complete disparity map. In a second step a surface model is fitted to the reconstructed features [Taylor 03].

Image-based methods that rely on multiscopic matching of n-tuples share an important advantage with scene-based methods: implicit consistency of the reconstruction. Furthermore, both take full advantage of pixel redundancy to avoid as many false matches as possible while enabling smart occlusion handling schemes. The photo-consistency cost implied in those methods is often defined, for each 3D point of interest, as the aggregation of dissimilarity costs of its corresponding pixels over a set $\mathcal{R}$ of several pairs of views

$$c(\mathbf{p}) = \sum_{(i,j)\in\mathcal{R}} c(\mathbf{p}_i, \mathbf{p}_j) \quad . \tag{8.8}$$

Here $c(\mathbf{p}_i, \mathbf{p}_j)$ is the same cost function as used before in the binocular case and $\mathbf{p}_i$ is the pixel position of the backprojected 3D point $\mathbf{p}$ into the i-th view $\mathbf{I}^i$. Commonly employed pair sets $\mathcal{R}$ consist of:

- successive views $\mathcal{R} = \{\ (i, i+1) \mid \forall i \in \{0, \ldots, n-2\}\ \}$,
- every available pair $\mathcal{R} = \{\ (i, j) \mid \forall i, j \in \{0, \ldots, n-1\}, i < j\ \}$,
- pairs specifically selected according to geometrical considerations and/or similar recording conditions.

The first option is often preferred in a rectified layout as it makes the stereo method less sensitive to colorimetric shifts among the image set. Contrarily, the third is used when a very large number of views is available with widely spread viewpoints.

Using all views to compute the dissimilarity cost in Eq.(8.8) rarely leads to high-quality reconstructions, as a scene point $\mathbf{p}$ may be occluded in some of the cameras. However, as multiple views are available visibility may be reconstructed as well during the correspondence estimation [Kolmogorov and Zabih 02, Kutulakos and Seitz 00, Seitz and Dyer 99]. This visibility information can be used to improve the correspondence reconstruction by:

- restricting to a useful set of image pairs $\mathcal{R}_\mathbf{p} = \{(i, j) \in \mathcal{R} \mid$ with $\mathbf{p}$ visible in both i and $j\}$,
- weighting the dissimilarity costs in Eq.(8.8) according to $\mathbf{p}$'s visibility in the images, or
- replacing the dissimilarity cost by a predefined, heavy, penality cost for pairs for which $\mathbf{p}$ would occlude some already reconstructed 3D point.

Multi-view stereovision methods vary strongly with respect to methodology and tend to be computationally more complex than their binocular

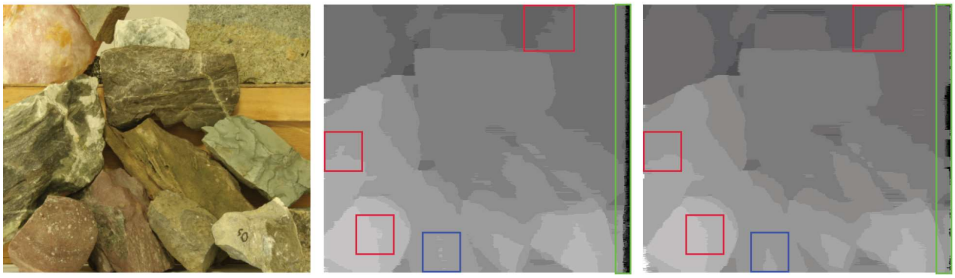


Figure 8.7: Results from a multi-baseline, scene-based method [Ismael et al. 14] on the Middlebury dataset "Rocks2": left, one source view; center, disparity map computed from two views only; right, disparity map computed from a set of 4 views. Green rectangles highlight an area more completely reconstructed from 4 views; red rectangles focus on some areas more regularly reconstructed from four views; the blue rectangle points to a region with higher accuracy in the 4-view case.

counterparts, as they have to deal with more data. Nevertheless, they tend to exploit the increased redundancy to achieve more robust reconstructions: Figure 8.7 shows that, with similar context, data, method, and parameters, results computed from four views are more regular and complete and contain fewer outliers than those using two views.

8.5 Stereo on the GPU

Stereo estimation is typically a significant computational expense and improving the computation time of stereo has long been in the focus of research. One characteristic that has been leveraged is that the depth/disparity estimation in local stereo for pixel (x, y) has no dependencies to any other pixel (Sections 8.2 and 8.3). Hence, parallel computation has long been explored for improving the computation time. Commodity graphics hardware (GPUs) nowadays provides a massively parallel processing platform with thousands of parallel compute cores and a significantly higher memory bandwidth than CPUs have at their disposal. Yang and Pollefeys [Yang and Pollefeys 03] proposed to leverage these highly parallel architectures to improve the computational performance utilizing the plane-sweeping stereo algorithm [Collins 96]. Besides leveraging the parallelism of GPUs, their method further leverages the high efficiency of texture mapping in GPUs.

Plane-sweeping stereo [Collins 96] is a multi-view stereo method with $n > 2$ views, $\mathbf{I}^0$ to $\mathbf{I}^{n-1}$ which does not rely on multiocular rectification

(Section 8.4). It can use any set of multiple overlapping views to perform stereo estimation. Plane-sweeping stereo estimation only requires the camera calibration of the views $\mathbf{I}^0$ to $\mathbf{I}^{n-1}$, as for example computed by structure from motion (Chapter 7). The core idea of plane-sweeping stereo is to perform the dense correspondence estimation by testing a series of plane hypotheses Π^i with $i = 1, \ldots, K$ for the scene, i.e., it assumes the scene is on a plane and then tests this hypothesis. Once all K plane hypotheses have been tested, the best plane is chosen for each pixel. This, however, does not mean that the scene has to be planar, as the plane only represents a local planar approximation of the scene for the function used to compute the matching cost. Please note that each plane is basically a slice in the cost-volume introduced in Section 8.3.

In plane-sweeping stereo the depth map relative to one of the images $\mathbf{I}^0, \ldots, \mathbf{I}^{n-1}$ is computed. This image is referred to as the reference view $\mathbf{I}^{ref}$ and its camera projection matrix is transformed to be $\mathbf{P}^{ref} = [\mathbf{U}_{3\times3} \quad 0_{3\times1}]$ where $\mathbf{U}$ is again the identity matrix. All other images' camera projection matrices are transformed as well to be in the same coordinate system as the reference image $\mathbf{I}^{ref}$. Given this unified coordinate system, the plane hypotheses Π^i are chosen with respect to $\mathbf{I}^{ref}$ to sample the depth interval $[d_{near}, d_{far}]$ with K steps[4] from a closest distance d_{near} to a farthest distance d_{far}. The plane hypotheses Π^i in Yang and Pollefeys [Yang and Pollefeys 03] were chosen to be fronto-parallel to the reference image $\mathbf{I}^{ref}$, i.e., their normals n_i are equal to $[0\ 0\ 1]^T$ and their distance d_i corresponds to the depth (distance from the reference camera to the plane). Conceptually, to test any specific plane hypothesis Π^i all views can be warped onto the plane and their photoconsistency[5] can be evaluated using any of the cost functions from Section 8.3. This would require the definition of a raster on the plane hypothesis Π^i, which is a challenge. Equivalently, a hypothesis Π^i can be tested with respect to the reference view $\mathbf{I}^{ref}$ by warping all other images $\{\mathbf{I}^0, \ldots, \mathbf{I}^{n-1}\} \backslash \mathbf{I}^{ref}$ to the reference view $\mathbf{I}^{ref}$. Photoconsistency at each pixel (x, y) in the reference image $\mathbf{I}^{ref}$ can then be tested using the warped images. The warp of pixel (x, y) from the reference view $\mathbf{I}^{ref}$ to image $\mathbf{I}^j$ over plane Π^i is a planar mapping and can be described by a planar homography $\mathbf{H}_{\Pi^i,\mathbf{P}^j}$. Here, $\mathbf{P}^j = \mathbf{K}_j \begin{bmatrix} \mathbf{R}_j^T & -\mathbf{R}_j^T\mathbf{C}_j \end{bmatrix}$ is the camera projection matrix of the camera corresponding to image $\mathbf{I}^j$ with $\mathbf{K}_j$ being the camera calibration matrix and $\mathbf{R}_j$, $\mathbf{C}_j$ representing the rotation of the camera and the camera center, respectively [Hartley and Zisserman 03].

[4]Please note that the steps are typically not equidistant steps. They are chosen to have equal disparity sampling in the reference view. For more details on the hypotheses generation see Gallup et al. [Gallup et al. 07].

[5]Photoconsistency is the color similarity, i.e., the highest photo consistency is achieved when all views have the same color.

Figure 8.8: Left: Plane-sweeping stereo's reference image from an outdoor video sequence. Right: Depth map computed by plane-sweeping stereo using a total of eleven views.

The homography $\mathbf{H}_{\Pi^i,\mathbf{P}^j}$ is given by:

$$\mathbf{H}_{\Pi^i,\mathbf{P}^j} = \mathbf{K}_j \left(\mathbf{R}_j^T + \frac{\mathbf{R}_j^T \mathbf{C}_j n_i^T}{d_i} \right) \mathbf{K}_r^{-1}. \tag{8.9}$$

Then, the location (x_j, y_j) in image $\mathbf{I}^j$ of the warped pixel (x, y) in the reference image $\mathbf{I}^{ref}$ can be computed by

$$(x' \; y' \; w')^T = \mathbf{H}_{\Pi^i,\mathbf{P}^j} (x \; y \; 1)^T \text{ and } x_j = \frac{x'}{w'}, \quad y_j = \frac{y'}{w'} \quad . \tag{8.10}$$

If the scene point that projects into pixel (x, y) is on the plane Π^i then the colors of pixel (x, y) in the reference image $\mathbf{I}^{ref}$ and the color of pixel (x_j, y_j) in image $\mathbf{I}^j$ should be very similar. Similar to the binocular case, their similarity can be measured by a variety of measures, as explained in Section 8.3.

Using the GPU, the warping between the reference image $\mathbf{I}^{ref}$ and image $\mathbf{I}^j$ with the homography $\mathbf{H}_{\Pi^i,\mathbf{P}^j}$ can be performed using projective texture mapping [Segal et al. 92], i.e., a projection of an input image onto a geometric primitive. This projection is highly efficient on GPUs. The similarity evaluation and the selection of the best plane hypothesis for each pixel in the reference view are independent for each pixel and hence can be performed in parallel as well. This high degree of parallelism provides speedup factors of one hundred and more for stereo estimation on GPUs [Gallup et al. 07]. In [Gallup et al. 07] it is further proposed to improve the local approximation of the scene geometry with the plane hypotheses Π^i by using multiple plane orientations. This indeed improves the accuracy of the stereo estimation by better modeling planes that are seen under oblique viewing directions, for example the ground plane. Figure 8.8 shows an image and its example depth map computed by multi-way plane-sweeping [Gallup et al. 07].

8.6 Summary

This chapter only touched the tip of the iceberg that represents the field of dense correspondence estimation. Nevertheless, the knowledge provided here poses a useful basis for understanding any of the other current state-of-the-art correspondence techniques and provides a flexible basic framework upon which to build. The simplicity of local methods makes them attractive from a beginner's perspective as well as a computational view. Choosing the right dissimilarity function proves crucial for the quality of the algorithm, but with well-chosen adaptive weights, state-of-the-art results are achievable. Posing the aggregation step as a filtering process can dramatically improve the speed of the correspondence algorithm. To handle occlusions, a symmetry check between the input images can be used and several extensions including higher dimensional solution spaces and slanted surfaces can be easily incorporated at the cost of higher computation times.

The finding that current state-of-the-art dense correspondence algorithms achieve energies in the cost function that are below that of ground truth scenes [Szeliski et al. 08] may appear dissatisfying for researchers starting in this area. It should be mentioned, though, that this opens the door for more creative approaches that do not follow the standard paths but try to come up with novel ideas and complete new algorithms that differ in more than the choice of a new data or regularization term. Maybe the way to go is also specialized algorithms specifically targeting certain scenes or applications. While from a vision perspective the goal is to find the best automatic algorithm for dense correspondence matching, it may make sense to have an algorithm that one can improve through additional user input [Klose et al. 11, Ruhl et al. 13], e.g., for multimedia applications like image interpolation. In such cases, these interactive correction tools are of utmost importance. Further on, if massive amounts of images are to be matched, speed may be of highest interest [Frahm et al. 10].

9
Sensor Fusion

Andreas Kolb, Jiejie Zhu, and Ruigang Yang

9.1 Introduction

Many means to recover scene depth have been introduced so far. They range from specialized sensors that can directly return depth (e.g., LiDAR sensors and ToF sensors described in Chapter 4) to the most widely used stereo matching algorithms in Chapter 8 that require just two or more images as input. As with many real-world situations, for example in the context of cultural heritage (Chapter 22), there is no single method that can claim to be the panacea for all and every application that relies on depth data. As shown in Table 9.1, these different depth sensing approaches all have their advantages and disadvantages with respect to several criteria. Some of the disadvantages are inherent to the principle of operations, such as the quadratically growing depth error in stereo, while others are due to design

Table 9.1: Characteristics of different depth sensing methods.

	LiDAR	ToF	Stereo	Light Field	Photo-metric Stereo
Range‡	long	near	mid	mid	near
Depth accuracy	high	high	variable*	variable*	high†
Cost	high	low	low	high	low
Robustness	high	high	low	medium	medium
Resolution	single	low/medium	high	high	high
Dynamic scene handling	limited	yes	yes	yes	yes
Computational overhead	little	little	high	high	medium

‡: long > 20m; mid < 20m; near < 5m;
* The depth error grows quadratically with respect to range.
†Photometric stereo estimates the surface normal direction, not absolute metric depth.

choices or practical and technological considerations, such as the limited measurement range of ToF sensors. Furthermore these different depth sensing methods all have different systematic errors. ToF sensors, for example, cannot reproduce sharp concave corners due to the multi-path propagation of light. Results from stereo matching usually show the "fattening" effect around discontinuities (often due to unavoidable regularization). Naturally, researchers have therefore developed techniques for combining different sensor observations or modalities to increase the sensing performance. This is the central topic of this chapter.

Strictly speaking, *sensor fusion* is the combination of sensory data from disparate sources to improve the resulting data quality, usually in terms of accuracy and robustness. In the scope of this chapter, sensor fusion methods can be roughly divided into two categories: *multi-sample* fusion and *multi-modal* fusion approaches. Multi-sample fusion takes advantage of the redundancy in the input data to significantly reduce the noise in individual sensor readings, generating much cleaner output. It will be discussed in detail in Section 9.2 using several specific algorithmic examples. Multi-modal fusion takes advantage of the often complementary nature of different sensing modalities; for example, it combines the ability of photometric stereo to capture detail with the metric reconstruction accuracy of stereo, in order to reduce systematic errors in the fused data. Several typical examples of multi-modal fusion are presented in Section 9.3. This chapter concludes with a summary in Section 9.4.

9.2 Multi-Sample Fusion

Multi-sample fusion is a research topic with a long history in computer vision. The necessity for multi-sample fusion arises if several independently acquired range scans have to be fused into a single consistent model. Historically, the *Digital Michelangelo Project* [Levoy et al. 00] can be seen as a project involving the full pipeline from laser-scanner based range acquisition via alignment to postprocessing. At that time, the acquistion of a full historical site took 30 nights, and the full processing of the dataset required 1080 man hours resulting in 32 GB model data.

Multi-Sample Fusion Pipeline

Having new real-time range sensing cameras at hand (Section 4), the quest for easy online scene acquisition techniques has gained momentum. Here, a system is desired that is capable of processing a stream of range images in a sequential manner. A first system that allowed for online acquisition of 3D models using a structured light scanner has been presented

by [Rusinkiewicz et al. 02]. In this work, a scanner with up to 60 Hz range image acquisition rate is used and the registration is based on an optimized version of the *Iterative Closest Point (ICP)* algorithm [Besl and McKay 92] (Chapter 10).

[Schuon et al. 09] present an approach for superresolution for range images acquired from time-of-flight (ToF) cameras. Here, only very small motions are performed, so that the range image alignment can be done using optical flow. By further extending this idea, [Cui et al. 10] present a fusion approach using a ToF camera. Due to the low lateral (x/y) resolution of the ToF camera used, their approach is formulated as a simultaneous superresolution and a probabilistic registration technique. The major drawback is the complex optimization approach which is very time intensive and cannot be performed in real-time.

