td>7(128$^3$)</td>
<td>8(256$^3$)</td>
</tr>
<tr>
<td>Target level</td>
<td>9(512$^3$)</td>
<td>9(512$^3$)</td>
<td>9(512$^3$)</td>
<td>10(1024$^3$)</td>
<td>9(512$^3$)</td>
</tr>
<tr>
<td>Visibility</td>
<td>2.9 s</td>
<td>6.5 s</td>
<td>5.9 s</td>
<td>5.9 s</td>
<td>35.9 s</td>
</tr>
<tr>
<td>Consistency</td>
<td>5.4 m</td>
<td>9.7 m</td>
<td>7.0 m</td>
<td>19.1 m</td>
<td>13.6 m</td>
</tr>
<tr>
<td>Graph cut</td>
<td>1.3 m</td>
<td>41.9 s</td>
<td>37.4 s</td>
<td>3.2 m</td>
<td>1.6 m</td>
</tr>
<tr>
<td>Meshing</td>
<td>45.0 s</td>
<td>17.5 s</td>
<td>25.5 s</td>
<td>2.6 m</td>
<td>1.2 m</td>
</tr>
<tr>
<td>Overall</td>
<td>9.5 m</td>
<td>12.8 m</td>
<td>10.5 m</td>
<td>27.8 m</td>
<td>20.4 m</td>
</tr>
<tr>
<td>Vertices</td>
<td>639 K</td>
<td>298 K</td>
<td>388 K</td>
<td>1570 K</td>
<td>575 K</td>
</tr>
</tbody>
</table>

Table 5.1: The time and space complexity of each reconstruction is mainly influenced by the number of input images, the initial level $l_0$, and the desired target resolution. This table provides accumulated timings in seconds and minutes respectively for each of the algorithm’s main phases. The overall reconstruction time additionally includes other involved processing steps such as the normal computation and the crust generation. The number of mesh vertices at the target resolution is given in the last row.

Figure 5.22: Middlebury results [Mid08a] for the Temple (a) and the Dino (b).
Figure 5.23: (a) Solid genus 0 reconstruction of a statue from non-uniformly sampled 3D points from raw laser scanned data. The point cloud contains a considerable number of outliers and holes. The backside of the upper arm and the lower part of the model are only partially sampled from the front without any samples at the back of the object. (b) The point cloud of the Max Planck consists of a set of circularly acquired laser scans. The images show that top and bottom of the bust as well as some smaller areas around the ears do not contain any samples. Our method closes these holes and produces a genus 0 mesh.

5.7.2 Point Clouds

We also applied our method to a variety of different point cloud data sets, e.g., acquired from laser scans and stereo vision based point reconstructions. All reconstructions are created only from the 3D sample positions without any normal information.

The statue shown in Figure 5.23 (a) is reconstructed from raw laser scanned data at a volumetric resolution of $1024^3$. The input point cloud contains significant noise, outliers and large gaps, e.g., at the bottom part or at the backside of the upper arm. Nevertheless our algorithm reconstructs a proper, watertight genus 0 model. This model is particularly difficult to reconstruct because of large regions with completely missing
5.7 Reconstruction Results

Figure 5.24: Solid reconstruction of a genus 3 object from a noisy scan. Due to significant noise, however, two of the rings are merged, resulting in a genus 4 reconstruction. Nevertheless, even significantly misaligned parts as shown in the right images are easily handled without producing topological artifacts.

samples on the backside of the statue. Further reconstructions from raw laser scanned point clouds are shown in Figures 5.23 (b) and 5.24. While the Max Planck example has a highly non-uniform sample distribution with large holes, especially at the top and the bottom, the Rings example contains significant noise and alignment artifacts.

In Figure 5.25 the point cloud for the Monkey (a) as well as for the Leo model (b) have been acquired by image-based 3D stereo reconstruction methods [SD97, HK07]. The Leo has a particularly non-uniform and noisy sample distribution with large clusters of points inside the model and larger gaps at the tail and the legs. The Monkey model has a more uniform sampling but consists only of samples for the front side of the face. The creation of watertight meshes from such models without at least approximate surface normals has been very challenging for previous methods.

Due to the fact that our method produces reconstructions of low genus without the typical small-scale topological artifacts often found in reconstructions from noisy signed distance fields, our method can also be employed for model repair. We applied our algorithm to the VRIP-reconstructions [CL96] of the Buddha (see Figure 5.14) and the Dragon (see Figure 5.26) available at the Stanford 3D Scanning Repository. Both models
5 High Quality Model Reconstruction

Figure 5.25: (a) The point cloud for the Monkey model was created by an image-based stereo reconstruction algorithm [HK07]. Despite the fact that only samples for the front of the model are available, our algorithm is capable of computing a closed mesh. We computed the interior component with the medial axis approximation described in Section 5.3.1. The ears, however, get cut away since they would include too many inconfident voxels on the back of the head, and the resulting surface energy would be higher than the given result. (b) The Leo point cloud was generated with a voxel-coloring algorithm for image-based 3D reconstruction [SD97]. It is quite noisy and highly non-uniform in terms of sample density and surface thickness. Nevertheless our graph cut approach reconstructs a proper genus 1 model.

have a very high genus due to topological artifacts. Our method faithfully reconstructs watertight models with low genus. For the Dragon model we also show the triangle quality generated by our meshing algorithm.

Quantitative evaluations in terms of the computation performance and the resulting complexity of the generated meshes are shown in Table 5.2.

5.7.3 Discussion

Our evaluation shows that our solution for surface reconstruction based on unsigned confidence maps and energy minimization using graph cuts is able to reconstruct 3D...
Figure 5.26: The original Dragon model in the Stanford 3D Scanning Repository contains numerous small holes and topological artifacts such as bridges, which results in a very high genus (> 400). When using the 3D point samples of this mesh as input to our algorithm, all topological artifacts are removed and the resulting watertight mesh has genus 1 (left images). The close-ups compare the original backside of one of the legs and a view from inside the model (upper right) to our result (lower right).