The generic data processing pipeline presented by [Rusinkiewicz et al. 02] is still used in many of nowaday's approaches for free-hand model acquisition based on range cameras and contains the following steps (Figure 9.1):

- **Range Image Acquisition and Data Pre-Processing**: Several sensor and algorithm alternatives are available to acquire range data (Chapter 4). Depending on the device, several pre-pocessing operations may be needed: For instance flying pixels in ToF camera data may need to be removed (Section 4.3). Also data may need to be further filtered and smoothed, e.g., using bilateral filtering.

- **Camera Pose Estimation/Registration**: Any sequential range image acquisition device is assumed to produce redundant data, i.e., overlapping regions between subsequent range images. Using this redundancy, the individual range images can be registered against each other. This operation is usually achieved with an ICP algorithm.

- **Model Update**: After a new frame has been registered into a model, any new range image has to be fused with the already existing scene

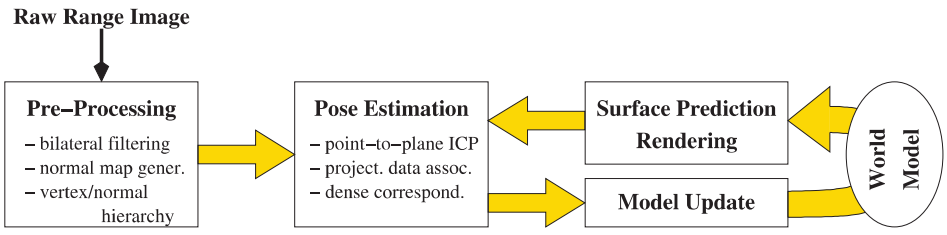


Figure 9.1: The basic workflow of the KinectFusion algorithm.

model. In this step, the redundancy that was needed for the registration step should be removed in order to reduce sensor noise and obtain a compact scene model.

- **Rendering**: Depending on the target application, the model is further processed. In most cases, some kind of visualization, e.g., for visual inspection, is provided, i.e., the model needs to be rendered.

KinectFusion

In the context of modern high-speed range imaging cameras, the KinectFusion system presented by [Izadi et al. 11, Newcombe et al. 11] is the first realization of the pipeline sketched above, that entirely runs in real-time.

Whenever designing a system that is able to process range data at this high throughput, i.e., 512×424 pixel for the Kinect 2^{TM}at 30 FPS, the key question is, what is the right representation of the model. This question is tied to the solution of the associated challenges of how to use this model representation for online registration, merging, and rendering. KinectFusion adopts the *volumetric model* and fusion method of [Curless and Levoy 96]. The KinectFusion algorithm utilizes a *regular voxel grid* in order to store the depth measurements of the range camera device and the world model as *truncated signed distance fields*, where the zero-set corresponds to the scene's surfaces S. The truncated distance of a 3D point $\mathbf{x}$ is defined using a distance threshold μ

$$\text{tsdf}(\mathbf{x}) = \begin{cases} \max(-\mu, -\min_{\mathbf{y} \in S} \|\mathbf{x} - \mathbf{y}\|) & \text{if } \mathbf{x} \text{ is inside } S \\ \min(\mu, \min_{\mathbf{y} \in S} \|\mathbf{x} - \mathbf{y}\|) & \text{if } \mathbf{x} \text{ is outside } S \end{cases}.$$

The motivation for the truncation is mainly the fact that signed distance values with respect to a surface are estimated using the polar distance given in the range image, which is only reliable close to the surface. Transforming the input range data into a truncated signed distance field and merging it with the current representation in the voxel grid is done by sweeping the volume (Algorithm 9.1).

In a first step, the proper alignment of the input range map to the data accumulated in the voxel grid so far needs to be computed. Therefore, for each individual depth map $Z^i(\mathbf{u}) \in \mathbb{R}$ with pixel coordinates $\mathbf{u} = (x, y)$, a map of 3D vertex coordinates, henceforth called *vertex map* $\mathbf{v}^i(\mathbf{u}) \in \mathbb{R}^3$, is generated using the intrinsic camera matrix $\mathbf{K}$. An additional *normal map* $\mathbf{n}^i(\mathbf{u}) \in \mathbb{R}^3$ is computed using a bilateral filtered version of the vertex map. Both maps are further processed to get a multi-scale representation, which is required for the fast ICP method (Section 10.3). The ICP correspondence finding and alignment is subsequently done in a coarse-to-fine

manner, starting with the coarsest hierarchy level. A fixed amount of iterations is performed per level. On each hierarchy level, ICP uses a dense correspondence map for the alignment, which is explained next.

For the alignment of the new input data frame i, it is assumed that the camera pose of the previous frame $\mathbf{T}^{i-1} = \begin{bmatrix} \mathbf{R}^{i-1} & \mathbf{t}^{i-1} \\ 0\ 0\ 0 & 1 \end{bmatrix}$ consisting of the rotation matrix $\mathbf{R}^{i-1}$ and the translation $\mathbf{t}^{i-1}$, is known. One also assumes that the input vertex map $\mathbf{v}^i$, normal map $\mathbf{n}^i$ and distance map Z^i with polar distances for frame i are given (Figure 9.1). The KinectFusion approach follows a frame-to-model alignment, where each vertex in the input vertex map is a potential correspondence to the current truncated signed distance field accumulated in the voxel grid which represents the current *model*. To identify the corresponding points in the model, the previous camera pose $\mathbf{T}^{i-1}$ is used in order to synthesize a *model range map* $\mathbf{v}_m^{i-1}$ and a *model normal map* $\mathbf{n}_m^{i-1}$. To serve this purpose, the so far accumulated truncated signed distance field of the model is rendered using ray casting. From the previously estimated camera pose $\mathbf{T}^{i-1}$, rays are cast through the volume searching for the first zero crossing [Parker et al. 98]. Aligning the input map $\mathbf{v}^i$ to the model map $\{\mathbf{v}_m^{i-1}, \mathbf{n}_m^{i-1}\}$ requires the estimation of the relative rigid transformation $\mathbf{T}^{i\to(i-1)}$, which enables the computation of the new camera pose $\mathbf{T}^i = \mathbf{T}^{i-1} \cdot (\mathbf{T}^{i\to(i-1)})^{-1}$. As mentioned earlier, $\mathbf{T}^{i\to(i-1)}$ is determined using an iterative closest point (ICP) approach. Initializing $\mathbf{T}_k^{i\to(i-1)} = [\mathbf{I}_{3\times3}|\mathbf{0}]$ for $k = 0$, the iterative solution minimizes the following error metric

$$\mathbf{T}_{k+1}^{i\to(i-1)} = \min\arg \sum_{\mathbf{u}\in\mathcal{M}} \left((\mathbf{T}_{k+1}^{i\to(i-1)} \tilde{\mathbf{v}}^i(\mathbf{u}) - \tilde{\mathbf{v}}_m^{i-1}(\mathbf{u}_k)) \cdot \mathbf{n}_m^{i-1}(\mathbf{u}_k) \right)^2. \tag{9.1}$$

Here, $\mathcal{M}$ denotes the set of input range map pixels $\mathbf{u}$ with valid depth, i.e., depth values masked out by the depth sensor as invalid and correspondences with strongly deviating normals $\mathbf{n}^i(\mathbf{u}) \cdot \mathbf{n}^{i-1}(\mathbf{u}_k) < 1 - \epsilon$ are discarded. Furthermore, $\mathbf{u}_k$ is the pixel coordinate in the model map corresponding to $\mathbf{u}$ in the k-th iteration when applying the previous estimate $\mathbf{T}_k^{i\to(i-1)}$ of the relative transformation. Using the known intrinsic camera matrix $\mathbf{K}$, $\mathbf{u}_k$ is computed in homogeneous coordinates as

$$\tilde{\mathbf{u}}_k = \mathbf{K} \cdot (\mathbf{T}_k^{i\to(i-1)})^{-1} \tilde{\mathbf{v}}^i(\mathbf{u}).$$

The computation of $\mathbf{T}^{i\to(i-1)}$ is based on the *small angle* assumption between the iterative steps $\mathbf{T}_k^{i\to(i-1)}$ and $\mathbf{T}_{k+1}^{i\to(i-1)}$. Based on this assumption, the incremental update matrix between the iterative steps can be

linearly approximated, i.e.,

$$\mathbf{T}_{k+1}^{i\to(i-1)} = \boldsymbol{\Delta}\cdot\mathbf{T}_{k}^{i\to(i-1)} \text{ with } \boldsymbol{\Delta} \approx \begin{pmatrix} 1 & -\gamma & \beta & t_x \\ \gamma & 1 & -\alpha & t_y \\ -\beta & \alpha & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R}^{\Delta} & \mathbf{t}^{\Delta} \\ 0\ 0\ 0 & 1 \end{pmatrix},$$

where α, β, and γ are the rotation angles around the $x-, y-$, and $z-$axis. After applying the following separation, $\mathbf{R}^{\Delta} = \mathbf{I}_{3\times 3} + \mathbf{S}^{\Delta}$, and after applying a linear approximation to Eq.(9.1), yields

$$\begin{aligned} &(\mathbf{T}_{k+1}^{i\to(i-1)}\tilde{\mathbf{v}}^{i}(\mathbf{u}) - \tilde{\mathbf{v}}_{m}^{i-1}(\mathbf{u_k}))\cdot\mathbf{n}_{m}^{i-1}(\mathbf{u_k}) = \\ &(\boldsymbol{\Delta}\underbrace{\mathbf{T}_{k}^{i\to(i-1)}\tilde{\mathbf{v}}^{i}(\mathbf{u})}_{=\tilde{\mathbf{v}}_{k}^{i}(\mathbf{u})} - \tilde{\mathbf{v}}_{m}^{i-1}(\mathbf{u_k}))\cdot\mathbf{n}_{m}^{i-1}(\mathbf{u_k}) \approx \\ &(\mathbf{S}^{\Delta}\mathbf{v}_{k}^{i}(\mathbf{u}) + \mathbf{v}_{k}^{i}(\mathbf{u}) + \mathbf{t}^{\Delta} - \tilde{\mathbf{v}}_{m}^{i-1}(\mathbf{u_k}))\cdot\mathbf{n}_{m}^{i-1}(\mathbf{u_k}). \end{aligned}$$

Since $\mathbf{S}^{\Delta}$ is the skew matrix $[\mathbf{a}^{\Delta}]_{\times}$ for vector $\mathbf{a}^{\Delta} = (\alpha, \beta, \gamma)$ one finally gets

$$(-[\mathbf{v}_{k}^{i}(\mathbf{u})]_{\times}\mathbf{a}^{\Delta} + \mathbf{t}^{\Delta} + \mathbf{v}_{k}^{i}(\mathbf{u}) - \tilde{\mathbf{v}}_{m}^{i-1}(\mathbf{u_k}))\cdot\mathbf{n}_{m}^{i-1}(\mathbf{u_k}).$$

This expression is linear in the unknowns $\mathbf{a}^{\Delta}, \mathbf{t}^{\Delta}$, thus minimizing Eq.(9.1) results in solving a linear system in six unknowns. It should be noted that the resulting angles α, β, γ are used to set up a correct rotation matrix instead of using the linearized version.

In KinectFusion [Newcombe et al. 11], a fixed number of ICP iterations is performed on each level, i.e., $4, 5, 10$ from the coarsest to the finest level. Also the ϵ-threshold to discard correspondences due to normal deviation should be adjusted while moving through the hierarchy, i.e., less variation should be allowed on finer levels.

Assuming several registered signed distance fields in a common voxel grid representing different overlapping regions of a given object, the resulting fused implicit model is simply achieved via averaging the distance fields. Thus, fusion is a very efficient and simple step. KinectFusion uses truncated signed distance fields, which carry valid information only in a small band around the measured surface. The thickness of this band is a user-defined parameter μ and needs to be chosen according to the noise level of the sensor and the resolution of the voxel grid.

Averaging needs to be done in such a way that the range inaccuracy or sensor noise is handled properly. In case a grid voxel stores information with high reliability, i.e., many input frames have "seen" this voxel, new points should not have a strong influence on the resulting signed distance value. Vice versa, an input point should be weighted stronger if the voxel

Data: Current depth map Z^i parameterized over pixel coordinates;
Voxel volume; voxel g stores weight $g.w$ and signed distance $g.s$;
Current transformation matrix $[\mathbf{T}_i|\mathbf{t}_i]$;
foreach (x, y) *volume slice* **in parallel do**
  while *sweeping voxel* g *from slice* z_{front} *to* z_{back} **do**
    $\mathbf{v}^g \leftarrow$ convert g from voxel to global 3D coordinates;
    $\mathbf{v} \leftarrow \mathbf{T}_i^{-1}\mathbf{v}^g$ `// voxel in camera space`
    $\mathbf{u}_v \leftarrow$ perspectively project vertex $\mathbf{v}$ onto image plane;
    if $\mathbf{v}$ *in camera view frustum* **then**
      sdf $\leftarrow \|\mathbf{t_i} - \mathbf{v}^g\| - Z^i(\mathbf{u}_v)$ `// compute signed distance`
      `// truncate signed distance`
      if $sdf > 0$ **then**
        tsdf $\leftarrow \min(\mu; \text{sdf})$;
      else
        tsdf $\leftarrow \max(-\mu; \text{sdf})$;
      end
      `// avg signed dist. and weight & assign to grid`
      $g.s \leftarrow (g.w \cdot g.s + w_i(\mathbf{u}_v) \cdot \text{tsdf})/(g.w + w_i(\mathbf{u}_v))$;
      $g.w \leftarrow \min(weight_{\max}, g.w + w_i(\mathbf{u}_v))$;
    end
  end
end

Algorithm 9.1: Updating the truncated signed distance field represented in the voxel volume by averaging a new registered input depth map.

has been observed only a few times so far, i.e., the stored surface information is less reliable. KinectFusion stores in each voxel g the truncated signed distance field value $g.s$ and a weight $g.w$. The weight might be a simple counter, i.e., each new point averaged within the truncation range increases the weight by 1. Alternatively, the input depth map carries additional per-distance weights $w_i(\mathbf{u})$ for each range pixel $\mathbf{u}$. These weights can, for example, represent the accuracy of the range value.

Algorithm 9.1 explains the update process of the signed distance field of the model's volume grid. Each voxel sequence $(x, y, z_{\text{front}}, \ldots, z_{\text{back}})$ is processed in parallel on the GPU. After transforming the current voxel to camera space $\mathbf{v}$ and identifying the corresponding range pixel $\mathbf{u}_v$, the signed distance with respect to the range map sdf is computed (note, that $\mathbf{t}_i$ represents the coordinates of the camera's focal point in world space). The last step is the update of the voxel's distance and weight values.

Improvements

There are several shortcomings of the KinectFusion approach. The first issue is the constrained work space and resolution, which is predefined by the fixed volume grid. As the grid has to be processed on the GPU, 512^3 voxels is a typical resolution that can be handled given the memory constraints of current GPUs. In case of a desired accuracy of 5 mm, the working volume is restricted to some 2.5 m in each dimension. Another costly step is the transformation of the input point set into a truncated signed distance field. A third issue is the difficulty to handle dynamic scenes.

Several approaches have been proposed to overcome constraints imposed by the restricted working volume. One solution strategy is simple volume moving methods, as proposed by [Roth and Vona 12]. They stream out voxels from the GPU in order to make space for new voxels; however, this approach is lossy. [Chen et al. 13a] present a hierarchical GPU data structure which is capable of lossless streaming of subvolumes between GPU and host, decoupling the active volume from a predefined physical space. [Nießner et al. 13] present a method to restrict the volumetric model representation to regions that contain input points. This is achieved by subdividing space into a coarse, regular and virtually infinite voxel block grid. In case input points get assigned to one of these coarse voxel blocks the first time, the "real" voxels on resolution level are allocated and referenced in a hash-table. The associated hash function transforms the voxel block ID into the hash table ID. Thus, voxel blocks without any surface points do not consume any memory. [Nießner et al. 13] demonstrate the acquisition of very large scenes, e.g. a courtyard of $16 \times 12 \times 2$ meters in extent (Figure 9.2).[1]

An alternative KinectFusion approach that uses a point-based rather than a voxel grid representation is presented by [Keller et al. 13]. Their major challenges are the rendering of the point-based model from a given camera pose, and the compactification of the points in the model. Every model point stores the point location, the normal, and a radius. Here, the radius describes the surface portion covered by a model point. Using this information, [Keller et al. 13] explicitly render the model by means of a simple surface-splatting technique which employs the per-point information to generate disk-shaped surface splats. Point fusion is applied in order to prevent memory overflow. First, an image-based correspondence finding establishes corresponding model points for each input point. This is achieved by rendering a high-resolution index map, which stores a single model point index in each pixel. Usually 4×4 pixels (indices) are associated with each input point. In case a model point is sufficiently close to the input point under consideration, the normal is sufficiently similar and the model

[1] http://graphics.stanford.edu/~niessner/niessner2013hashing.html

Figure 9.2: Large courtyard scene acquired with the method presented by [Nießner et al. 13].

point's radius is larger, the model and the input points are merged. Similar to the original KinectFusion algorithm, the number of merges is stored as a weight which represents the reliability of the point; model points with low weights are removed from the model. The radius of an input point is given as the projected pixel size scaled with respect to the distance. Thus, larger points implicitly model higher distances from the camera and, at the same time, they have a higher uncertainty due to sensor noise. Therefore, if an input point has a larger radius than the corresponding model point, it is less reliable and should not be merged. In contrast to the original KinectFusion, this approach explicitly merges normal and radius information.

In order to handle dynamic scenes to a certain extent, the original KinectFusion system [Izadi et al. 11, Newcombe et al. 11] can simply be extended by incorporating a temporal blend weight, resulting in a fading out of model information over time. Thus, if a surface point is not observed for a longer period, it is removed from the model. [Keller et al. 13] propose a more explicit approach to handle dynamic scene objects. The idea is to mark the input points in the dense ICP correspondence map as potentially moving objects, in case they have not been successfully assigned to a model point. By further morphological operations and region growing, connecting parts of potentially moving objects are identified. In case marked input points get a model point as merge partner in the merging stage, the weight of the corresponding model point is set to a low value. If the object is really moving, it is very likely that this model point will not be observed again and thus it is automatically removed from the model later on. Figure 9.3 shows some sample results obtained with this strategy for handling of dynamic scenes. [Keller et al. 13] further show, that the segmentation

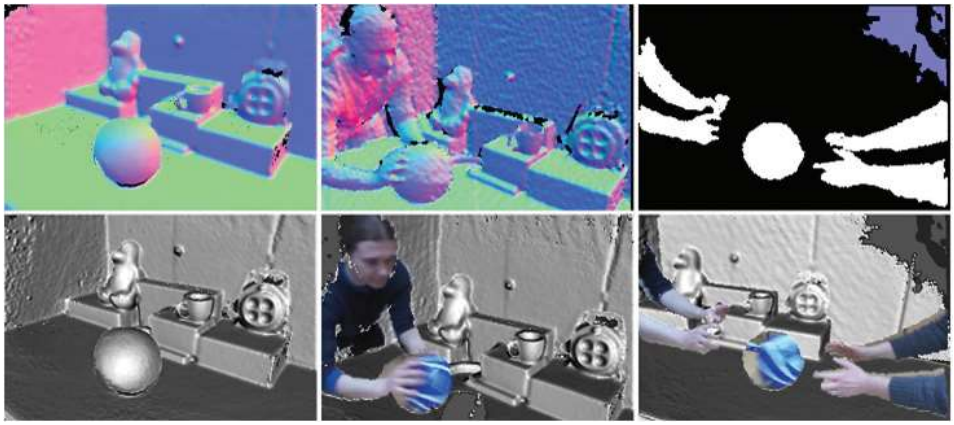


Figure 9.3: Sample scenario capturing a scene with dynamic objects. Merged model normals of the static scene (top left), current input normals after players start to interact with the ball (top center), and the regions detected as potentially dynamic (top right). The lower sequence shows the corresponding renderings of the model (bottom left), superimposed with the dynamic scene parts (bottom center and right).

of potentially moving objects has a very positive influence on the ICP stability, since potentially moving parts are suppressed from correspondence finding during ICP.

9.3 Multi-Modal Fusion

In the previous section, the fusion has been carried out by using multiple samples from the same sensor to achieve better results than any single frame can deliver. However, some sensing modalities have an inherent system bias, which it can be difficult to remove by considering data only from the sensor itself. Fortunately, some sensor modalities have complementary characteristics. In these situations, fusion using two or more distinctive sensor modalities is thus a natural choice. In this section two sets of widely used fusion schemes will be discussed. They are fusion of stereo with shape-from-shading and fusion of time-of-flight sensors with stereo.

Fusion of Stereovision and Shape-from-Shading

Stereovision is based on the principle of *triangulation* (Chapter 8). Given two or more images of the same object, taken from known and different viewpoints, one can triangulate the depth of a 3D point from its corresponding pixels in the input images. The earliest attempt to solve the stereo

problem, by Marr and Poggio, dates back to 1976 [Marr and Poggio 76]. The foundations of shape-from-shading are described in the pioneering work of Woodham [Woodham 80], in which the surface normal of a Lambertian object is estimated using images taken under illumination from three distant light sources. The surface normal field can then be integrated to obtain a 3D surface [Horn and Brooks 86, Agrawal et al. 06].

The complementary nature of stereovision and shape-from-shading has been recognized for many years. Stereovision systems are simple to set up and can generate metric measurements, but their accuracy is inversely proportional to the object distance from the cameras, and stereo reconstructions often lack fine scale detail. On the other hand, photometric stereo is known for capturing surface details, but the integrated surface is typically not metrically accurate and often times suffers from global non-uniform distortions due to the inaccuracy in normal estimation and unknown boundary conditions. Therefore, several methods have been developed to fuse (stereo) depth maps with normal maps [Cryer et al. 95, Nehab et al. 05, Anderson et al. 11]. Early methods [Cryer et al. 95, Mostafa et al. 99] focused on integrating the normal field and the depth map in a postprocessing step. Chen et al. [Chen et al. 03] formulate the reconstruction as a nonlinear optimization; Vogiatzis et.al [Vogiatzis et al. 06] apply the non-linear optimization on vertices of a closed mesh. Nehab et al. [Nehab et al. 05] presented the first paper to combine positional and orientational information in a linear way. Their positional measurement was obtained via active stereo with a projected random pattern and then depth values are assigned to all image pixels. By taking the normals from the photometric stereo technique into account, the recovered shape in the depth map is augmented with high-frequency detail, and the low-frequency reconstruction bias is successfully countered. In the following, a state-of-the-art fusion scheme that explicitly models discontinuities [Zhang et al. 12] is presented as an example.

Suppose the 3D surface $S(u,v) = [x,y,z]^T$ is parameterized in a 2D field $\Omega = [u,v]$ and the initial measured surface is S^0. In this section the superscript ^{0} is used to denote the initial measurement. Assume the field $[u,v]$ coincides with the image grid $\mathbf{u} := [i,j]$, and that per-pixel normal and depth are denoted as $\mathbf{n}(\mathbf{u}) = N_{\mathbf{u}}$ and $z(\mathbf{u}) = Z_{\mathbf{u}}$, respectively. Considering the perspective projection of 3D positions into pixels, the surface can be represented in terms of the depth map $Z_{\mathbf{u}}$:

$$S(\mathbf{u}) = \mu(\mathbf{u}) Z_{\mathbf{u}}, \tag{9.2}$$

$$\mu_{\mathbf{u}} := \begin{bmatrix} \frac{1}{f_x} & & -\frac{p_x}{f_x} \\ & \frac{1}{f_y} & -\frac{p_y}{f_y} \\ & & 1 \end{bmatrix} \begin{bmatrix} \mathbf{u} & 1 \end{bmatrix}^T, \tag{9.3}$$

where $[f_x, f_y]$ and $[p_x, p_y]$ are the camera's focal length and principal point

(Section 1.6). Then the distance of the surface to be reconstructed from the measurement is represented using the depths, as follows:

$$E_p = \sum_{\mathbf{u}} \|\mu_{\mathbf{u}}\|^2 \left(Z_{\mathbf{u}} - Z^0_{\mathbf{u}}\right)^2. \tag{9.4}$$

To use the normal information, the cost function is defined as

$$E_n = \sum_{\mathbf{u}} \left(N^0_{\mathbf{u}} \cdot \frac{\partial S}{\partial u}\right)^2 + \left(N^0_{\mathbf{u}} \cdot \frac{\partial S}{\partial v}\right)^2. \tag{9.5}$$

The corresponding derivatives of the surfaces are

$$\frac{\partial S}{\partial u}(\mathbf{u}) = \mu_{\mathbf{u}} \frac{\partial Z}{\partial u}(\mathbf{u}) + \left[\frac{Z_{\mathbf{u}}}{f_x}, 0, 0\right]^T, \text{ and} \tag{9.6}$$

$$\frac{\partial S}{\partial v}(\mathbf{u}) = \mu_{\mathbf{u}} \frac{\partial Z}{\partial v}(\mathbf{u}) + \left[0, \frac{Z_{\mathbf{u}}}{f_y}, 0\right]^T, \tag{9.7}$$

where $\frac{\partial Z}{\partial u}$ and $\frac{\partial Z}{\partial v}$ are discrete derivatives computed through finite differences. Due to the truncated error and the Gibbs phenomenon[2] arising near the gradient discontinuities, a 3-point derivative formula similar to [Harker and O'Leary 08] was used.

Combining all terms above, the desired surface is obtained by minimizing the following function in terms of depth values:

$$\begin{aligned} E &= \lambda_p \sum_{\mathbf{u}} \|\mu_{\mathbf{u}}\|^2 \left(Z_{\mathbf{u}} - Z^0_{\mathbf{u}}\right)^2 \\ &+ \lambda_n \sum_{\mathbf{u}} \left((N^0_{\mathbf{u}} \cdot \mu_{\mathbf{u}}) \left.\frac{\partial Z}{\partial u}\right|_{\mathbf{u}} + \frac{N^0_{\mathbf{u},x}}{f_x} Z_{\mathbf{u}}\right)^2 \\ &+ \lambda_n \sum_{\mathbf{u}} \left((N^0_{\mathbf{u}} \cdot \mu_{\mathbf{u}}) \left.\frac{\partial Z}{\partial v}\right|_{\mathbf{u}} + \frac{N^0_{\mathbf{u},y}}{f_y} Z_{\mathbf{u}}\right)^2, \end{aligned} \tag{9.8}$$

where $\lambda_p + \lambda_n = 1$ and $\lambda_p, \lambda_n \geq 0$ are blending weights for each penalty. Since the differential operator over the depth map boils down to a matrix multiplication, the total energy is a quadratic form of depth values, which can be solved by an over-constrained linear least square system:

$$\begin{bmatrix} \lambda_p I\mu \\ \lambda_n \left[(N^0 \cdot \mu)\frac{\partial}{\partial u} + \frac{N^0_x}{f_x}\right] \\ \lambda_n \left[(N^0 \cdot \mu)\frac{\partial}{\partial v} + \frac{N^0_y}{f_x}\right] \\ \lambda_s \nabla^2 \end{bmatrix} [Z] = \begin{bmatrix} \lambda_p Z^0 \mu \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{9.9}$$

[2]When a signal with a sharp discontinuity is approximated by its Fourier series, the values around a discontinuity are always oscillating. This is referred to as the *Gibbs phenomenon*.

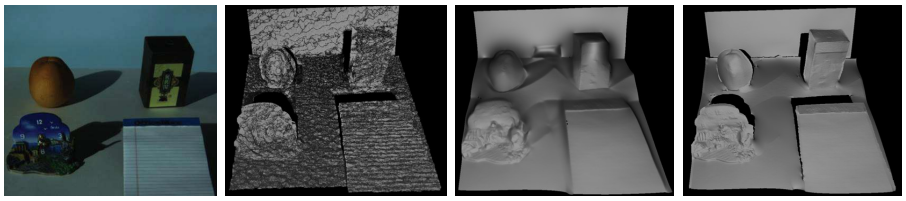


Figure 9.4: Fusion results of shape-from-shading and stereovision [Zhang et al. 12]. From left to right: color input image; depth map obtained via stereo reconstruction rendered in 3D; reconstructed 3D surface after [Nehab et al. 05]; 3D surface reconstructed after [Zhang et al. 12].

where $[Z]$ symbolizes stacking all the depth variables into a column vector, and the multiplications taken in the left-hand side are arranged in the order of each corresponding pixel. The smoothness term is meant to suppress artifacts due to quantization and due to the Gibbs phenomenon, where ∇^2 denotes the Laplacian operator on the 4-neighbor image grid and the weight λ_s is set to a small value. The weights λ_p and λ_n adjust how the depth and normal affect the final reconstructed surface. In practice, a large λ_n and small λ_p are chosen to ensure high-quality details as long as the depth bias of the result is comparable to the original one.

Fusion of Time of Flight (ToF) and Stereo

ToF cameras provide independent range estimates for each pixel in real-time, and are now becoming available from companies such as Microsoft, SwissRanger, and PMDTechnologies at commodity prices. A basic principle of ToF measurement is the timing of the roundtrip of a pulse of light in order to estimate the depth (Chapter 4). Given an array of such pulsed light emitters and a properly designed and synchronized pixel grid, a ToF sensor can return an array of depth measurements. In addition, by calculating the amplitude between the emitted and the returned signal, a ToF sensor can return an intensity image.

Unfortunately, while having the advantage of obtaining depth from texture-less areas, which is difficult with passive 3D reconstruction methods, the ToF sensing principle also has several disadvantages. Each of the pixel range estimates typically has a measurement bias which is dependent on the albedo of the recorded scene. Pixel measurements are also corrupted by noise which usually follows a Poisson distribution around the true depth. In addition, ToF sensors often have low resolution, both in x and y dimensions. In order to overcome these issues, recent works have shown improved depth estimates by modeling ToF and stereo depth as probability functions, and using a global optimization framework to fuse both data sources to yield an improved reconstruction [Zhu et al. 11].

Fusion Framework The estimation of the depth map $Z_{\mathbf{u}}$ for an image can be viewed as a conditional probability. Its posterior probability can be written as

$$P(Z|o) = \frac{P(Z,o)}{P(o)} \propto P(Z,o), \tag{9.10}$$

where o is evidence or observation, $P(o)$ is the normalization factor and usually assumed to be a constant.

To solve for the joint distribution $P(Z,o)$, the true depth map is modeled as a Markov Random Field (MRF). Each node in the MRF represents an independent pixel depth value, and each edge represents this node's relation to a neighboring node. According to the Hammersley–Clifford theorem, the joint distribution of $P(Z,o)$ can be written as

$$P(Z,o) \propto \prod_i \phi(Z_i, o_i) \prod_{\substack{i,j \\ j \in N(i)}} \varphi(Z_i, Z_j), \tag{9.11}$$

where $N(i)$ represents the neighborhood of the node i. Function $\phi(Z_i, o_i)$ is called the evidence function. It encodes the local evidence for the node i. Function $\varphi(Z_i, Z_j)$ is called the compatibility function. It is a smoothness function measuring the depth difference between the node i and its neighboring node j. Applying the negative logarithm to both sides of the above equation yields

$$\begin{aligned} -\log(p(Z,o)) &= -\log(\prod_i \phi(Z_i, o_i) \prod_{\substack{i,j \\ j \in N(i)}} \varphi(Z_i, Z_j)) \\ &= \sum_i -\log(\phi(Z_i, o_i)) + \sum_{\substack{i,j \\ j \in N(i)}} -\log(\varphi(Z_i, Z_j)). \end{aligned} \tag{9.12}$$

To compute the optimal depth values for a frame is thus equivalent to minimizing the right-hand side

$$\mathbf{E(Z)} = \mathbf{min}(\sum_i -\log(\phi(Z_i, Z_i)) + \sum_{\substack{i,j \\ j \in N(i)}} -\log(\varphi(Z_i, Z_j))), \tag{9.13}$$

The first sum in the above equation represents a data term which calculates the deviation of the estimate from the observed data (evidence). The second sum is often referred to as a smoothness term, and it encodes prior knowledge of what depth values a node is likely to have given its neighboring pixel's depth values. The notation can be simplified in the above equation by using function $D(i)$ to represent the data term, and function $V(i,j)$ to represent the smoothness term. Node j is a neighbor of node i:

$$\mathbf{E(Z)} = \sum_i D(i) + \sum_{\substack{i,j \\ j \in N(i)}} V(i,j). \tag{9.14}$$

In order to fuse depth from the ToF sensor and that from the stereo, the observation O is modeled from two components x, y. The variable x denotes the depth evidence from stereo, and y is the depth evidence from ToF. By assuming that x and y are independent, and $P(x)$ and $P(y)$ are constants, the posterior probability from Eq.(9.10) becomes

$$P(Z|x,y) = \frac{P(Z,x,y)}{P(x,y)} = \frac{P(Z,x,y)}{P(x)P(y)} \propto P(Z,x,y), \tag{9.15}$$

An enhanced energy function is derived by introducing two weighting factors to allow for more flexibility in fusing ToF and stereo:

$$D(i) = w_s \cdot f_s(Z_i, x_i) + w_t \cdot f_t(Z_i, x_i). \tag{9.16}$$

Here, f_s and f_t are functions of the data cost for stereo and the ToF sensor, respectively. The factor w_s and w_t are weights that trade-off the relative robustness of depth measurements from passive stereo and ToF reconstruction, respectively. The following paragraphs described these two terms in more detail.