<table>
<thead>
<tr>
<th>Model</th>
<th>Resolution</th>
<th>Timings</th>
<th>Genus</th>
<th>Vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rings</td>
<td>256(^3)</td>
<td>45 s</td>
<td>4</td>
<td>91 K</td>
</tr>
<tr>
<td>Leo</td>
<td>256(^3)</td>
<td>48 s</td>
<td>1</td>
<td>47 K</td>
</tr>
<tr>
<td>Monkey</td>
<td>256(^3)</td>
<td>82 s</td>
<td>0</td>
<td>72 K</td>
</tr>
<tr>
<td>Buddha</td>
<td>512(^3)</td>
<td>112 s</td>
<td>10</td>
<td>264 K</td>
</tr>
<tr>
<td>Dragon</td>
<td>512(^3)</td>
<td>150 s</td>
<td>1</td>
<td>318 K</td>
</tr>
<tr>
<td>Max Planck</td>
<td>512(^3)</td>
<td>199 s</td>
<td>0</td>
<td>320 K</td>
</tr>
<tr>
<td>Statue</td>
<td>1024(^3)</td>
<td>269 s</td>
<td>0</td>
<td>448 K</td>
</tr>
</tbody>
</table>

Table 5.2: The time and space complexity for all presented reconstructions. The timings include all processing steps, from confidence estimation to mesh smoothing.
models of high quality at low computation times compared to other state-of-the-art
techniques (see [Mid08a] for a comparative evaluation). However, there remain a number
of open theoretical and practical questions as directions for future research.

Part of the energy functional optimized via graph cuts is the minimization of the
area of the reconstructed surface, which is required for achieving a regularizing force for
noisy or ambiguous estimates of the surface confidence map $\phi$. This generally results
in a certain surface shrinking bias which has been addressed in the literature in several
ways, e.g., by introducing a so-called ballooning term [VTC05]. This additional term for
surface inflation can be integrated in a straightforward manner to our graph definition.
In practice, however, the necessity for different regularization forces or more complex
energy functionals depends also strongly on the quality of the computed confidence
maps. Although our photo-consistency estimation presented in Section 4.2 produces
quite distinct optima of the confidence function, these topics should be investigated
further in future work.

Another topic are grid artifacts in the reconstructed surface due to the underlying
surface area metric induced by the graph connectivity. Because of our octahedral graph
structure, regions without any confidence information are reconstructed as planar hor-
izontal, vertical, or diagonal surfaces (e.g., Figure 5.23 (a)). However, it was shown in
previous work [BK03b] for graph-based segmentation that minimal surfaces and even
higher order smoothness constraints can be imposed by increasing the graph connectiv-
ity. Unfortunately, this higher connectivity causes increased computation times, often
without a significant increase in the reconstruction quality. In practice, these artifacts
can be effectively removed by our confidence weighted smoothing. Nevertheless, there
remain open questions concerning an optimal discretization of the surface energy func-
tional using graph cuts.

The current main limitation of our method to compute resolutions higher than $1024^3$
is the fact that we explicitly generate the octahedral graph structure within the voxel
grid, which leads to a noticeable memory overhead. This overhead could be reduced by
exploiting the special structure of our graph, e.g., by using some indexing scheme in the
voxel grid.
5.8 View Selection for Improved Multi-view Stereo

In the previous sections we discussed a general approach to 3D surface reconstruction by constructing a confidence map $\phi$. The question of creating such a function has been investigated intensively for input in the form of point clouds [PMG04]. But although a large body of work for computing $\phi$ from input images exists as well (see Section 4.2), there still remain a number of open issues.

For instance, so far we have assumed that all available input images should be considered for the model reconstruction process. However, for all types of image-based methods, the performance in terms of quality and efficiency generally depends considerably on the input data, i.e., the number and choice of images. On the one hand one needs enough measurements for computing $\phi$ reliably in order to create a faithful reconstruction of the 3D object. On the other hand, it is as well desirable to minimize the amount of data, since processing overly redundant input increases the overall computing time without improving (or even decreasing) the reconstruction quality. Moreover, special care has to be taken to resolve difficulties inherent to image-based reconstruction methods such as occlusions or complex surface materials.

These requirements pose a difficult challenge in the 3D reconstruction pipeline and often render a manual intervention by a human operator inevitable. Hence, especially in the field of active range imaging, there has been a lot of effort to automate this process [Pit99], generally known by the term “Next-Best-View Planning” (NBV). Central issues addressed in previous work are, for instance, the identification of occluded surface regions [MB93], the integration of prior model knowledge [BWDA00], or efficient range image quality evaluation by using the GPU [KS00a]. But although there have been many advances in this field, a lot of practically relevant problems are still considered unsolved [SRR03].

For MVS the dependency of the reconstruction on the input data is probably even more critical and many of the involved problems are inherently different to this previous work on NBV planning. In contrast to active reconstruction techniques, we cannot assume that one measurement, i.e., one input image, already generates a sufficiently good partial reconstruction. Instead, a fundamental requirement is that we need at least two input images for one single reconstruction step. In order to improve the robustness of the reconstruction process with respect to problems such as calibration errors, illumination changes, or image noise and blur, it is often necessary to increase the number of images.
Finally, a particular challenge are surface details and features like deep concavities or fine, topologically relevant structures such as holes (see, e.g., Figure 5.20).

In practice there are two common ways for acquiring input images for MVS reconstruction. The first possibility is a manual control of the image acquisition by the user, who can identify problematic regions of the object and hence choose the camera positions accordingly [RHHL02]. Current automatic acquisition setups are generally using a turn-table or a robot based system, and simply acquire several images of an object on regularly spaced camera positions. Neither of these approaches is guaranteed to capture all relevant parts of the object’s surface in a sufficient quality. Hence, one often captures an unnecessarily redundant set of input images (up to several hundred). However, while the acquisition of large numbers of images is easy, many recent algorithms still cannot process such a high number of images efficiently [Mid08a].

A few authors have developed dedicated selection schemes in the context of image-based reconstruction. A first set of view selection strategies for MVS has been presented in [FLB94]. Later selection strategies are based on contours [KD94], structure from motion [MC99], or Kalman filtering [WDH06]. Moreover, the problem of identifying optimal viewpoints for reconstruction has been addressed as well in related areas such as shape from silhouette and structured light [STK03], image-based rendering [VFSH03], and BRDF sampling [LLSS03]. The importance of a proper view selection for the quality and speed of MVS has also been emphasized in [GSC07]. This indicates that the question of image selection is a so far mostly untapped resource for optimization of current MVS techniques. However, the diverse foci of each of the above methods reflect also the inherent problem of selecting optimal input data with respect to the requirements of a specific technique.

In [HZK08] we presented an analysis of the typical requirements of MVS algorithms and an implementation of an efficient GPU-based algorithm of a corresponding image selection scheme. This scheme shows improved reconstruction quality and speed for different representative reconstruction techniques based on feature matching and patch expansion [FP07], surface growing [HK07], deformable models [HS04], and the volumetric graph cut technique described in the previous sections.
5.8 View Selection for Improved Multi-view Stereo

5.8.1 Multi-view Stereo Requirements

Given a (possibly very large) set of calibrated input images of an object, we aim at automatically selecting a subset of images which sufficiently capture all relevant features without adding too much redundancy.