Stereo Matching Cost and Weighting The cost function of stereo matching f_s is designed as pixel dissimilarity between the left and right views (the two views are rectified in which matched pixels can be searched on a scan line) with an aggregation process. To weight on both smooth and discontinuous regions, an appropriate window should be selected during the cost aggregation. In a sense, the window should be large enough to cover a sufficient area in textureless regions, while small enough to avoid crossing regions with depth discontinuities. In our implementation, a color weighted aggregation is incorporated to obtain this reliable correlation volume.

The weights are computed using both color and spatial proximity to the central pixel of the support window. The color difference in this support window (in the same view) is expressed in RGB color space as:

$$\Delta C(x,y) = \sum_{c \in R,G,B} |I_c(x) - I_c(y)|, \tag{9.17}$$

where I_c is the intensity of the color channel c. The weight of pixel x in the support window of y (or vice versa) is then determined using both its color and spatial difference as:

$$w_{xy} = e^{-(\frac{\Delta C(x,y)}{\gamma_C} + \frac{\Delta G_{xy}}{\gamma_G})}, \tag{9.18}$$

where ΔG_{xy} is the geometric distance from pixel x to y in the 2D image grid. The factors γ_C and γ_G control the shape of the weighting function, and their values are determined empirically.

The data term (cost in the left and right views) is then an aggregation with the soft windows defined by the weights as:

$$f_s(x_l, x_r) = \frac{\sum_{y_l, y_r \in W(x_l) \times W(x_r)} w_{x_l y_l} w_{x_r y_r} d(y_l, y_r)}{\sum_{y_l, y_r \in W(x_l) \times W(x_r)} w_{x_l y_l} w_{x_r y_r}}, \tag{9.19}$$

where $W(x)$ is the support window around x; $d(y_l, y_r)$ represents the pixel dissimilarity using Birchfield and Tomasi's approach [Birchfield and Tomasi 98]; x_l and y_l are pixels in the left view; x_r and y_r are pixels in the right view. Results show that f_s can provide both moderate smoothness and preserve boundary sharpness on depth.

The best depth candidate for a pixel should have the lowest cost value of all the possible depth candidates. matching reliability of pixel p is inutitively defined as how distinctive the best and the second best cost is:

$$Ratio_p = \begin{cases} 1 - \frac{c_p^{1st}}{c_p^{2nd}} & C_p^{2nd} > T_c \\ 0 & otherwise \end{cases}, \tag{9.20}$$

where c_p^{1st} and c_p^{2st} are the best (lowest) and the second best matching cost of all depth candidates at pixel p. T_c is a small threshold value to avoid division by zeros. $Ratio_p \in [0, 1]$ can be used to prevent the deteriorating effects of ambiguous matching in case of poor signal-to-noise ratio (SNR).

ToF Cost The cost function for the ToF data f_t measure the depth difference reported from the ToF sensor d_t and a vector of depth candidates from stereo triangulation d_s. The transformation between them is estimated by a pre-calibration step similar to the geometric calibration step in [Zhu et al. 11] where the three cameras (for calibration, the ToF sensor is regarded as a regular camera because it can return a grayscale image along a depth map) are registered into one coordinate system. The method is explained in the following.

For each pixel p in the left (stereo) view, given a vector of disparity candidates d_c, a list of 3D candidate point locations for pixel p after stereo triangulation is generated. For each depth candidate, the pixel's depth cost is computed as the distance to the point returned from the ToF sensor (Figure 9.5).

[Zhu et al. 11] observed that the ToF's measurement reliability depends on the distance to the scene and the reflection of the captured object. In order to mathematically model the relationship between reliability and a pixel's true geometric distance, they placed a white calibration board into the scene and captured multiple frames at different distances. They noticed that the further a depth pixel is radially away from the center of the image, the larger the variance in the depth measurements of multiple

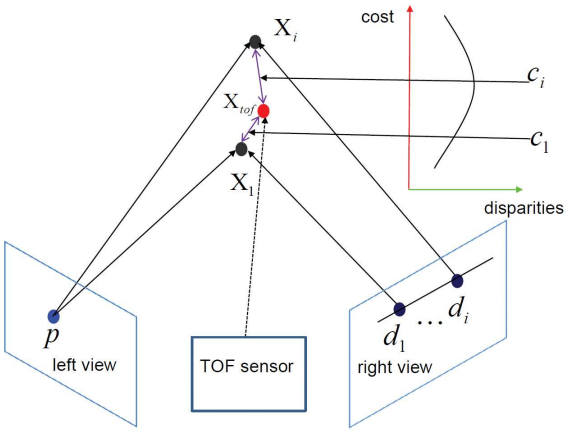


Figure 9.5: Example of how to calculate the ToF sensor cost. Disparity candidates for a pixel in the view of the left stereo camera and its disparity candidates are triangulated into a 3D space; ToF and RGB cameras are calibrated into a common coordinate system. The Euclidian distance between 3D candidates and 3D points captured by the ToF sensor determine the cost.

frames is. To calibrate the same dependency on an object's reflection, they took multiple depth images of boards with changing reflectances. They observed low deviations from the ground truth depth for highly reflective objects, and high deviations for low intensity objects. Based on these two observations, they were able to fit a reliability function for depth values measured by the sensor that depends on distance and intensity.

Figure 9.6 shows an example of ToF and stereo fusion results.

Other Fusion Methods The fusion framework introduced in Section 9.3 can be regarded as a *global method* which considers the depth from the ToF sensor through an additional data term, and then jointly optimizes for the depth values in a whole frame. In the literature, first methods for fusing ToF with stereo employed *local methods* for reconstruction. They optimize the per-pixel depth within a local window using evidences from either type of sensor modality [Kuhnert and Stommel 06, Beder et al. 07, Gudmundsson et al. 08, Hahne and Alexa 09, Bartczak and Koch 09, Chan et al. 08, Yang et al. 10] instead of over a whole frame. Recently, most of ToF and stereo fusion works have focused on *global methods*, because they yield and improved depth accuracy. For example, Kim et al. [Kim et al. 09] proposed a method for fusion of multi-view ToF and multi-view stereo to reconstruct dense 3D models by combining depth from ToF and multi-view stereo, and by using additional object silhouette constraints. Ruhl

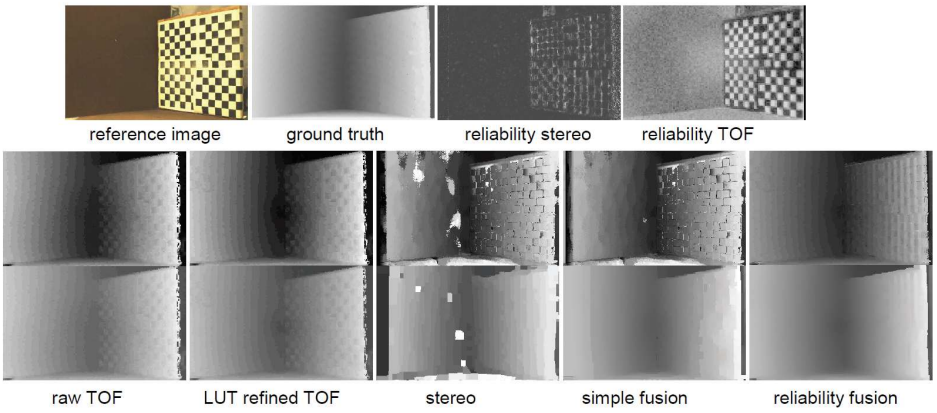


Figure 9.6: Example from [Zhu et al. 11]. Depth map from a simple scene with two planar boards. The first row shows the reference image, ground truth depth from a structured-light method and two maps of per-pixel reliability for stereo and the ToF sensor. The second row shows the depth map from local method. The third row shows the depth map from the global method. From left to right are: Raw depth from the ToF sensor, refined depth from the ToF sensor using a look up table, depth from stereo, depth from simple fusion, and depth from reliability fusion. Simple fusion results are obtained by setting f_s and f_t equal to 0.5. While the reliability fusion results are obtained by automatically computing f_s and f_t, as introduced in the text.

et al. [Ruhl et al. 12b] and Nair et al. [Nair et al. 12] formulated a variational model to fuse ToF and stereo for a sequence of frames by assuming a continuous image domain and continuous variables for estimation.

9.4 Summary

Sensor fusion has been proven to be a very effective approach to improve the quality of range sensing. When the depth map from a single measurement from a single sensor type is not of satisfactory quality, sensor fusion techniques can be considered for data enhancement.

10 Mesh Reconstruction from a Point Cloud

Tamy Boubekeur

10.1 Introduction

Typical 3D capture setups, such as *structure from motion*, Chapter 7, *multi-view stereo*, Chapter 8, and *RGBD sensors*, Chapter 9, provide a dense sampling of the measured 3D object or scene. In most cases, the measured data can take the form of a dense point cloud sampling the geometry of the scene and carrying surface attributes such as diffuse color and more advanced appearance parameters. However, the majority of visual computing applications require a continuous surface representation of their models which either interpolates or approximates the point data. The standard format of this representation is an *indexed triangle mesh* which consists of a list of vertices and a list of triangles indexed over the vertices.

Surface reconstruction algorithms [Hoppe et al. 92, Amenta et al. 01, Kazhdan et al. 06] convert 3D point clouds to meshes, "filling the holes" between the measured point samples and eventually removing the noise they embed. The pipeline supporting most of such algorithms contains several pieces:

1. **pre-processing stages**, including registration, outlier removal, and normal estimation,
2. **a surface model** defined from the processed sampling, which can take the form of a scalar field approximating the distance to the surface in 3D, a projection procedure folding the entire 3D space on the surface subspace, a binary indicator function defining an inside/outside segmentation of the surrounding space, or an energy measuring how close a given surface is from the point set,
3. **a mesh extraction method** discretizing the model in a polygon surface mesh,
4. **post-processing stages**, which are optional and act on the newly defined mesh structure by refining, simplifying, or filtering it.

Most of the numerous reconstruction methods can be described through this decomposition and formulated within an implicit, combinatorial or variational framework. They are characterized according to the kind of input point cloud they can handle (e.g., noisy, incomplete, large, varying in density) and the performance level they can reach (e.g., interactive, parallel scalable, local).

10.2 Overview

A complete review of existing reconstruction techniques goes beyond the scope of this chapter and can be found in the survey by Berger et al. [Berger et al. 14]. Essentially, these algorithms range from local surface approximation models to global and slower solutions. Local methods model the surface at a certain point in space by looking only at a subset of the input set to fit a simple object (e.g., radial basis function, analytical shape). These methods scale well and can still handle small-scale high-frequency noise in the input. Global methods compute the surface model at once using the entire input set and can cope with large holes, outliers, and high-quality connectivity in the output.

This chapter focuses on a simple solution which gives convincing results on dense point sets while being quite simple to implement: *moving least-squares* (or MLS) approaches [Levin 98, Levin 03]. Such reconstructions are local in the sense that they are mostly output-sensitive and do not require solving a large system of equations. They are also amenable to parallel implementations and streaming/out-of-core evaluations.

A typical MLS reconstruction pipeline is composed of three steps. First the point set is preprocessed by registering several partial subsets in the same frame, indexing the resulting sample set in a hierarchical data structure (e.g., kd-tree) to speed-up neighborhood queries, removing outliers, and estimating normals vectors (if not provided). Second, a particular MLS model is choosen, which typically takes the form of a simple geometric primitive, a fitting procedure and a projection operator. Last, a meshing machinery is executed using the MLS operator to locally detect the surface in 3D space. Optionally, a cleaning post-processing step can be performed to improve the connectivity quality, simplify/densify the mesh structure, or even compress it. When implementing parts of the full mesh reconstruction pipeline, a large number of the required operators can be supported by open source libraries such as the *Point Cloud Library*,[1] for instance.

[1] http://pointclouds.org

10.3 Registration

Mesh reconstruction algorithms assume that their input point cloud provides a reasonable sampling of a piecewise smooth real-world surface. This sampling usually corresponds to the union of several subsets which have been independently generated and individually provide only a partial covering of the object to reconstruct, as seen from a particular point of view in the real scene. Therefore, prior to reconstruction, these subsets are registered together. Essentially, this registration step aligns together different point sets measured on the same object by providing, for each of them, a specific space transformation function. When applied to each individual sample, this transformation brings it to the same global coordinate system used for the complete point set.

The *iterative closest point* (or ICP) algorithm [Besl and McKay 92] and its variants [Rusinkiewicz and Levoy 01] has become the standard method to perform this registration (Section 9.2). Assuming an initial rough alignment of two point subsets A and B, as well as some overlapping between them, the ICP algorithm computes a single rigid transformation for each individual subset by iterating with the following procedure:

1. select a subset of samples in A,
2. find, for each of them, the closest sample in B,
3. compute the rigid transformation that minimizes the distance of the so-defined pairs,
4. apply this transformation to B,
5. restart in 1 until convergence or a prescribed number of iterations.

The minimization is usually performed in the least-square sense and the numerous variants of this algorithm are either improving on the metric used to select closest samples, or they are improving the heuristic for selecting samples to match.

ICP provides an estimation of the extrinsic parameter of the sensors relative to a particular one, is simple to implement and fast to compute (Figure 10.1). Open source implementations, such as libICP[2] are available online. However, the intrinsic parameters and depth estimation biases may cause distortion in each subset already during the capturing step. In this case, the single per-subset rigid transformation of ICP is not suifficient and a non-rigid solution [Brown and Rusinkiewicz 07] may be required.

[2] http://www.cvlibs.net/software/libicp/

Figure 10.1: ICP registration of two point sets (in green and orange) coming from two distinct acquisitions (scans) of the same object.

10.4 Outlier Removal

While the smale-scale high-frequency noise is usually removed efficiently by mesh reconstruction algorithms, isolated samples located far away from the surface and corresponding to various errors occurring during the capturing process usually degrade strongly the quality of the reconstruction and shall be removed prior to the modeling and meshing stages.

Such samples are usually called *outliers*, and can be efficiently classified using local statistics that capture, for each sample, how far its closest neighbors are. A simple example for such statistics is the distance between the sample and the centroid of its k nearest neighbors: starting from a conservative threshold, which is application- and scale-dependent, the outlier classification is typically performed by iterating the computation of the statistics while decreasing the threshold progressively. An alternative solution casts the problem as a local density estimation by computing the *local outlier factor* [Kriegel et al. 09, Wang et al. 13a], classifying samples as outliers when the local density of their neighborhood is low.

The particular case of structured outliers is certainly harder to handle automatically. Although recent advances have shown efficiency for a number of cases [Giraudot et al. 13], this problem remains open in general, in particular in the context of high performance reconstruction pipelines.

10.5 Normal Estimation

Providing a normal vector $\mathbf{n}$ for each point sample can be done with at least two techniques. When the point cloud has been generated by registering several depth (2.5D) images, the gradient of the depth values in image space provides a good estimate of the local tangent plane of the geometry and therefore a simple (normalized) cross product of the horizontal and vertical gradients at the depth pixel $\{x, y\}$ gives a good normal vector for the equivalent 3D sample pretty much everywhere:

$$\mathbf{n} = \frac{G^X_{x,y} \otimes G^Y_{x,y}}{\|G^X_{x,y} \otimes G^Y_{x,y}\|}.$$

This estimate is indeed wrong only when there is a local high variance in depth (e.g., object contour). In such cases, the normal is corrupted and can be considered as noise, which will usually be filtered out in the upcoming stage of the pipeline. If the sensor topology is not known in the point cloud (unorganized point clouds), Hoppe et al. [Hoppe et al. 92] have proposed a simple solution to compute the normal: given a point sample with position $\mathbf{p}$ and $\mathcal{N}_\mathbf{p}$ its k nearest neighbors, the covariance matrix M of $\mathcal{N}_\mathbf{p}$ is computed as:

$$M = \Sigma_{\mathbf{q} \in \mathcal{N}_\mathbf{p}} (\mathbf{q} - \mathbf{o}) \bullet (\mathbf{q} - \mathbf{o})$$

with $\mathbf{o}$ the centroid of $\mathcal{N}_\mathbf{p}$ and $\bullet$ the outer product vector operator. The estimated normal $\mathbf{n}$ is taken along the eigenvector associated to the smallest eigenvalue of M. This eigenvector provides a good estimate of the orientation of the normal vector, but is defined locally and therefore cannot help deciding on the direction of the normal, i.e., two possible normals depending on which side of the local tangent plane is inside the object. To cope with this problem, the direction of the normals requires building a minimum spanning tree to be consistently oriented, unless the original position of the sensor which generated the sample is known, which avoids this direction ambiguity. In both cases, the resulting normal field carried by the point set can eventually be improved using a bilateral filter.

10.6 Point Set Surface

A simple and elegant way to define a surface model from the pre-processed sample set $\mathcal{S}$ is to use *point set surfaces* [Alexa et al. 01, Guennebaud and Gross 07, Alexa and Adamson 09], or (PSS), which are defined as the stationary set of $\mathbb{R}^3$ under a MLS projection.

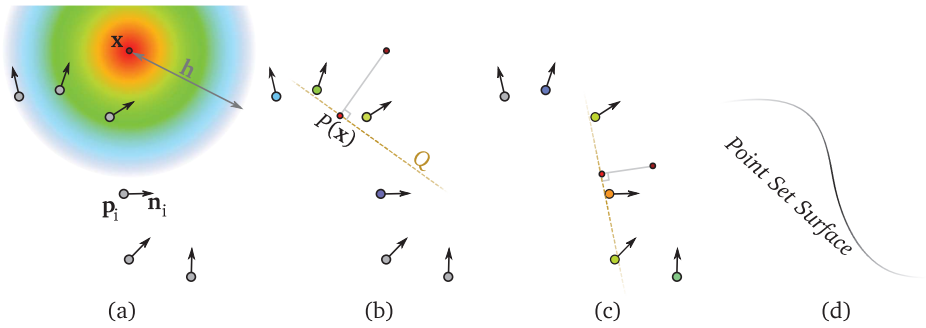


Figure 10.2: PSS from MLS projection illustrated in 2D. (a) The weighting kernel (in gradient color) is set at the evaluation position $\mathbf{x}$ (dark red) with support h. (b) A primitive (here a simple plane) Q is optimized to fit, in the weighted least square sense, the input point set and their normals, with the per-sample weight defined by the $\mathbf{x}$-centered kernel. The resulting projection $P(\mathbf{x})$ is in light red. (c) Illustration of the "moving" effect when evaluating the projection from a different point in space. (d) resulting PSS.

MLS projection Let us consider an input point sample set $\mathcal{S} = \{\mathbf{s_i}\}$, for which each sample $\mathbf{s_i} = (\mathbf{p_i}, \mathbf{n_i})$ carries a 3D spatial position $\mathbf{p_i} \in \mathbb{R}^3$ and a normal estimate $\mathbf{n_i} \in \mathbb{S}^2$. Given any point $\mathbf{x} \in \mathbb{R}^3$, the MLS projection of $\mathbf{x}$ with respect to $\mathcal{S}$ is defined as:

$$MLS_{\mathcal{S}}^{Q,\omega_h}(\mathbf{x}) := {P_{\mathcal{S}}^{Q,\omega_h}}^{\infty}(\mathbf{x}) = P_{\mathcal{S}}^{Q,\omega_h} \circ ... \circ P_{\mathcal{S}}^{Q,\omega_h}(\mathbf{x})$$

with $P_{\mathcal{S}}^{Q,\omega_h} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ being a projection onto the geometric primitive Q (e.g., plane), fitted to $\mathcal{S}$ using a weighting kernel ω_h centered at $\mathbf{x}$ with support h.

This MLS projection procedure is rather easy to implement and can be summarized as follows (see Figure 10.2):

1. gather $\mathcal{N}_{\mathbf{x}}$, a local set of neighboring samples of $\mathbf{x}$ in $\mathcal{S}$,
2. fit Q to this set, weighting the contribution of each neighboring sample by its distance to $\mathbf{x}$ using ω_h,
3. project $\mathbf{x}$ onto Q once fitted,
4. restart in 1 until convergence or a maximum number of iterations is reached.

The neighborhood query is usually the most expensive step and is greatly sped up if a kD-tree [Bentley 75] has been pre-built over $\mathcal{S}$. The weighting kernel ω_h typically drives the low-pass filtering effect of this procedure. Typical choices for the weighting kernel are Gaussian or compact

polynomial ones, such as, for instance, the Wendland quartic kernel [Wendland 95]:

$$\omega_h(t) = (1 - \frac{t}{h})^4(\frac{4t}{h} + 1) \text{ if } 0 < t < h,\ 0 \text{ otherwise .}$$

The support size h of ω tailors how much high frequency will be removed in the reconstruction, which corresponds to both noise and surface details: large values for ω better remove noise but may lead to over-smoothed surfaces.

Beyond the choice of this kernel, the main difference between the various flavors of MLS projections resides in the choice of the fitted primitive Q: a plane, a sphere or other simple shapes are usually good choices because they allow using simple fitting procedures. The most simple solution is probably to compute a "mean plane" [Alexa et al. 04] defined by a center $\mathbf{c}$ and a normal $\mathbf{n}$, for which the fitting procedure boils down to the weighted average of the neighoring sample positions and normals:

$$\mathbf{c}_{\mathcal{S}}^{\omega_h}(\mathbf{x}) = \frac{\Sigma_{\mathbf{s_j} \in \mathcal{N}_\mathbf{x}} \omega_h(||\mathbf{x} - \mathbf{p_j}||)\mathbf{p_j}}{\Sigma_{\mathbf{s_j} \in \mathcal{N}_\mathbf{x}} \omega_h(||\mathbf{x} - \mathbf{p_j}||)} \quad \mathbf{n}_{\mathcal{S}}^{\omega_h}(\mathbf{x}) = \frac{\Sigma_{\mathbf{s_j} \in \mathcal{N}_\mathbf{x}} \omega_h(||\mathbf{x} - \mathbf{p_j}||)\mathbf{n_j}}{||\Sigma_{\mathbf{s_j} \in \mathcal{N}_\mathbf{x}} \omega_h(||\mathbf{x} - \mathbf{p_j}||)\mathbf{n_j}||}$$

and the projection is simply defined as:

$$P_{\mathcal{S}}^{\omega_h}(\mathbf{x}) = \mathbf{x} - ((\mathbf{x} - \mathbf{c}_{\mathcal{S}}^{\omega_h}(\mathbf{x})) \cdot \mathbf{n}_{\mathcal{S}}^{\omega_h}(\mathbf{x}))\mathbf{n}_{\mathcal{S}}^{\omega_h}(\mathbf{x}),$$

A number of extensions have been proposed to improve this basic framework. One can, for instance, fit algebraic spheres [Guennebaud and Gross 07] to better cope with poor input samplings, take into account the normals of $\mathcal{S}$ using a hermite interpolation [Alexa and Adamson 09], preserve sharp features [Fleishman et al. 05] or even exploit the input set self-similarity in a non-local scheme [Guillemot et al. 12]. Several alterative projection procedures have also been proposed, to ensure orthogonality or better control the low pass filtering.

Implicit form The MLS projection procedure also provides an implicit form $f_{\mathcal{S}} : \mathbb{R}^3 \to \mathbb{R}$ of the surface when simply looking at the signed distance between the original location $\mathbf{x}$ and the converged location under MLS projection. This distance can be approximated by estimating a local tangent plane at the converged location and computing the signed distance from $\mathbf{x}$ to this plane:

$$f_{\mathbf{S}} = -(\mathbf{x} - MLS(\mathbf{x}) \cdot \mathbf{n_x}).$$

The 0-contour (iso-contour for the value 0) of this implicit MLS form is often instrumental for meshing algorithms which compute a polygonal contour of the point set surface (Figure 10.3).

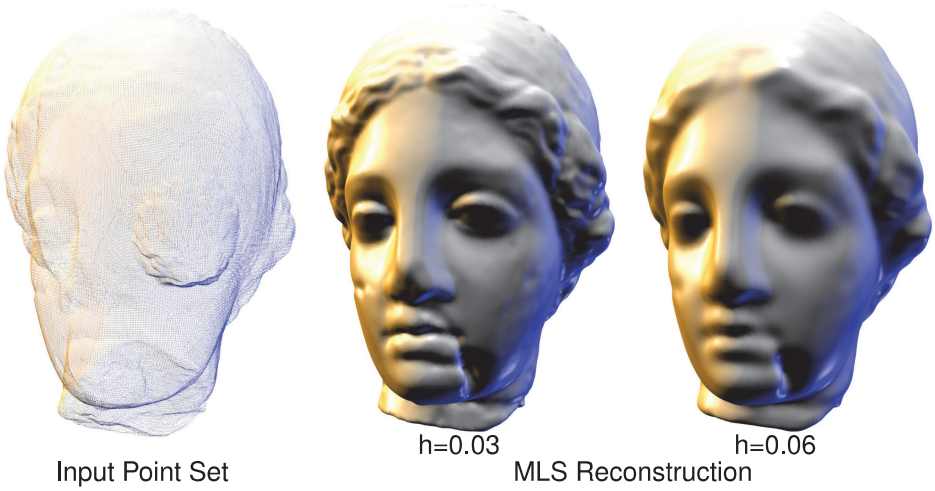


Figure 10.3: Mesh reconstruction from an unorganized point cloud using the MLS operator (here with a least square plane as the geometric primitive) and a mesh contouring algorithm.

10.7 Meshing

With the point cloud in hand and the MLS projection choosen, the output mesh structure can be extracted using several techniques. At least three classes of algorithms have proven to be effective here: *contouring*, *Delaunay meshing*, and *3D snakes*.

Contouring Contouring algorithms, such as *marching cubes* [Lorensen and Cline 87] or *dual contouring* [Ju et al. 02] methods, can be used on the implicit form of the surface. A number of open source implementations of such techniques are available online [Kobbelt et al. 01] and they can be quite easily translated to parallel (e.g., GPU) versions. Such approaches clearly have the advantage of being fast, but the resulting mesh quality is often low, and a remeshing post-process may be mandatory. Still, contouring is clearly the method of choice for most application scenarios requiring to mesh an implicit surface. Most of these techniques can be summarized with the following sequence:

1. Compute a bounding box of the surface to mesh.
2. Generate a 3D lattice inside the box, such as for instance a regular grid or an octree.
3. Evaluate the implicit surface either at the corners of the lattice cells (e.g., marching cube) or inside the cell (e.g., dual contouring).

4. Depending on the sign of the implicit surface at those evaluation locations, the cell containing the surface can be detected. The combinatorics of positive (inside) and negative (outside) implicit values induce the right local mesh structure to generate.

The original marching cube algorithm is deciding on the cell polygonization looking only at the 8 cell's corners sign (256 possibilities, boiling down to 15 when accounting for symmetry). The small set of polygons generated in each cell then approximates the geometry of the implicit surface in the cell, but solving for all cases requires looking also in the neighboring cells to ensure a watertight surface mesh in the end.

Delaunay meshing A better, yet slower, alternative is to use a *restricted Delaunay triangulation* over the implicit form of the PSS stemming from the point cloud. In this case, the connectivity is nearly optimal and the triangles are well-shaped. Such a meshing method can be implemented using the CGAL library[3] for robust computations.

3D snakes Last, a 3D *deformable model* can also be used together with the projective form of the PSS to generate the mesh while controlling its topology. The evolving mesh can be progressively remeshed to preserve well-shaped triangle and adapt to the geometric signal but such solutions are usually slow and hard to implement robustly.

10.8 Mesh Processing

The reconstructed mesh often has a resolution which is proportional to the input point set or related to the level of details captured in the original sampling. This mesh density may not fit the application scenario and the mesh then needs to be either simplified or subdivided. The simplification step can be performed using progressive *edge-collapses*, ordered using the quadric error metric [Garland and Heckbert 97], for which a number of open source implementations are available, such as QSlim[4] or MeshLab.[5] The subdivision can be performed using the Loop subdivision scheme, which has also a number of implementations available, such as OpenSubdiv.[6] A complete survey on how to process polygonal meshes can be found in the book of Botsch et al. [Botsch et al. 10].

[3] http://www.cgal.org
[4] http://www.cs.cmu.edu/afs/cs/Web/People/garland/quadrics/qslim.html
[5] http://meshlab.sourceforge.net/
[6] http://graphics.pixar.com/opensubdiv/

10.9 Summary

Using simple local operators with contouring extraction can cope with the meshing of dense point clouds exhibiting a restricted amount of noise. This solution is extremely fast and easy to run on GPU. However, point clouds exhibiting severe defects, such as large holes or unreliable normals, require more complex solutions. The Poisson Surface Reconstruction [Kazhdan et al. 06] is for instance quite efficient at filling large holes, while a combinatorial solution based on the *graph minimal cut* algorithm [Hornung and Kobbelt 06] can generate a high-quality surface without using the normal information.

The mesh reconstruction pipeline presented in this chapter allows to convert efficiently the raw 3D point sampling coming, for instance, from multiview stereo, laser range scanning, structure-from-motion data, or explicit surface models. The resulting mesh model can be enhanced with a parameterization to store surface attributes and support the basic intersection tests (with lines or volumetric primitives) which are at the root of physically based simulation and rendering. Last, it serves itself as input for numerous higher level geometry processing and analysis methods, including 4D face capture, full body reconstruction, animated cloth motion modeling, automatic rigging (Chapter 13), and 3D web compression and transmission (Chapter 22).

11 Reconstruction of Human Motion

Yebin Liu, Juergen Gall, Céline Loscos, and Qionghai Dai

11.1 Introduction

Motion capture is a technology to record and digitalize the motion information of living creatures. In recent years, with the emergence of new types of sensors and the improvement of computational performance, motion capture has started to play an important role in many fields, such as realistic character animation for games and movies, motion analysis for medical diagnostics and sport science, or virtual reality. While motion capture techniques can apply to general moving objects, they are mostly applied to capture full-body motion of humans or the motion of body parts like the face or hands. This chapter focuses only on human motion.

Motion capture techniques can be split into two categories, depending on if they use some form of fiducials or not. Marker-based systems require the attachment of different kinds of sensors or markers on the subject that is captured. By considering the types of sensors used, one can distinguish four categories: mechanical systems, electromagnetic systems, inertial systems, and optical systems (Figure 11.1). Among these, optical marker-based motion capture systems are the most popular ones, in particular in the movie industry, since they are very accurate and achieve real-time performance. However, marker-based systems suffer from widely known shortcomings, such as the need for a special controlled capture environment, errors due to broken marker trajectories, and their inability to capture motions of people wearing normal everyday apparel. Moreover, it is difficult to capture natural motions when a special marker suit or other technical equipment is disturbing. For example, when capturing the motion of hands interacting with objects, markers or sensors on the hands or the object might result in unnatural grasping poses. In general, the often considerable setup times with such invasive systems may also be a problem. Finally, there is a need for capture systems of detailed human motion, which also includes

(a) mechanical MoCap system | (b) electromagnetic MoCap system | (c) inertial MoCap system | (d) LED based MoCap system

Figure 11.1: Examples of commercial marker-based and invasive motion capture systems: (a) Gypsy^{TM}—mechanical motion capture, (b) magnetic motion capture, (c) Xsens MVN—inertial motion capture, (d) LED-based optical motion capture.

non-rigid surface motion such as is caused by tissue or garment, while marker-based systems capture only articulated motion. Markerless motion capture approaches that use only video cameras or depth sensors (Chapter 4) provide a solution to these problems. Although they still lack the accuracy of marker-based systems, there has been substantial progress in recent years, promising less restrictive motion reconstruction.