Finding an optimal image subset is a complex combinatorial optimization problem. Hence, a common approach in the field of NBV planning for range imaging is the use of iterative greedy procedures, which are generally based on the following generic work cycle [SRR03]: First, the algorithm takes one or more initial measurements (scans), and generates a corresponding geometric proxy. This proxy is then iteratively refined by the optimization procedure. At the beginning of each iteration, different quality measures are computed over the current surface approximation. These measures describe, for example, the current coverage or measurement certainty. Based on this information the algorithm selects new views for which one expects a maximal quality gain and updates the surface proxy with this new information. These steps are repeated until a termination criterion with respect to the quality measures is met.

For an algorithm supporting MVS, we have to identify the corresponding specific requirements. For recent techniques such as the ones mentioned in the previous Section 5.8 one finds that the reconstruction accuracy and efficiency depends largely on the following three criteria:

**Initial Surface Proxy:** Most algorithms either require or iteratively generate a proxy of the object as an initialization, e.g., for a proper topology and visibility estimation (see Section 5.3.1). Furthermore, a faithful proxy minimizes the computation time, since methods based on deformable models evolve the initial proxy to the actual object surface [HS04]. Volumetric approaches perform better the more voxels are carved away which are not part of the true surface. With a few exceptions [HVC07], this geometric prior is generally based on segmented input images (i.e., the visual hull), since efficiently computing a more accurate stereo-based proxy from a large set of input images is a non-trivial problem in practice. Hence, the first important goal of our method is the selection of a small subset of input images, which allows for an efficient generation of a stereo-based proxy that is a good approximation of the unknown object surface and which is sufficiently covered by the selected images.
Surface Visibility: For MVS reconstruction, every surface point has to be visible in at least two images. In general, however, the reconstruction quality is strongly affected by texture, image noise and blur, calibration errors, or illumination effects which are not handled by the used photo-consistency metric. On the one hand, these problems can be alleviated by capturing redundant data. On the other hand, unnecessary redundancy increases the processing time. Hence, an algorithm for image selection should ensure that each surface point is visible for a certain number of cameras with a guaranteed minimum viewing angle and without including unnecessary images. Although this criterion seems similar to the previous requirement, the essential difference is that a visibility optimization without a proper initial proxy would potentially lead to a suboptimal selection of images which focus on incorrectly approximated or even nonexistent surface parts.

Adaptivity: While the above steps guarantee a minimum viewing quality for every surface point on the proxy, they do not ensure a good reconstruction performance in particularly difficult surface regions, for which the proxy is only a suboptimal approximation to the real object surface. Typically, this can happen for deep concavities or thin holes through the object, which are difficult to detect and capture properly. As a consequence, methods requiring an initially correct topology of the proxy fail. However, because of the distance of these proxy regions to the true object surface, they can often be characterized by having bad photo-consistency values in the input images. Hence, our algorithm should adapt to the surface reliability by selecting additional images focusing on photo-inconsistent regions.

5.8.2 Image Selection and Proxy Generation

The above criteria exhibit a natural successive order since each of them relies on the respective previous criterion. So instead of simultaneously optimizing all criteria, these dependencies allow for an efficient iterative optimization procedure consisting of three corresponding phases: Phase 1 aims for choosing views that support a fast convergence towards an initial geometric proxy. Phase 2 then ensures a sufficient coverage of each point on the proxy surface in at least 2 images, and phase 3 adds additional images focusing on proxy regions with locally bad photo-consistency values.

In each phase, a corresponding quality criterion has to be evaluated on every point of the current surface approximation. Based on this evaluation, the algorithm selects a
new image which is expected to maximally improve this quality criterion. The underly-
ing proxy representation based on a voxel grid $V$ is identical to the one introduced in Section 5.2. In our experiments a medium grid resolution of $128^3$ showed the best tradeoff in terms of accuracy and efficiency. In phase 1, the algorithm classifies all voxels $v \in V$ as empty, which can be identified as not being part of the object. The remaining full voxels $S_i \subset \cdots \subset S_0 = V$ represent the iteratively improved object proxy. Full voxels with empty neighbors lie on the current proxy surface and are denoted by $\partial S_i$.

Surface normals are estimated as in Section 5.3.1 by fitting a regression plane to the local neighborhood of surface voxels.

Depending on the image acquisition setup, the available images can be distributed in the whole embedding space or in some arbitrary sub-region, e.g., with viewpoints constrained to lie on a sphere around the object. Our method is not limited to any specific configuration but can handle arbitrary viewpoints and -directions. In the following the set of camera positions corresponding to the set of images is denoted as $C$.

**Selection Procedure**

At the beginning of the $i$th step of the algorithm, a proxy $\partial S_{i-1}$ has been constructed from the views $I_1, \ldots, I_{i-1}$ seen from viewpoints $c_1, \ldots, c_{i-1} \in C$. Our goal is to pick a new viewpoint $c_i$ such that adding the corresponding view $I_i$ leads to a maximally improved proxy $\partial S_i$ with respect to the quality criterion of the current phase.

The major problem with this approach is that one cannot predict $\partial S_i$ without actually knowing the new view $I_i$ and integrating it into the current reconstruction. However, for a large number of views, such a tentative integration and evaluation of all possible images $I_i$ is computationally infeasible. Hence, the best we can do is to rate the improvement that the new view would have on the old proxy $\partial S_{i-1}$. With this approach, all quality criteria can be formulated in terms of the viewpoints $c_i$ only. This allows us to estimate the quality gain for a given image $I_i$ efficiently by rendering the current proxy $\partial S_{i-1}$ as seen from the respective viewpoint $c_i$, without having to recompute and evaluate the proxy for every image. The next best view is the image which shows the maximal number of low quality voxels.

Each phase continues as long as the quality gain per iteration stays above a certain threshold. Otherwise the algorithm switches to the next phase by changing the quality
5 High Quality Model Reconstruction

Figure 5.27: Illustration of phase 1. (a) For each image the algorithm computes a depth map (b) using our small-baseline stereo method (see Section 4.3) and uses the carving approach described in Section 5.3.1 to create an updated model proxy (c). (d) For the remaining voxels it then estimates a quality value depending, e.g., on their visibility in the images selected so far. The color coding visualizes the current state of the proxy (shown from a different viewpoint in (e) for illustration purposes). Visible voxels are marked blue, while voxels currently not visible in any image are marked red.

criterion. The following describes the phases and criteria in detail. Section 5.8.3 then shows how these criteria can be evaluated efficiently.