This chapter concentrates on markerless motion capture using multi-view video cameras (Chapter 2), and multiple depth cameras (Chapter 4). After decades of research efforts, many kinds of markerless motion capture techniques have emerged for different application scenarios. From a technical aspect, markerless motion capture can be mainly classified into two categories: discriminative approaches [Ganapathi et al. 10, Shotton et al. 11] and generative approaches [Bregler et al. 04, Deutscher et al. 00, Gall et al. 10, Gall et al. 09].

Discriminative approaches rely on data-driven machine learning strategies to convert the motion capture problem into a regression or pose classification problem. Although learning approaches require sufficient training data and lack the accuracy of generative approaches, they are able to estimate the pose from a single frame in real-time and do not require an initial pose estimate. They are therefore currently the best choice for applications like human-computer interfaces or games where reliability is more important than accuracy.

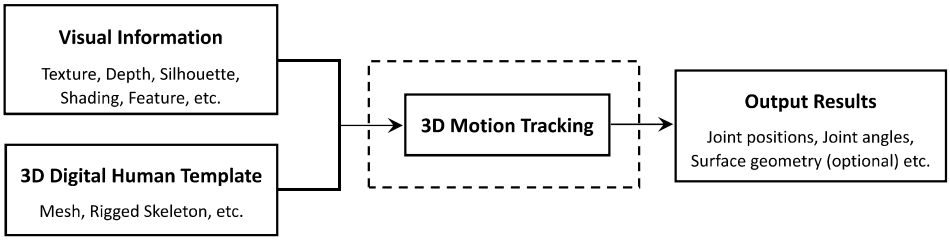


Figure 11.2: A general framework for template-based markerless motion capture. The module in the dashed box is the key processing step for motion reconstruction.

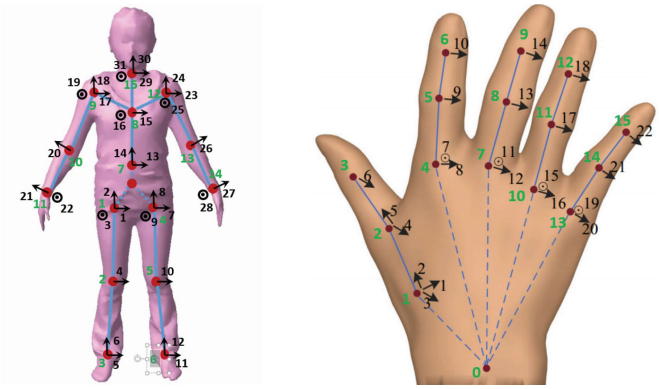
In contrast, generative approaches such as [Gall et al. 09], often rely on temporal information and solve a tracking problem. Many of these approaches parameterize the high dimensional digital human body by a low dimensional skeleton embedded in a template shape. Based on this template, an analysis-by-synthesis approach is started which optimizes the skeletal pose, and optionally the surface geometry, such that the synthesized human body images are consistent with the observed images. Figure 11.2 shows the generalized processing framework of template-based generative motion capture methods. Generative approaches are the preferred choice when very accurate results or motion details beyond joint positions are required. However, they require an initial pose, struggle to recover from tracking errors, and often rely on a user specific model. Consequently, some methods combine these two strategies [Wei et al. 12] where the initial pose is estimated by a discriminative approach and the estimated pose is then refined by a generative approach. This chapter mainly focuses on the discussion of generative approaches for markerless motion capture.

Despite a lot of progress in markerless motion capture in recent years, there are still many challenges remaining. The two major challenges for generative approaches to be addressed are how to capture motion of very high precision and how to achieve robust motion capture under general and uncontrolled capture settings.

One of the key factors to achieve high accuracy is the number and the precision of visual features that can be associated with the template model. While marker-based approaches only need to detect sparse but very reliable features, detailed fine-grained motion can only be captured by dense features. However, reliable dense features from images or depth data are very difficult to extract and the development of methods for this purpose is still a very active area of research. More details on 4D reconstruction using spatio-temporal features can be found in Chapter 12. Another important factor to achieve high accuracy is the quality of the template model [Liu

et al. 13]. When it is not only the reconstruction of the skeletal motion that matters, but also the fine-scale rigid and non-rigid surface deformation, both surface and skeleton models need to be accurate and sufficiently fine-grained, and both need to be properly coupled with each other. Information on how to embed the skeleton into the template mesh model can be found in Chapter 13. The proper design and reliable solution of the objective function are undoubtedly the other two key factors to achieve high tracking accuracy and robustness. Usually, the objective function is defined as a data term measuring the consistency between the hypothesized pose and the observed image data. The data term is commonly extended by other terms that regularize the objective function to increase the robustness to noise or occlusions. An example of such a regularizer is a skeletal pose prior that prefers poses that are more likely to occur. The development of efficient algorithms for optimizing these analysis-by-synthesis energy functionals is an important research question in computer vision. Optimizers are challenged by the fact that these functions are usually non-convex and non-smooth, and for various reasons analytic derivatives may not be available. Commonly proposed optimization strategies for these functions can be split into local and global approaches. While local optimization is fast, it can easily erroneously converge to local optima if not properly initialized. Global optimization methods avoid this problem, but normally have a much higher computational cost.

Motion capture under general and uncontrolled settings presents another challenge. Early generative markerless motion capture approaches require tens of calibrated cameras, hardware camera synchronization, green screen background, and controlled indoor lighting [Moeslund et al. 06]. With the progress of camera technology, especially the emergence of depth cameras, outdoor markerless motion capture with more convenient setups becomes feasible and is thus intensively investigated in the research community. State-of-the-art motion capture techniques that have been shown to succeed under less controlled capture settings include motion capture algorithms using a single [Baak et al. 13, Wei et al. 12] or multiple handheld depth cameras [Ye et al. 12], methods using input from binocular video [Wu et al. 13], as well as approaches that can handle unsynchronized multi-view video [Elhayek et al. 12]. Other methods have demonstrated successful motion capture in outdoor environments [Hasler et al. 09b], or under time varying illumination in the scene [Wu et al. 12a, Li et al. 13]. These new techniques thus take important steps toward removing the need of complicated acquisition setups, and thus greatly advance practical usability of markerless motion capture. In the following, this chapter introduces fundamental algorithmic concepts used in core components of the above generative motion capture systems.



(a) skeleton in the human body (b) skeleton in the hand

Figure 11.3: Mesh templates and skeletons for the whole human body and the hand. The red dots are the joints and green numbers are the joint indexes; the black arrows and black numbers are the DoFs and their indices.

The chapter continues with a brief introduction of basic background concepts employed in generative markerless motion capture, including human skeleton parameterization and mesh skinning in Section 11.2. Two example algorithms for generative motion capture of one human actor, namely an approach based on local optimization and an approach based on global optimization, are presented in Section 11.3. Subsequently, Section 11.4 explains what challenges need to be solved for motion capture of multiple closely interacting subjects. In Section 11.5, it is explained how concepts developed for full body motion capture can be employed to capture the motion of hands interacting with objects.

11.2 Kinematic Skeleton and Skinning

A surface mesh template with its embedded skeleton is a widely used data structure to model the tracked subject in markerless motion capture. The mesh template can be a generic model, such as the SCAPE human shape model that is fitted to the subject, or it can be a laser-scanned mesh model with detailed surface geometry (Chapter 14). The process of embedding a skeletal structure into the mesh template, such that the surface deforms plausibly with the skeleton motion, is commonly referred to as "rigging" (Chapter 13).

Generally, the skeleton model is represented as a tree structure (Figure 11.3). In this figure, the nodes in the tree, which represent joints, are marked as red dots. Each joint represents a possible rigid body transformation, and it can have several degrees of freedom (DoFs). In the figure,

these DoFs are marked as black arrows to indicate the rotation axes of each joint. For example, the joint of the wrist has 2 rotational DoFs while the joint of the neck has 3 rotational DoFs. Henceforth, the number of joints in the skeleton is denoted by N and the number of DoFs by M.

The rigid motion of a joint can be represented by a "twist" [Murray et al. 94], which can be mathematically expressed in two ways. The first representation is in the form of a 6D vector

$$\xi = (v_1, v_2, v_3, \omega_x, \omega_y, \omega_z)^T, \tag{11.1}$$

and the other representation takes the form of a 4×4 matrix

$$\hat{\xi} = \begin{bmatrix} 0 & -\omega_z & \omega_y & v_1 \\ \omega_z & 0 & -\omega_x & v_2 \\ -\omega_y & \omega_x & 0 & v_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{11.2}$$

In Eqs.(11.1) and (11.2), $\omega_\xi = (\omega_x, \omega_y, \omega_z)^T$ is the unit vector that points in the direction of the rotation axis. The vector $\mathbf{v}_\xi = (v_1, v_2, v_3)^T$ is the cross product of the rotation center and the rotation axis, i.e., $\mathbf{v}_\xi = \mathbf{p}_\xi \times \omega_\xi$. and $\mathbf{p}_\xi$ is the position of the joint in 3D space. If a joint has only one DoF, the 3D rigid motion transform can be described by the rotation angle θ, i.e., $\theta\hat{\xi}$, and similarly more rotation angles are needed to parameterize the transformation at a joint with more DoFs.

The twist obeys the cascade property [Murray et al. 94] in the skeleton, namely, for skeleton joint i, all of its parent joints (according to the tree hierarchy) affect its motion. Therefore, the combined transformation of joint i is represented as a matrix $\mathbf{T}_i$, which takes all the transformations of its parent joints into consideration,

$$\mathbf{T}_i = e^{\theta_0 \hat{\xi}_0} \cdot \prod_{j \in Parent(i), j \neq 0} e^{\theta_j \hat{\xi}_j}, \tag{11.3}$$

where j is the index for the DoF, $Parent(i)$ the set of all the parent DoFs of joint i, θ_j the rotation angle of the jth DoF and $\hat{\xi}_j$ the rotation matrix corresponding to the jth DoF. For example, in Figure 11.3(a) the $10th$ joint is parameterized by the $20th$ DoF in the model, and $Parent(10) = \{13, 14, 15, 16, 17, 18, 19\}$. Here, note that $\theta_0\hat{\xi}_0$ is the global translation and rotation of the whole human body. Eq.(11.3) can be linearized by Taylor expansion to yield

$$\mathbf{T}_i = \mathbf{I} + \theta_0\hat{\xi}_0 + \sum_{j \in Parent(i), j \neq 0} \theta_j \hat{\xi}_j, \tag{11.4}$$

where $\mathbf{I}$ is the identity matrix.

Given a new pose parameter set (i.e., value assignment to each DoF) of the skeleton, the deformed surface geometry that corresponds to the new skeleton pose can be computed by a technique called "skinning." One widely used skinning technique is the linear blend skinning (LBS) operator [Lewis et al. 00]. The main idea of LBS is that the position of each vertex on the surface model depends linearly on the transformation matrices of the skeleton joints

$$\mathbf{v} = \left(\sum_{i=1}^{N} \omega_i \mathbf{T}_i \right) \tilde{\mathbf{v}} \tag{11.5}$$

Here, $\tilde{\mathbf{v}}$ and $\mathbf{v}$ are the positions of a vertex on the surface of the 3D model before and after the deformation; ω_i is the precomputed skinning weight of the surface vertex with respect to the skeleton joint i with $\sum_{i=1}^{N} \omega_i = 1$. Intuitively, a skinning weight describes if and how strongly the transformation of a joint i influences the deformation of the vertex on the surface. Assigning skinning weights to vertices on the surface is a process supported by most standard computer animation packages through some form of painting interface. But there are also automatic methods to compute the weights [Baran and Popović 07]. Using Eq.(11.4), Eq.(11.5) can be rewritten as

$$\mathbf{v} = \left(\mathbf{I} + \theta_0 \hat{\xi}_0 + \sum_{m=1}^{M} \left(\sum_{j \in Children(m)} \omega_j \right) \theta_m \hat{\xi}_m \right) \tilde{\mathbf{v}}, \tag{11.6}$$

where $Children(m)$ is the set of all the children (joints) of the joint corresponding to the mth degree of freedom. By defining $\bar{\omega}_m = \sum_{j \in Children(m)} \omega_j$, Eq.(11.6) can be simplified to

$$\left(\theta_0 \hat{\xi}_0 + \sum_{m=1}^{M} \bar{\omega}_m \theta_m \hat{\xi}_m \right) \tilde{\mathbf{v}} = \mathbf{v} - \tilde{\mathbf{v}}. \tag{11.7}$$

Assuming $\mathbf{v} = (x, y, z, 1)^T$, $\tilde{\mathbf{v}} = (\tilde{x}, \tilde{y}, \tilde{z}, 1)^T$, Eq.(11.7) can be further written as

$$\mathbf{A}\boldsymbol{\Theta} = \mathbf{b} \tag{11.8}$$

where

$$\boldsymbol{\Theta} = (\theta_0 v_1^0, \theta_1 v_2^0, \theta_1 v_3^0, \theta_1 \omega_x^0, \theta_1 \omega_y^0, \theta_1 \omega_z^0, \theta_1, \cdots, \theta_M)^T,$$

$$\mathbf{b} = (x - \tilde{x}, y - \tilde{y}, z - \tilde{z})^T,$$

and

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & z & -y \\ 0 & 1 & 0 & -z & 0 & x \\ 0 & 0 & 1 & y & -x & 0 \end{bmatrix} \begin{bmatrix} \\ \mathbf{e}_1 \\ \\ \end{bmatrix} \begin{matrix} \cdots \\ \cdots \\ \cdots \end{matrix} \begin{bmatrix} \\ \mathbf{e}_M \\ \\ \end{bmatrix} \quad \text{with}$$

$$\mathbf{e}_1 = \begin{bmatrix} (v_{1,1} - \omega_{z,1}y + \omega_{y,1}z)\bar{\omega}_1 \\ (v_{2,1} + \omega_{z,1}x - \omega_{x,1}z)\bar{\omega}_1 \\ (v_{3,1} - \omega_{y,1}x + \omega_{x,1}y)\bar{\omega}_1 \end{bmatrix}, \mathbf{e}_M = \begin{bmatrix} (v_{1,M} - \omega_{z,M}y + \omega_{y,M}z)\bar{\omega}_M \\ (v_{2,M} + \omega_{z,M}x - \omega_{x,M}z)\bar{\omega}_M \\ (v_{3,M} - \omega_{y,M}x + \omega_{x,M}y)\bar{\omega}_M \end{bmatrix}.$$

In this way, all the vertices on the surface model can be linearized by M rotational degrees of freedom of the skeleton nodes. Here,

$$\xi_m = (v_{1,m}, v_{2,m}, v_{3,m}, \omega_{x,m}, \omega_{y,m}, \omega_{z,m}),$$

and

$$\mathbf{\Theta} = (\theta_0\xi_0, \theta_1, \cdots, \theta_M)^T, \tag{11.9}$$

Finally, Eq.(11.5) can be reformulated as

$$\mathbf{v} = \mathbf{T}(\Theta)\tilde{\mathbf{v}}. \tag{11.10}$$

where $\mathbf{T}$ maps an input pose $\mathbf{\Theta}$ to a linear deformation function, which further maps the vertex coordinates $\tilde{\mathbf{v}}$ to new coordinates $\mathbf{v}$.

11.3 Pose Optimization

The modeling of the energy function is essential to the success of the generative skeleton pose optimization. The energy function E to be optimized can be generally described by the sum of a data term E_D and a smoothness term E_S:

$$E(\mathbf{\Theta}) = E_D(\mathbf{\Theta}) + \gamma\, E_S(\mathbf{\Theta}). \tag{11.11}$$

Here, E_D is a data term that measures the difference between the synthesized human model and the observed image data. E_S is a pose prior that penalizes poses that are not physically possible (such as intersections) [Oikonomidis et al. 11b], that are dissimilar to some prior motions in a motion database [Martin and Crowley 95], or that are inconsistent with a predicted pose from the temporal sequence [Gall et al. 09]. The term γ is a weighting factor trading off between the data term and the smoothness term. Vision cues, such as silhouette, texture, edges, features, and estimated depth can all be integrated in the objective function to guide the optimization. Depending on the method used to solve for $\mathbf{\Theta}$, the generative motion capture approaches can be categorized into methods based on local pose optimization and global pose optimization.

Local Pose Optimization

Local optimization methods aim at fast computation of the desired $\mathbf{\Theta}$ and can only guarantee local convergence, therefore require a good initialization. Given the estimated skeleton pose $\tilde{\mathbf{\Theta}}$ and the deformed mesh model

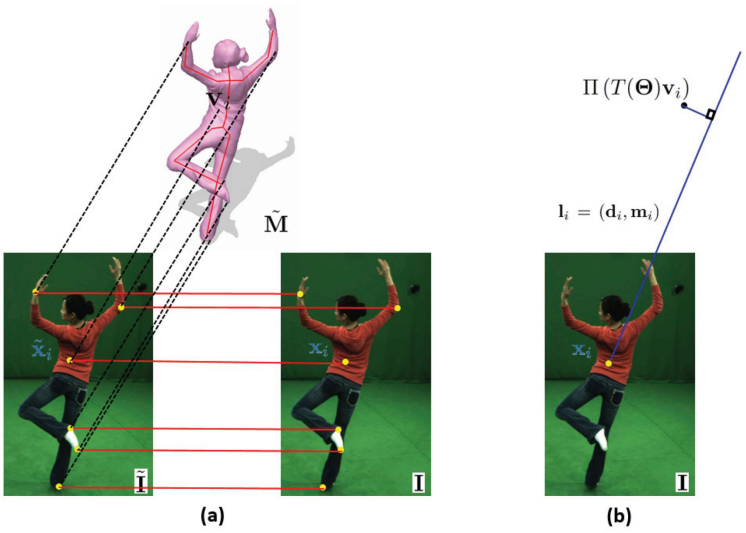


Figure 11.4: (a) 3D-to-2D alignment using SIFT matching. (b) The concept of a Plücker line.

$\tilde{\mathbf{M}}$ in a previous (multi-view video) frame, local pose optimization methods search for the best pose $\boldsymbol{\Theta}$ of the current frame in the vicinity of $\tilde{\boldsymbol{\Theta}}$. For instance, temporal cues can be used [Bregler et al. 04]. Such temporal cues are usually obtained by feature matching.

Specifically, for multi-view video input, the feature alignment is a 3D-to-2D alignment, namely, a correspondence set between the 3D model vertices $\mathbf{v}$ on the former tracked model $\tilde{\mathbf{M}}$ and 2D image pixels $\mathbf{x}$ in the current frame $\mathbf{I}$. Silhouette contour pixels and salient texture pixels are two commonly used features [Gall et al. 09]. In both cases, the 2D feature pixels $\mathbf{x}$ in the current frame are associated with a projected model vertex $\mathbf{v}$ yielding a 3D-to-2D correspondence $(\mathbf{v}, \mathbf{x})$. In the contour case, a contour vertex $\mathbf{v}_i$ on $\tilde{\mathbf{M}}$ is projected to the image plane forming a contour pixel $\tilde{\mathbf{x}}_i$, and the contour pixel $\mathbf{x}_i$ in image $\mathbf{I}$ which is closest to $\tilde{\mathbf{x}}_i$ is selected to form a correspondence $(\mathbf{v}_i, \mathbf{x}_i)$. In the texture case (Figure 11.4(a)), 2D-to-2D correspondences $(\tilde{\mathbf{x}}, \mathbf{x})$ between the former tracked frame $\tilde{\mathbf{I}}$ and the current frame $\mathbf{I}$ are first obtained by matching SIFT features [Lowe 04], and then model vertices $\mathbf{v}$ which are projected to $\tilde{\mathbf{x}}$ are associated with $\mathbf{x}$ to form the correspondence set.

Given the correspondence set $(\mathbf{v}, \mathbf{x})$, the projection of a deformed 3D point $T(\Theta)\mathbf{v}$ to the image coordinates should be as close to $\mathbf{x}$ as possible. To model this, the concept of a Plücker line [Stolfi 91] is used. A Plücker line $\mathbf{l} = (\mathbf{d}, \mathbf{m})$ is determined by a unit vector $\mathbf{d}$ and a moment $\mathbf{m}$, where $\mathbf{x} \times \mathbf{d} - \mathbf{m} = 0$ for all points $\mathbf{x}$ on the line. Since each 2D point $\mathbf{x}_i$ defines a projection ray that can be represented as Plücker line $\mathbf{l}_i = (\mathbf{d}_i, \mathbf{m}_i)$, the

error of a pair $(T(\boldsymbol{\Theta})\mathbf{v}_i, \mathbf{x}_i)$ is given by the norm of the perpendicular vector between the line $\mathbf{l}_i$ and the transformed point $T(\Theta)\mathbf{v}_i$ (Figure 11.4(b)):

$$E_d(\boldsymbol{\Theta}) = \sum_i w_i \left\| \Pi \left(T(\boldsymbol{\Theta})\mathbf{v}_i \right) \times \mathbf{d}_i - \mathbf{m}_i \right\|_2^2, \tag{11.12}$$

where Π denotes the projection from homogeneous coordinates to non-homogeneous coordinates and w_i are the weights for the correspondences. These weights can be used to assign a confidence to each correspondence, for instance in order to give lower confidence to less trustworthy evidence. The objective can be locally minimized by an iterative Gauss–Newton scheme using the linear Eq.(11.8).

When depth cameras are used as opposed to video cameras for marker-less motion capture, the feature alignment becomes a 3D-to-3D alignment between mesh model vertices $\mathbf{v}$ and 3D points (derived from the depth data) $\mathbf{p}$. The data term of Eq.(11.11) is then formulated as in Ref. [Ye et al. 12]

$$E_d(\boldsymbol{\Theta}) = \sum_i w_i \left\| T(\boldsymbol{\Theta})\mathbf{v}_i - \mathbf{p}_i \right\|_2^2. \tag{11.13}$$

Similar to Eq.(11.12), the energy function can be linearized and locally optimized using an iterative Gauss–Newton scheme. If both video data and depth data are available, Eqs.(11.12) and (11.13) can be combined to obtain a more informative energy term.

In order to stabilize the optimization, the linear system is regularized by the smoothness term in Eq.(11.11) defined as

$$E_S(\boldsymbol{\Theta}) = \left\| \boldsymbol{\Theta} - \tilde{\boldsymbol{\Theta}} \right\|_2^2. \tag{11.14}$$

This smoothness prior introduces a constraint by penalizing deviations from the previous pose $\tilde{\boldsymbol{\Theta}}$. In practice, feature alignment and energy optimization are usually iterated for about 10 to 20 times. For each iteration, the weights w_i can vary depending on the alignment confidence. The combined computation time for local pose optimization is low, and it takes less than one or two seconds to estimate the pose for a single time frame, even when using unoptimized code.

Global Pose Optimization

Compared with local pose optimization, global pose optimization defines the error measurement (11.11) between the input and the rendered images in a more direct and accurate way. The energy function is therefore usually non-linear, which demands time-consuming optimization like particle swarm optimization (PSO) [Oikonomidis et al. 11a] or interacting and annealing particle filters (ISA) [Gall et al. 10] for solving the optimal pose in a global manner.

Figure 11.5: Silhouette term. (a) The original image; (b) the extracted silhouette map $\mathbf{S}_{i,v}$; (c) the synthesized silhouette map $\mathbf{S}_{r,v}$ automatically generated by the rendering process.

Since the energy function is not linearized, there is more flexibility in designing the objective function. For example, Eq.(11.11) can be defined as the sum of a silhouette term, a texture term, and a depth term, as

$$E_D(\boldsymbol{\Theta}) = \delta_{silh} E_{silh} + \delta_{tex} E_{tex} + \delta_{depth} E_{depth}. \tag{11.15}$$

Here, features are not matched to obtain correspondences, but each term in Eq.(11.15) measures by a more direct analysis-by-synthesis strategy the consistency of images data and current estimate [Gall et al. 09].

Silhouette Term This term measures the discrepancy between silhouette maps of the synthesized images and the silhouette maps extracted from the observed images (Figure 11.5). A silhouette image $\mathbf{S}$ is represented as an image whose foreground and background pixels are set to one and zero, respectively. The mismatch between the segmented silhouette $\mathbf{S}_i$ of the human in the foreground and the rendered silhouette $\mathbf{S}_r$ of its hypothesis model is measured pixel-wise for each camera view v by

$$E_{silh} = \sum_v \left(\frac{\sum \left(\mathbf{S}_{i,v} \cap \bar{\mathbf{S}}_{r,v} \right)}{\sum \mathbf{S}_{i,v}} + \frac{\sum \left(\mathbf{S}_{r,v} \cap \bar{\mathbf{S}}_{i,v} \right)}{\sum \mathbf{S}_{r,v}} \right). \tag{11.16}$$

The measurement is a bidirectional distance function minimizing the area of non-overlapping foreground regions between the segmented and synthesized silhouette images. Here, $\sum S$ sums the values of all binary pixels of S, and $\bar{\mathbf{S}} = 1 - \mathbf{S}$ is the inverse of the binary silhouette image.

Texture Term The texture differences between the synthesized image data and the observed image data are the most direct measurement, and can be defined as

$$E_{tex} = \sum_v \left(\sum_i D\left(\mathbf{W}_i\left(\mathbf{R}_v\right), \mathbf{W}_i\left(\mathbf{I}_v\right) \right) \right). \tag{11.17}$$

Specifically, for each pixel i in the rendered image $\mathbf{R}_v$, the difference D between a window $\mathbf{W}$ centered around pixel i in $\mathbf{R}_v$ and a window with the same position in the captured image $\mathbf{I}_v$ is computed. One of the measurement metrics for D is the zero-mean normalized cross-correlation (ZNCC) [Martin and Crowley 95].

The texture of the human model can be obtained in the first frame of a video and slowly updated over time. Without updates the texture remains constant over time and does not handle appearance changes or invisible parts in the first frame. If the texture is updated too frequently, tracking errors may occur due to accumulation of small errors that lead to drift.

Depth Term A depth term can be incorporated when a depth camera or a binocular video camera for stereo matching is used. The term measures the consistency of the observed 3D data and the current 3D model by pixel-wise comparison for each projected surface point of M:

$$E_{depth} = \sum_{v} \left(\sum_{i \in S_{r,v}} \| M(i) - P(i) \|_2 \right). \tag{11.18}$$

$\mathbf{S}_{r,v}$ is the projected silhouette of M.

A smoothness prior, as defined in Eq.(11.14), can also be added. The objective function can be optimized by interacting simulated annealing (ISA) [Gall et al. 09]. Although the optimization can be hastened by GPU computing [Shaheen et al. 09], it is still considerably slower than the previously described local optimization algorithm. The reason for this is that particle-based optimizers, such as ISA, require the evaluation of the pose energy function for a very high number of particles, where each particle is one pose hypothesis, i.e., one possible set of skeletal pose parameters. Local optimization, however, frequently gets stuck in local minima if the tracker is not initialized with a pose hypothesis sufficiently close to the true optimum. Therefore, an approach that combines local and global pose optimization is proposed in [Gall et al. 09] to achieve efficient and high-quality skeletal motion capture. Briefly, local optimization is first operated for each processing frame, followed by global optimization only for body parts with a high residual error. Figure 11.6 shows the results of using this joint optimization strategy.

11.4 Multi-Person Motion Capture

Many types of human motion can only be observed during close interaction between two or more humans. Examples are found in many types of sport, for instance wrestling, judo, or ballroom dancing. In these motions,

Figure 11.6: Markerless motion capture results with joint local and global optimization as proposed in [Gall et al. 09].

humans often move very close to each other, and there may be physical contact between them. From the perspective of markerless motion capture, this complicates the situation since the degree of pose ambiguity increases significantly. Commonly used features to determine the correct body pose, like silhouettes, color, edges, or interest points cannot be uniquely assigned to a person anymore. Due to frequent occlusions, these ambiguities become even more challenging when people interact closely. For these reasons, it is infeasible to directly apply single-person pose optimization algorithms that use such features, such as the one described earlier in this chapter, to the multi-person case. Attempting to use them and jointly optimize for the pose parameters of two body models will fail very quickly, since the model-to-data association is undetermined.

To successfully track multiple people in close interaction, [Liu et al. 11] and [Liu et al. 13] propose a method that employs a combination of tracking and robust segmentation of input multi-view video frames. The segmentation allows the method to generate separate silhouette contours and image features for each person. This way, a local pose optimization method, such as the one described earlier, can be applied in the multi-person case since model-to-data-association is known from segmentation. After each image pixel has been assigned to one of the observed persons or the background, pose estimation can be performed for each person independently (Section 11.3).

The core operation of the integrated tracking and segmentation algorithm is as follows. Before pose estimation on a new frame of multi-view

Figure 11.7: Multiview image segmentation results for a sequence with three subjects. The class labels for the three subjects are indicated by the colors red, green, and blue.

video commences, a label is assigned to each pixel in the foreground of the scene, denoting to which person the pixel belongs. This pixel assignment is computed by solving a discrete optimization problem which finds the most likely assignment of pixel labels based on a Markov random field (MRF) energy. The MRF energy formulation uses color information and a local regularization. But in addition to these appearance cues, it also uses shape cues of the scene that are derived from the reconstructed poses of all actors in the previous time step. The optimization generates high quality segmentation results even under serious occlusions and ambiguous appearance (Figure 11.7). In consequence, even under such challenging conditions, and even with three people in the scene, high-fidelity motion capture results can be obtained (Figure 11.8).

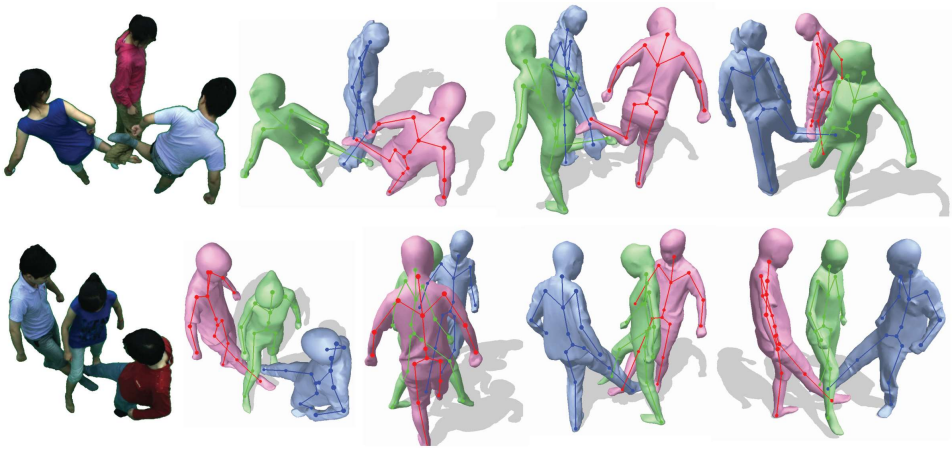


Figure 11.8: Multi-Person motion tracking results on two multi-view video sequences.

11.5 Capturing Hand Motion

Capturing the motion of hands is another important problem with many important applications in computer graphics, human computer interaction, and robotics. In principle, one could argue that hand motion can be captured by the same type of algorithms used for full body motion capture. In the end, the hand can be described by a kinematic skeleton itself that almost reaches the complexity of the full body. Markerless hand motion capture, however, is more challenging than full body motion capture. There are several reasons for this, for instance the fact that the hand has a very uniform appearance, and many fingers look very similar, which makes marker-free pose estimation challenging and ambiguous. Also, self-occlusions happen in many hand postures. Capturing the motion of one or two hands that are interacting with an object is an even more challenging task, because it requires not only the estimation of fine-grained hand articulation and object movement, but also subtle interactions and contact phenomena between the hand and the object. Two categories of approaches have been proposed recently to address this problem. One set of methods uses depth cameras as input sensors [Oikonomidis et al. 11b]; other algorithms resort to multi-view video input [Ballan et al. 12, Wang et al. 13b].

This section briefly describes the approach proposed in [Wang et al. 13b] for acquiring physically realistic hand motion from multi-view video while the hand is grasping and manipulating an object. The key idea of this approach is to use a composite physically based motion controller to simultaneously model hand articulation, object movement, and subtle interactions between the hand and the object. The motion controller of both hand and object motion is integrated in an analysis-by-synthesis framework, similar to the one described in Section 11.3.