Phase 1: Initial Surface Proxy The first phase aims for guaranteeing that every voxel \( v \in \partial S_{i-1} \) of the current proxy is visible in at least one image \( I_j \) from an acute viewing angle \( \leq \alpha \) (see Figure 5.27) in order to reliably classify it as being inside or outside of the object’s photo-hull. The viewing angle is the angle between the surface normal \( n \) of a voxel \( v \) and the vector \( d_j = (c_j - v) / \| c_j - v \| \) pointing from \( v \) to a camera center \( c_j \).

This requirement can be formalized as

\[
\forall v \in \partial S_{i-1} \exists j \in [1, i] : P_{\alpha}(v, c_j) \text{ with } P_{\alpha}(v, c_j) : \text{visible}(v, c_j) \wedge d_j^T \cdot n \geq \cos \alpha .
\] (5.6)

The sets of low quality voxels \( \partial S_{i-1}, \partial S'_{i-1} \subseteq \partial S_{i-1} \), which violate this condition before and after the integration of view \( c_i \), are defined as

\[
\partial S'_{i-1} = \{ v \in \partial S_{i-1} : \forall j \in [1, i - 1] : \neg P_{\alpha}(v, c_j) \} \text{ and }
\] (5.7)

\[
\partial S''_{i-1} = \{ v \in \partial S_{i-1} : \forall j \in [1, i] : \neg P_{\alpha}(v, c_j) \} .
\] (5.8)

130
5.8 View Selection for Improved Multi-view Stereo

Figure 5.28: With each iteration, the algorithm selects a new image which maximizes the number of visible voxels and then iterates steps (b)-(d) shown in Figure 5.27.

The quality gain $g_i$ in the $i$th iteration can then be defined as the relative improvement of low quality voxels, i.e.,

$$g_i(c_i) = (\# \tilde{\partial}S_{i-1} - \# \tilde{\partial}S'_{i-1}) / \# \tilde{\partial}S_{i-1}. \quad (5.9)$$

The free parameter to maximize $g_i(c_i)$ (i.e., to minimize $\# \tilde{\partial}S'_{i-1}$) is the next viewpoint $c_i$ among all the candidates in $C$.

Maximizing this expression directly would correspond to counting the number of improved voxels, without taking the actual degree of improvement for each voxel into account. Hence, we use the weighted improvement of the viewing direction with respect to all previous views ($d^T_i \cdot n - \max_{j \leq i-1} (d^T_j \cdot n)$) in order to increase the robustness and sensitivity of the algorithm. This allows the image selection to focus on proxy regions with the lowest quality first. Furthermore, by taking the minimum of $d^T_i \cdot n$ and $\cos \alpha$, this weighted approach does not reward improvements beyond the angle threshold $\alpha$, which implicitly promotes a sufficient parallax between the input images. The complete functional $g'_i(c_i)$ for the quality gain of a viewpoint $c_i$ then is

$$g'_i(c_i) = \sum_{v \in \tilde{\partial}S_{i-1}} g'_i(v, c_i),$$

with

$$g'_i(v, c_i) = \begin{cases} 
\min(d^T_i \cdot n, \cos \alpha) - \max_{j \leq i-1} (d^T_j \cdot n) & \text{if } P_\alpha(v, c_i) \nonumber \\
0 & \text{else} \end{cases}. \quad (5.11)$$

If the viewpoint maximizing $g'_i(c_i)$ actually leads to an effective improvement $g_i(c_i)$ above a prescribed threshold $\delta$, the algorithm adds $I_i$ to the set of images.
High Quality Model Reconstruction

Then the proxy has to be updated $S_{i-1} \rightarrow S_i$ by identifying all voxels, which are outside of the photo-hull as seen from this new view $I_i$. We achieve this by using the same voxel carving approach as described in Section 5.3.1. As comparison images, the algorithm simply selects 2 images from the whole set of available images, which are closest to $I_i$ and which have the most similar viewing direction. Silhouettes can optionally be exploited.

Phase 1 terminates if either the percentage of low quality voxels drops under a threshold $\tilde{\partial} S_{i-1}/\tilde{\partial} S_{i-1} < \epsilon$, or if the improvement of low quality voxels measured by $g_i(c_i)$ is less than $\delta$. Otherwise, the algorithm continues with iteration $i+1$ (see Figure 5.28).

Low quality surface voxels left in $\tilde{\partial} S_i$ at the end of phase 1 which have never been visible in any of the images, e.g., the bottom of the Middlebury Dino model (see Figure 5.31), are excluded from further processing. The result of this phase is a sequence of images, which supports a fast convergence towards a sufficiently covered, faithful approximation of the unknown object.

Phase 2: Surface Visibility

After the initial proxy generation in phase 1 is accomplished, the quality criterion is changed and additional images are added such that each voxel now becomes visible in a user specified number $\kappa \geq 2$ of images with a guaranteed maximal viewing angle $\leq \alpha$. The possibility of enforcing visibility of each voxel in more than 2 images generally helps to increase the robustness of the subsequent MVS reconstruction process. The corresponding requirement can be expressed similar to Equation 5.6:

$$\forall v \in \tilde{\partial} S_{i-1} : Q(v), \text{ with } Q(v) : \# \{ j \in [1, i] : P_\alpha(v, c_j) \} \geq \kappa ,$$

(5.12)

with the low quality voxels $\tilde{\partial} S_{i-1}$ analogously defined as

$$\tilde{\partial} S_{i-1} = \{ v \in \partial S_{i-1} : -Q(v) \} .$$

(5.13)

The computation of the quality gain $g_i(c_i)$ and the termination criterion is identical to phase 1. To promote a more uniform distribution of viewpoints, phase 2 is started with a variable $\kappa' = 2$. Each time the termination criterion is met, $\kappa'$ is increased by one until $\kappa' = \kappa$. 

132
5.8 View Selection for Improved Multi-view Stereo

For the selection of the next view \( c_i \), we again adopt a weighted approach similar to phase 1, which takes the visibility improvement for each surface voxel into account, i.e., a voxel which is visible in only one other view \( I_1, \ldots, I_{i-1} \) counts more than a voxel which is already visible in several other views. The corresponding functional is defined analogous to Equation 5.10, with a different quality gain \( g'_i(v, c_i) \)

\[
g'_i(v, c_i) = \begin{cases} 
1 - \frac{m(v)}{\kappa} & \text{if } P_{\alpha}(v, c_i) > 0 \\
0 & \text{else} 
\end{cases} ,
\]

(5.14)

where \( m(v) = \# \{ j \in [1, i-1] : P_{\alpha}(v, c_j) \} \) is the number of views among \( I_1, \ldots, I_{i-1} \) in which \( v \) is sufficiently visible.

Phase 3: Adaptivity  The final phase supports the reconstruction of problematic or topologically important surface regions such as concavities or holes. These regions can often be identified by their bad photo-consistency because of a significant deviation from the true surface or because of deficiencies of the consistency metric. We found that the reconstruction quality can be considerably improved by integrating additional images focusing on these regions.

Hence, we compute for each voxel \( v \in \partial S_{i-1} \) a consistency value \( \phi(v) \) using a standard metric based on, e.g., normalized cross-correlation or color variances. These metrics are widely used among recent MVS methods and therefore are a reasonable choice for addressing consistency problems [GSC*07]. We employ the robust consistency estimation based on voxel supersampling introduced in Section 4.2.1. Large values of \( \phi(v) \) correspond to a high color variance and hence represent a bad photo-consistency. So for each voxel having a value \( \phi(v) \) larger than a consistency threshold \( \psi \), the algorithm should guarantee \( \tau \) additional views from a viewing angle \( \beta < \alpha \):

\[
\forall v \in \partial S_{i-1} : R(v), \text{ with } R(v) : \phi(v) < \psi \lor \# \{ j \in [1, i] : P_{\beta}(v, c_j) \} \geq \tau .
\]

(5.15)

The number of additional views \( \tau \) and the angle \( \beta \) have an equivalent meaning to \( \kappa \) and \( \alpha \) in phase 2. However, our experiments showed that \( \tau \) can be chosen smaller than \( \kappa \) since one generally needs only a few extra images to improve the reconstruction in problematic regions. \( \psi \) obviously depends on the method for measuring photo-consistency. In our implementation we found these parameters to work quite stable for different data sets.
Thus, we could simply keep them constantly set to \( \tau = 2, \beta = 30, \) and \( \psi = 0.7. \) Low quality voxels \( \partial S_{i-1} \) are defined as before

\[
\widetilde{\partial S}_{i-1} = \{ v \in \partial S_{i-1} : \neg R(v) \},
\]

and the quality gain \( g_i(c_i) \) and termination criteria are again analogous to phase 1. As in the previous phases, we compute a weighted estimate based on the photo-consistency values for selecting an image:

\[
g'_i(v, c_i) = \begin{cases} 
\phi(v) & \text{if } P_i(v, c_i) \\
0 & \text{else}
\end{cases}
\]

### 5.8.3 GPU-based Implementation

Section 4.5 already described efficient GPU-based implementations for small baseline stereo and photo-consistency estimation. The remaining most time consuming part of this algorithm is the evaluation of the quality criteria in all 3 phases. Practically useful computation times can only be achieved if we manage to evaluate each input image in just a few milliseconds. Remember that a quality estimate for each single image requires the following steps: (1) check every surface voxel for visibility, (2) estimate the quality gain per voxel (based on the viewing angle or photo-consistency), and (3) accumulate the quality gain over all voxels to estimate the total gain. In order to achieve the required efficiency, we transfer the computation of these steps to the GPU as well.

The GPU implementation consists of two main passes. First, we have to determine the visibility of all surface voxels for a given image \( I_i. \) Here, we can apply the same technique as described in Section 5.3.1. Next, we transfer the local quality gain estimation to the GPU by computing the corresponding equations (e.g., Equation 5.11) in a fragment shader [SA06], and encoding the results in the rendered splat color. With the support of recent graphics processors for floating point output buffers we achieve the same computational accuracy as in a CPU based implementation. Unfortunately, summing up frame buffer pixels for accumulating color-encoded quality gain values and counting the number of improved voxels is not an efficient operation on today's GPUs. We can, however, exploit the color blending functionality to perform the quality gain accumulation. Instead of rendering each voxel to its projected 2D position in the input image, we define a frame buffer of size 1 \times 1 and render the required values of each...
5.8 View Selection for Improved Multi-view Stereo

<table>
<thead>
<tr>
<th>Model</th>
<th>( \alpha )</th>
<th>Images</th>
<th>Error UNI</th>
<th>Error SEL</th>
<th>Rel. Improv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mouse</td>
<td>45</td>
<td>27</td>
<td>0.35 (4.46)</td>
<td>0.24 (2.82)</td>
<td>30% (37%)</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>44</td>
<td>0.33 (4.58)</td>
<td>0.24 (2.72)</td>
<td>27% (41%)</td>
</tr>
<tr>
<td>CAD</td>
<td>60</td>
<td>23</td>
<td>0.99 (4.81)</td>
<td>0.44 (3.77)</td>
<td>55% (22%)</td>
</tr>
<tr>
<td>Scarecrow</td>
<td>45</td>
<td>24</td>
<td>0.62 (5.93)</td>
<td>0.35 (3.84)</td>
<td>44% (35%)</td>
</tr>
<tr>
<td>Bahkauv</td>
<td>45</td>
<td>19</td>
<td>1.65 (7.52)</td>
<td>0.75 (5.89)</td>
<td>55% (22%)</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>26</td>
<td>0.92 (6.33)</td>
<td>0.67 (5.95)</td>
<td>27% (6%)</td>
</tr>
<tr>
<td>Temple</td>
<td>60</td>
<td>21</td>
<td>0.60 (3.55)</td>
<td>0.52 (3.20)</td>
<td>13% (10%)</td>
</tr>
<tr>
<td></td>
<td>45</td>
<td>50</td>
<td>0.50 (4.28)</td>
<td>0.42 (2.58)</td>
<td>16% (40%)</td>
</tr>
<tr>
<td>Dino</td>
<td>45</td>
<td>41</td>
<td>0.56 (4.37)</td>
<td>0.47 (3.73)</td>
<td>16% (15%)</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>50</td>
<td>0.53 (4.52)</td>
<td>0.45 (3.28)</td>
<td>15% (27%)</td>
</tr>
</tbody>
</table>

Table 5.3: Evaluation of the view selection with our MVS approach described in this chapter for several data sets and parameter settings. This table shows the RMS & (MAX) Hausdorff distance to the respective reference model.

visible voxel into this single pixel. By configuring the rendering pipeline to perform additive blending of the output colors, we achieve the desired accumulation over the proxy surface.

5.8.4 Evaluation

Our algorithm for image selection was evaluated with four representative classes of MVS techniques participating in the Middlebury evaluation [Mid08a] and showed considerable improvements for all techniques in terms of quality and efficiency. Table 5.3 shows results for several experiments on synthetic and real input data with our MVS approach described in this chapter. For each data set, we created a reference model from all available images and then compared reconstructions with different parameter settings from images selected by our algorithm (SEL) vs. uniformly distributed images (UNI).

The color codings in Figures 5.29 to 5.31 visualize the approximation error with respect to the reference model.

In the synthetic experiments in Figure 5.29 we investigated the performance of the algorithm for different types of features, such as concavities or thin holes. We generated 800 images uniformly distributed around a laser scanned 3D mesh. Since the
Figure 5.29: (a) In the synthetic Mouse experiment, only the selected image set consistently reproduces the deep concavities in the cheese. (b) For the CAD model, the uniform images as well as phases 1&2 fail to capture the thin holes in the ring. Phase 3 reveals inconsistent voxels in these proxy regions (green) and selects corresponding images with a significantly improved result.

Figure 5.30: The complex Scarecrow (a) and the Bahkauv (b) model show significant improvements, in particular for difficult features like the Scarecrow’s hat.

photo-consistency metric (Phase 3) is based on color variances, we simulated a non-trivial consistency estimation by rendering each model with a white, textureless surface illuminated by a few light sources and set parameter $\kappa = 2$ due to the perfect image calibration and noise-free images.

The experiments with real data were performed with the Middlebury Temple and Dino (see Figure 5.31) data set (>300 images each), with 150 images of a Scarecrow model captured using a turntable, and with 290 images of the Bahkauv statue captured with a hand-held camera (see Figure 5.30). For the latter two models the reference
View Selection for Improved Multi-view Stereo

5.8 View Selection for Improved Multi-view Stereo

![Images of Temple and Dino models]

Figure 5.31: (a) The Temple model shows a few improvements, e.g., at the backside. (b) For the Dino model our algorithm selects mainly side views and only a small number of top views (see also Figure 5.32), and hence captures the head region and the concavities between the legs much better.

reconstructions were generated using all available images. To compensate for calibration errors and other problems like image noise we set $\kappa = 3$ for these experiments.

We then measured the RMS and maximal symmetric Hausdorff distance of the SEL and UNI models to the respective reference model using [ASCE02]. Table 5.3 shows that the reconstruction error is consistently lower in the case of selected images. Although the numerical improvement sometimes seems relatively small, the visual improvement of the overall shape is often significant (see Figure 5.30). Especially in cases with a relatively small number of images for complex surfaces, the results are significantly better, e.g., the Scarecrow, the Bahkauv, or the CAD model. Reconstructions using selected images even perform better than the uniform data with considerably more images, e.g., UNI Dino, 50 images: 0.53 RMS / 4.52 MAX vs. SEL Dino, 41 images: 0.47 RMS / 3.73 MAX (see also Mouse, Temple, or Bahkauv).

Table 5.4 presents the Middlebury results [Mid08a] for four MVS approaches participating in this evaluation, each representing a fundamentally different class of techniques: feature matching and patch expansion [FP07], surface growing [HK07], deformable models [HS04], and our volumetric graph cuts. We used the 41 selected ($\alpha = 45$) and uniformly distributed images of the Dino as input images for each method, since this model is generally considered a difficult example because of the missing texture. The parameter settings used by the corresponding authors were not identical to the ones used in their own Middlebury submission, so that the reconstruction quality might differ a bit. However, the parameter settings used for each technique were identical for the selected
and uniform image distributions. The first three rows of Table 5.4 show the quantitative results of uniform vs. selected views (UNI / SEL) for different accuracy thresholds. The last three rows show the results in terms of completeness (see [SCD06]). Again, our selected images consistently produce better results for all methods and thresholds. Figure 5.32 shows the selected image set for the Dino model. In contrast to the uniform set, our selection scheme chose an increased number of side views of the Dino and less images from the top, which helps to improve the head and the concavities around the legs considerably.

The processing time of our algorithm depends on the number of input images and iterations, the proxy resolution, and the number of low quality voxels. However, even for relatively high numbers of images as in the experiments with synthetic data (800), the algorithm needs only 1 to 15 seconds (4 seconds on average) for a single iteration. The quality gain for a single input image is evaluated in about 3 to 20 ms. For all experiments the overall processing time never took more than 1 to 4 minutes, which is negligible compared to the total runtime of most MVS reconstruction algorithms [Mid08a]. For instance, using the view selection algorithm in combination with our graph cut based reconstruction technique, the UNI Temple with 50 images took 15 min. to compute, while the SEL result with 21 images took only 7 min. (including the image selection) with similar reconstruction errors. The presented experiments and measurements were performed on a P4 2.8 GHz with a GeForce 6800 Ultra GPU. Our results (e.g., Table 5.3) show that reconstructions based on images selected by our algorithm usually have an even higher quality than reconstructions from non-optimized input with up to twice as many images. Considering the fact that the running-time of MVS algorithms is generally

<table>
<thead>
<tr>
<th>Thresholds</th>
<th>Matching</th>
<th>Growing</th>
<th>Deformable</th>
<th>Graph Cuts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[FP07]</td>
<td>[HK07]</td>
<td>[HS04]</td>
<td>[HK06a]</td>
</tr>
<tr>
<td>80% (mm)</td>
<td>0.43 / 0.41</td>
<td>0.52 / 0.49</td>
<td>0.36 / 0.33</td>
<td>0.64 / 0.59</td>
</tr>
<tr>
<td>90% (mm)</td>
<td>0.60 / 0.56</td>
<td>0.90 / 0.66</td>
<td>0.50 / 0.45</td>
<td>1.00 / 0.88</td>
</tr>
<tr>
<td>99% (mm)</td>
<td>1.36 / 1.31</td>
<td>1.38 / 1.27</td>
<td>1.11 / 0.83</td>
<td>2.86 / 2.08</td>
</tr>
<tr>
<td>0.75 mm (%)</td>
<td>92.1 / 93.2</td>
<td>81.5 / 85.2</td>
<td>95.5 / 97.4</td>
<td>79.5 / 82.9</td>
</tr>
<tr>
<td>1.25 mm (%)</td>
<td>97.8 / 97.8</td>
<td>92.3 / 94.2</td>
<td>99.0 / 99.4</td>
<td>90.2 / 93.0</td>
</tr>
<tr>
<td>1.75 mm (%)</td>
<td>99.2 / 99.3</td>
<td>93.8 / 97.3</td>
<td>99.8 / 99.6</td>
<td>94.3 / 96.9</td>
</tr>
</tbody>
</table>

Table 5.4: Middlebury evaluation [Mid08a] for four different MVS approaches with 41 uniform / selected (α = 45) images of the Dino.
5.8 View Selection for Improved Multi-view Stereo

(a) Full set of input images.

(b) Uniformly distributed images.

(c) Images selected by our method.

**Figure 5.32:** Side and bottom view of the full image data for the Dino model, a uniform selection of images, and the image set selected by our method.

The numerical and visual evaluation shows that proper image selection is an important, yet currently insufficiently considered resource of optimization in MVS reconstruction. Similar observations have been made in [GSC’07]. Our automatic image selection
dominated by the number of input images, this property helps to considerably reduce processing times, while achieving the same or better output quality.
5 High Quality Model Reconstruction

provides a basis for next-best-view planning in the context of multi-view stereo for an increased flexibility and automation. Similar to the previous chapters, the key to the efficiency of this method was the actual proxy representation which allowed for a fast evaluation of the quality gain using a splat-based approach. In future work we would like to investigate extensions to our method such as an explicit evaluation and handling of calibration errors, additional entropy based image quality measures, or view selection based on robust statistics [VHTC07]. Moreover, techniques based on photometric stereo [HVC08] obviously have requirements different from the standard MVS setting. We believe that investigating image selection for a wider range of techniques such as the free viewpoint rendering in Chapter 4 is an interesting direction for future work as well.
6 Conclusion

We conclude this dissertation thesis with a discussion of our contributions and an elaboration of future research perspectives.

Discussion of Contributions

Image- and video-based reconstruction and rendering has received a considerable amount of attention in recent computer vision and graphics research. The potential and importance of this field is reflected in the variety of novel technologies which have enabled entirely new applications for photo-realistic image synthesis or accurate surface reconstruction. From this diversity emerged our motivation to investigate three representative problems from this spectrum of image-based techniques, with a main focus on the development of new shape representations and corresponding optimization techniques.

In Chapter 3 we have proposed a deformable, template-based shape representation which is specifically adapted to the problem of reconstructing and animating articulated characters from images. The central idea was to fit this general 3D template model to a character’s shape in the input images by using prerecorded human motion data, a character-based camera estimation, and an as-rigid-as-possible shape deformation technique. For the realistic animation of single images we have extended the as-rigid-as-possible deformation technique to correctly handle perspective distortions of 2D character shapes with simulated depth. In the case of multiple input views, our occlusion- and silhouette-aware mesh tracking algorithm allows for the computation of a dense correspondence field throughout several image sequences. We have introduced the new idea of pose synchronization which enables the transformation of uncalibrated image sequences of dynamically moving persons into a synchronized multi-view setup for refining the 3D shape of the template model. As results we presented a variety of different character animations from single images to full body reconstructions and animations from video.
6 Conclusion

In comparison to previous work our generic base template results in a significantly increased flexibility, which allows us to process a variety of different input types in a unified character reconstruction and animation framework. Our method is scalable in the sense that the quality and completeness of a character reconstruction naturally scales with the number and quality of the input images. However, the increased flexibility also induces a lower reconstruction quality when compared to perfectly calibrated and controlled setups with multiple cameras. Nevertheless, we have shown that our method is capable of producing plausible animations from images with a minimal set of requirements and that it represents a reasonable extension to the existing spectrum of more constrained image-based character reconstruction and animation techniques.

Chapter 4 then focused on the problem of rendering novel views of general scenes from multiple synchronized input images. Our particle-based geometry representation supports an accurate handling of object silhouettes and occlusions in arbitrary scenes by using anisotropic particle shapes. We have proposed a new approach to photo-consistency estimation of these particles by volumetric supersampling and projection, which addresses the common problem of misaligned image comparison windows for unconstrained camera configurations and unknown surface geometry. The key elements for efficiently generating accurate input view proxies are our logarithmic view-space parameterization and the continuous optimization and filtering procedure. Our proxy generation enables a faithful approximation of curved and oblique surfaces and hence supports a more accurate view synthesis than previous techniques using discrete depth labels.

For the synthesis of new views we have shown how multiple geometry proxies can be combined in an outlier robust way into a single view-dependent output proxy. This proxy supports the computation of a pixel-accurate blending field for merging color contributions of all relevant input views. By decoupling the proxy resolution from the output view sampling, our blending field generation allows us to reproduce the visual appearance of complex shapes and anisotropic surface reflectance properties even at relatively coarse input proxy resolutions. For the computationally intensive and time critical tasks, such as photo-consistency estimation or view synthesis, we have proposed data structures and algorithms which are suitable for exploiting the parallel processing power of current graphics processors. A particular advantage of our approach is its support for real-time, unconstrained user navigation due to our ability to synthesize high
quality output views fully on the GPU and for camera parameters which significantly
differ from the original input images.

In Chapter 5, we have turned our attention to a unified approach for accurate and
consistent surface reconstruction from image-based data. Here, our key contribution
has been the introduction of a new surface representation based on a volumetric con-
fidence map. This unsigned scalar function increases the robustness to degenerate and
noisy data considerably in comparison to more conventional approaches based on signed
distance fields. We have shown how the desired surface can be characterized by an en-
ergy functional for which we can find the globally optimal solution by computing the
minimum cut of an embedded graph structure. Our novel octahedral structure of this
graph provides a proper mapping of the surface confidence values to the graph edges.
Moreover, by employing a hierarchical approach, we are able to achieve a high computa-
tional efficiency as well as improved reconstruction results by an iterative refinement
of the confidence map. The graph cut surface can then be converted directly into a
consistent explicit mesh for further processing. We have presented several high quality
reconstructions from images and from point clouds without the requirement of accurate
surface normals. Our results showed to be internationally very competitive with respect
to the reconstruction quality as well as the algorithm’s performance [Mid08a].

In the second part of Chapter 5, we have introduced the concept of next-best-view
planning to the field of multi-view stereo reconstruction. Based on an analysis of multi-
view stereo requirements we have developed an algorithm for image selection which
provides a better object coverage and an improved focus on problematic surface regions
than the standard uniform view distributions. Furthermore, we demonstrated the practi-
cal benefit of our algorithm by an evaluation of four different classes of multi-view stereo
techniques which belong to the current top performers in the Middlebury evaluation
[Mid08a]. Since our method was able to consistently improve the reconstruction qual-
ity and performance of all evaluated techniques, it can be considered as a new general
resource for the optimization of multi-view stereo reconstruction.

From a more global point of view, one important aspect of our work has been the design
of data structures that support a generalized and more unified approach to problems in
image-based reconstruction and rendering, which were previously considered separately.
Particular examples are the integrated approach to 2D and 3D character animation pre-
sented in Chapter 3 using a single template mesh and the unified method for surface
reconstruction based on confidence maps and graph-based surface extraction presented

143
Conclusion

in Chapter 5. Finally, for each of the three investigated topics, we have presented all components required for implementing the complete processing pipeline introduced in Chapter 2. By providing data structures and algorithms with a well-balanced consideration of quality, versatility, and efficiency we have tried to facilitate the use of these methods with an actual benefit for practical systems.

Perspectives for Future Research

Several ideas for specific directions of future work have already been pointed out in the respective discussions of the previous chapters. For instance, for the problem of character reconstruction and animation, it would be desirable to automate the camera and pose estimation as well as the silhouette fitting at least for those images that show a human person. Current research on object detection and segmentation [LLS08] and human pose estimation [AT06, GEJ08] has made considerable progress, and it would be interesting to evaluate these techniques in the context of our work. However, also for these techniques it is still difficult to compete with the accuracy of controlled multi-view setups [KBV05, BSB07]. In this regard, it would also be interesting to investigate how those techniques could be extended to handle more uncommon poses and characters that differ significantly from an average human shape (as some of the characters in Chapter 3).

An advantage of our proposed rendering technique for 2D animations from single images over our 3D character rendering is the preservation of complex character silhouettes by using alpha matting. Although we support view-dependent textures during the 3D character animation, the employed standard rendering pipeline for triangle meshes does not easily allow for preserving more complex or transparent materials such as hair. Support for rendering these types of materials (as, e.g., in [MPN02]) as well as the integration of more sophisticated shape deformation techniques [BPWG07] could be interesting extensions to our current approach.

The free viewpoint rendering technique would also benefit from an explicit handling of alpha boundaries [SG98]. Currently, a single depth value is computed for each particle. Image pixels at object boundaries, however, often integrate light from the foreground and background simultaneously. Moreover, the quality of the input view proxies could be improved by integrating the non-planar surface sampling proposed in Chapter 5 as a subsequent proxy refinement phase. Another interesting topic for future work would be
the extension of our purely image-based approach with additional synthetic illumination models, e.g., for relighting a rendered scene. However, the next essential step for supporting free viewpoint synthesis for multiple video streams [MP04, WWCG07] instead of images would be the extension of our method to spatially and temporally coherent video processing, e.g., during the proxy filtering or the output view generation.

Our surface reconstruction from confidence maps using graph cuts provides a number of opportunities for future work as well. For example, our method’s support for surface reconstruction from unoriented points is an important property to handle noisy input data. However, the integration of approximate information about the surface orientation could help impose constraints on the object topology (as for the Rings example in Figure 5.24). Our banded graph cuts based on an iteratively refined crust may lose surface details which are not contained in the crust. Although we have presented a solution to resolve this issue for input in the form of point clouds, this problem is more difficult to handle in the case of multi-view reconstruction from images due to the missing explicit data samples. Some of the above issues have been investigated in [LB07]. Concerning the surface characterization and computation, it would be interesting to examine alternative surface energy functionals with higher-order smoothness and corresponding graph structures. This could alleviate issues such as the shrinking bias or metrization artifacts [BK03b].

The input view selection could be extended by more sophisticated image selection criteria which consider, e.g., the quality of the camera calibration, an entropy based analysis of the image content, or the different requirements of reconstruction techniques which are not based on standard photo-consistency measures [HVC08]. Techniques for image-based rendering such as our view synthesis in Chapter 4 could benefit from an input view selection as well. Additionally, our greedy image selection scheme based on three subsequent phases should be evaluated with respect to more sophisticated optimization techniques.

From a broader view, an essential ingredient of all presented techniques, which provides considerable potential for future improvements, is the image-based correspondence analysis or photo-consistency estimation. Despite a number of exceptions, most work in the discussed fields is still based on relatively simple consistency measures for Lambertian scenes. Although we discussed in Chapter 4 how those simple measures can be modified in order to increase their robustness with respect to non-Lambertian surfaces, their applicability to arbitrary surfaces types is obviously restricted. These issues
6 Conclusion

can be resolved by using more sophisticated illumination and surface models [GCHS05],
but those more complex models are often restricted in their practical applicability due
to more stringent requirements with respect to the input data and a higher computa-
tional complexity. With the permanently increasing performance of programmable
GPUs, however, the required processing power is likely to become available in the near
future even in consumer hardware. In this context it would also be interesting to con-
sider the change from the current generation of digital cameras to more accurate high
dynamic range and high resolution acquisition systems for images and video. Improving
image-based correspondence analysis using those techniques could have a considerable
impact on the quality and speed of existing approaches. Finally, it would be helpful to
perform an in-depth analysis of the fundamental relations between problems in shape
representation and reconstruction which are currently considered separately. For exam-
ple, our representation based on volumetric confidence maps and energy minimization
via graph cuts allowed us to devise a unified approach to multi-view stereo and point
cloud reconstruction. Similar observations have been made in related areas in computer
vision and graphics research as well, e.g., for the problem of segmentation or the relation
between graph cuts and level set methods in multi-view stereo [Vog06]. This type of
unification could eventually lead to a better understanding and a more efficient handling
of related problems from other domains.
Bibliography


<table>
<thead>
<tr>
<th>Reference</th>
<th>Citation Details</th>
</tr>
</thead>
</table>


2003, 303–312.


of SIGGRAPH ’05) 24, 3, 2005, 544–552.


Bibliography


[LQ03] Lhuillier M., Quan L.: Surface reconstruction by integrating 3D and 2D data of multiple views. In IEEE International Conference on Computer Vision (ICCV), 2003, pp. 1313–1320.
Bibliography


Bibliography


Bibliography


Curriculum Vitae

Alexander Hornung

E-Mail hornung@inf.ethz.ch

Date of Birth 03.09.1976
Place of Birth Karlsruhe, Germany
Citizenship German

Academic Education

Nov. 2008 – Present Postdoctoral Researcher at ETH Zurich, Computer Graphics Laboratory.
Degree: Dr. rer. nat. (with honors).
Supervisor: Prof. Dr. Leif Kobbelt.
Apr. 2003 – July 2003 Research Assistant at the Academy of Media Arts Cologne, Laboratory for Mixed Realities.
Degree: Diplom (with honors).
Publications


Botsch M., Hornung A., Zwicker M., Kobbelt L., High Quality Splatting on Today’s GPUs, ACM and Eurographics Symposium on Point-Based Graphics (PBG), 17–24, 2005

Hornung A., Sar-Dessai S., Kobbelt L., Self-Calibrating Optical Motion Tracking for Articulated Bodies, IEEE Virtual Reality Conference (VR), 75–82, 2005


Hornung A., Kobbelt L., Robust and Automatic Optical Motion Tracking, 1. Workshop Virtuelle und Erweiterte Realität der GI-Fachgruppe VR/AR, 2004