Specifically, the global optimization model in Eq.(11.15) is parameterized by a motion controller

$$\underset{\mathcal{M}}{\operatorname{argmin}}\, E(\Theta(\mathcal{M})), \tag{11.19}$$

where $\mathcal{M}$ are the parameters of the controller. In this way, the pose of the hand is not directly parameterized in kinematic joint parameters (11.9), such as described earlier for the full body case, but indirectly through the control input of a physics-based dynamical motion model. Using such a physics-based dynamical model is advantageous in the case of hand motion capture, as it naturally models the subtle interactions between the hand and an object. In consequence, they can help to eliminate physically implausible motions or contact phenomena between the hand and the surfaces it interacts with. Another benefit is that the recovered motion controller can be utilized to conveniently adapt the captured motion to a new object with different geometry, which is an important requirement in many

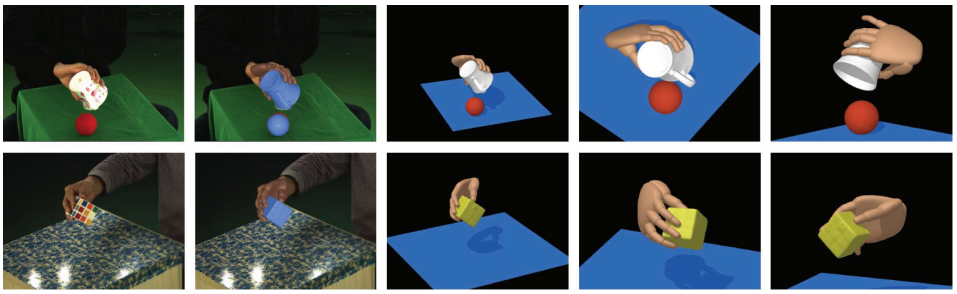


Figure 11.9: Video-based motion capture of hands that manipulate objects. Both rows show: input images, the reconstructed poses superimposed on the input images, the reconstructed poses from the same viewpoint, and the reconstructed poses from two different viewpoints (from left to right).

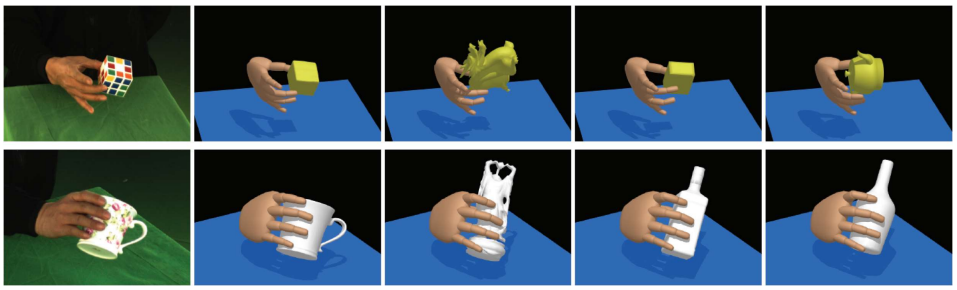


Figure 11.10: The captured motion controller is applied to animate the hand interacting with a new object with different geometry. Both rows from left to right show: the original image data, the captured pose of the hand and the object, and the controller applied to three different objects.

graphics and animation applications. Figures 11.9 and 11.10 show several results of hand motion capture and animation.

11.6 Summary

In this chapter the foundations of algorithms for markerless capture of human motion from multi-view video or depth data have been described. The focus was on generative approaches to the problem, and as examples, two methods have been discussed in more detail. Ongoing research in the field tries to further relax the constraints on situations under which the proposed methods are applicable. For instance, new methods for capturing the motion of multiple subjects or interactions with objects were developed. This imposes additional challenges and makes it necessary to combine generative approaches with physical scene models as well as discriminative approaches. There are several challenges to be addressed in the

future. The pose estimation accuracy of current generative and discriminative approaches is still not quite as high as the one achieved with intrusive marker-based approaches. Reducing the error to a comparable range of a few millimeters is necessary for applications that require accurate measurements. Although there are several approaches for outdoor motion capture, motion capture under any general lighting condition is still not possible at the moment. While discriminative approaches generalize well to other subjects, generative approaches often require each tracked subject to strike a special initial pose, such that a personalized template model can be fitted to the subject. Further improving the initialization methodology is thus an active area of research. Finally, real-time performance with low latency is required for interactive applications. This chapter has shown that several approaches already satisfy some of these requirements. However, an approach that satisfies all of them and is applicable in general real-world scenes is still not in sight.

12
Dynamic Geometry Reconstruction

Edmond Boyer, Adrian Hilton, and Céline Loscos

12.1 Introduction

Recent progress in shape modeling allows the recovery of precise shape models from visual data, such as color and depth images. These models can come in different representations, usually points, surfaces, or volumes, depending on the strategy employed for their estimation, e.g., stereo vision or sensor fusion, which were discussed in preceding chapters. When considering visual information over time, as in videos, temporal sequences of models can be obtained. Yet, whatever the representation, static reconstructions performed independently over time do not provide clues on the motion and the deformation of a shape. However, natural scenes are usually dynamic and composed of shapes that evolve over time, for instance humans and objects (Figure 12.1). Several applications are based on the capture of shape evolution over time: in digital content production, for instance, for realistic animation of virtual characters; or in motion analysis for sport or medical applications. The next step in modeling reality is therefore concerned with the ability to recover or capture motion and deformation of shapes of unknown types by using visual information sampled over time.

Image observations over time are discrete by construction and the capture of shape motion, also referred to as shape tracking, consists in the recovery of temporal correspondences between successive time instants. To this aim, a large variety of directions can be employed depending on the representation chosen for objects and on the prior information assumed for shapes and their deformations. For example, when scenes are composed of humans only, articulated motions can be assumed and represented by the poses of a skeleton (Chapter 11). For more general scenarios with less prior information, motion models must be less constrained. At the end of the spectrum, when no prior information on the observed objects is available,

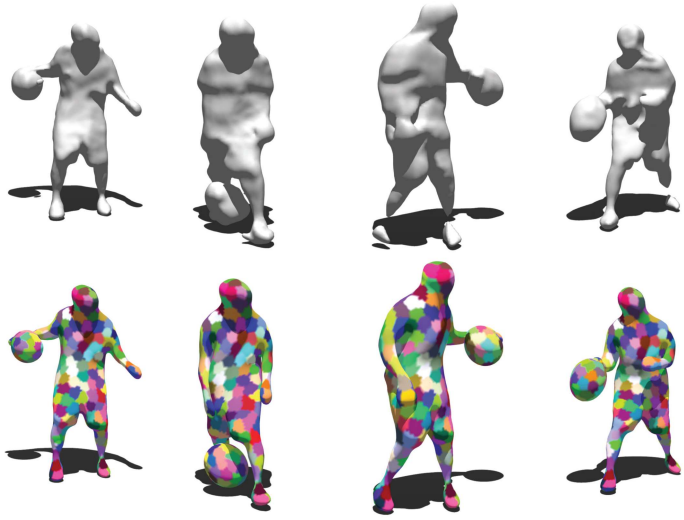


Figure 12.1: Example of captured dynamic scene: (top) independent reconstructed geometries; (bottom) registered shape model.

tracking reduces to finding the motion field between two successive time instants, i.e., matching, and global consistency over temporal shape models can hardly be enforced, and thus trajectory drifts along time sequences are often introduced. This is why numerous recent approaches assume a known surface model, such as a mesh, to start with. This allows the recovery of time consistent 3D models, also called 4D models, that encode both shape and motion information. 4D models include human models such as the templates presented in Chapter 11, but the concepts generalize to generic dynamic scenes.

In the following the strategies that exist in the literature to solve the tracking problem are explained. Different methods are exemplarily illustrated, starting with a mesh tracking approach that follows a popular sequential strategy and estimates the deformations of a mesh model between two time instants. Subsequently, a global strategy is described that can track a mesh model over several time sequences in an optimal and thus non-sequential way. Further on, an approach is presented that assumes little about the allowed deformation and that estimates motion fields. The chapter finishes with a discussion of current limitations of the existing approaches, as well as future research directions in the field.

12.2 Strategies

Existing strategies for shape tracking can be divided with respect to the amount of prior knowledge they assume and, consequently, to the constraints they impose on the observed shape. Probably the most widespread strategy consists in fitting a given deformable shape template to the observations at each time instant, which gives a 4D model (i.e., a time consistent 3D model). While providing robust approaches this strategy requires a deformation model that inherently limits the admissible observations, e.g., articulated movements only with a skeleton based model (Chapter 11). However, such limitation appears to be not critical in many applications and this strategy has received much attention in recent years as a result of the growing interest in human motion analysis. Another strategy worth mentioning consists in directly mapping observations from one frame to the next, which in turn yields motion trajectories over time. This approach requires less prior knowledge but, as a consequence, suffers more easily from drift over long sequences. In both cases the problem can be formulated as the estimation of motion parameters and the strategies can be classified with respect to the elements that feed into this estimation. To reduce the accumulation of drift errors for sequential tracking over long-sequences, and to allow alignment of tracked models across multiple sequences, non-sequential alignment approaches have been proposed. Non-sequential approaches align similar frames across the entire sequence to reduce the path length of sequential tracking.

Problem Formulation

Given observations $\mathcal{O}$, e.g., a point cloud or a mesh, at a given time instant, the problem is therefore to fit a given template, or to find a mapping from the previous time instant, to these observations. Existing methods usually formulate the problem as a maximum *a posteriori* (MAP) estimation of the deformation or mapping parameters $\hat{\Theta}$ that maximizes the posterior distribution $\mathcal{P}(\Theta|\mathcal{O})$ of the parameters Θ given the observations $\mathcal{O}$:

$$\hat{\Theta} = \arg\max_{\Theta} \; P(\Theta|\mathcal{O}) \simeq \arg\max_{\Theta} \; P(\mathcal{O}|\Theta) \; P(\Theta), \tag{12.1}$$

where $P(\mathcal{O}|\Theta)$ is the likelihood of the observations given the motion parameters and $\mathcal{P}(\Theta)$ is the prior information on these parameters. Taking the log of the above expression yields the following optimization problem:

$$\hat{\Theta} = \arg\max_{\Theta} \; E_{dt}(\mathcal{O}, \Theta) + E_r(\Theta), \tag{12.2}$$

where the data term E_{dt} corresponds to the log-likelihood and the regularization term E_r to the log-prior. Approaches then differ in the observations

$\mathcal{O}$ they consider, the parameterization Θ they use for the deformation, and the regularization E_r they impose over these parameters. Alternatives for these terms are discussed in the following.

Observations

Observations $\mathcal{O}$ are used, in the data term, in combination with a distance function to measure how good a motion prediction is. For instance Markers are used in traditional *mocap* systems to recover the pose of known skeletons. However, they only partially describe how shapes are moving, as they only describe the motion of sparse locations in space. To fully track shapes more densely, observations that vary from 2D image information to 3D geometric information can be considered.

Image Silhouettes Several model-based approaches deform a reference model so that the contour of the projected shape matches the contour of the observed silhouettes [Vlasic et al. 08, de Aguiar et al. 08b, Gall et al. 09, Straka et al. 12]. In these works, the silhouette overlap error is in general considered as a measure of the prediction error. This assumes that image silhouettes are accurate, which is not necessarily true, especially in unconstrained environments where foreground background discrimination is often difficult. In addition, due to the projections, large discrepancies along viewing lines can be missed. Nevertheless silhouettes are robust image primitives widely used for shape tracking.

Point Clouds Point clouds can, for instance, be obtained from range scanners or multi-view reconstructions methods, e.g., [Franco and Boyer 09, Furukawa and Ponce 10]. They are used as 3D observations in numerous approaches that typically proceed in two steps: (i) estimate correspondences between the model or the previous frame and the observed points; (ii) find the motion parameters that minimize distances between correspondents. The first step is either deterministic, as in the well known ICP method for rigid registration [Besl and McKay 92], or probabilistic, as in many recent approaches [Myronenko and Song 10, Horaud et al. 09, Cagniart et al. 10]. In the latter methods, observations are assumed to be drawn from Gaussian mixture distributions that are estimated with an EM strategy [Dempster et al. 77]. This allows the methods to explicitly handle outliers in the observations.

In the case where points are mesh vertices, shape matching methods that exploit local shape properties, such as [Anguelov et al. 04b, Starck and Hilton 05, Bronstein et al. 07, Varanasi et al. 08], could also be considered to find correspondences that would yet be deterministic. Besides,

photometric information, as available in images, can be considered as well when searching for correspondences [Starck and Hilton 07a, Gall et al. 09].

Motion Parameterization

Parameters Θ in the estimation (12.2) are used to parameterize the mapping from a template model, or from the previous frame, to the current observations. They come from the knowledge that is available on the motion and explicitly or hardly constrain the motion to belong to a specific category as opposed to the regularization term where prior assumptions on the motion parameters, such as smoothness, are softly enforced. As a simple example of this principle, mocap systems parameterize the poses of a known skeleton with joint angles that are optimized in order to minimize the distances between markers and joints at each time instant. In the general situation, and with shape models or with point clouds, motion can be assumed to be:

Unconstrained In which case motion parameters are directly the unconstrained vector field over all vertices or points. This field being initialized using tracked features, e.g., [Varanasi et al. 08, de Aguiar et al. 08b] or using normal flow constraints, e.g., [Petit et al. 11, Blache et al. 14] (Chapter 8).

Rigid In this category, the shape is assumed to move rigidly and the optimized motion parameters are those of a 3D rotation and a 3D translation. This is the case with the well known ICP approach [Besl and McKay 92] that alternates between: (i) associating observations to the shape model, i.e., defining distances between the observations and the shape; (ii) optimizing for the best rigid motion parameters that minimize these distances.

Articulated As in [Mundermann et al. 07, Vlasic et al. 08, Horaud et al. 09, Gall et al. 09] where the parameters of a skeleton are optimized (e.g., the joint angles). Since the skeleton is not observed, a surface model that surrounds it must be defined in order to verify whether the estimation satisfies the shape observations. This surface is, for example, composed of ellipsoids in [Horaud et al. 09] or a mesh whose vertex positions are obtained via blend skinning [Vlasic et al. 08, Gall et al. 09] (Chapter 11).

Locally Rigid As in [Cagniart et al. 10, Huang et al. 13] where vertices of a template mesh are grouped into patches, each of which is assumed to move rigidly. This motion parameterization is more flexible than the articulated model and allows therefore for more general shape motions such as cloth deformations. On the other hand, this local model requires additional global constraints (e.g., regularization constraints) to ensure consistency between locally rigid motions.

Learned In contrast to the previous generative motion models, a body of work follows a discriminative strategy that consists in recognizing the most likely pose of a shape within a pre-learned dataset and given the observations. This is the case, for example, in [Agarwal and Triggs 06, Anguelov et al. 05, Sigal et al. 12] or with the well known Kinect application [Shotton et al. 11].

Regularization

The data term in Eq.(12.2) is usually not sufficient to identify a unique set of motion parameters, especially with noisy observations, as is often the case in practical and real scenarios. In order to reduce the space of admissible solutions and disambiguate potential candidates, additional knowledge on the motion is required. This comes in the form of constraints on the motion parameters Θ such as:

Smoothness of Motion Field In the case of point clouds or shapes, an intuitive constraint is to enforce smoothness of the estimated motion field, as in [Myronenko and Song 10, Jian and Vemuri 11] with non-rigid point registration.

Shape Preservation There is a good deal of work that assumes the preservation of local shape properties during motion. These properties can be the Laplacian coordinates [Sorkine et al. 04] as in [Varanasi et al. 08, de Aguiar et al. 08b, Gall et al. 09, Petit et al. 11, Straka et al. 12] or other local rigidity terms, e.g., [Cagniart et al. 10, Huang et al. 13]. This regularization is often stronger than the previous smoothness assumption. Consequently, it sometimes induces tracking failures when the observations do not satisfy the constraint and regularization adjustments may be necessary in that case.

Learned Models In a way similar to the parameterization of motion, a few works take benefit of known tracking examples to learn the space of possible motions beforehand. For example [Duveau et al. 12] proposes a supervised strategy that regularizes the estimated motion parameters based on their learned distribution in a low-dimensional latent parameter space (Gaussian Process Latent Variable Models). Also using a non-sequential strategy for tracking, [Klaudiny et al. 12, Budd et al. 13] learn a shape similarity tree in order to ease shape matching.

12.3 Sequential Shape Tracking

In this section, an approach is illustrated that sequentially tracks a mesh model over a time sequence [Cagniart et al. 10]. The observations $\mathcal{O}$

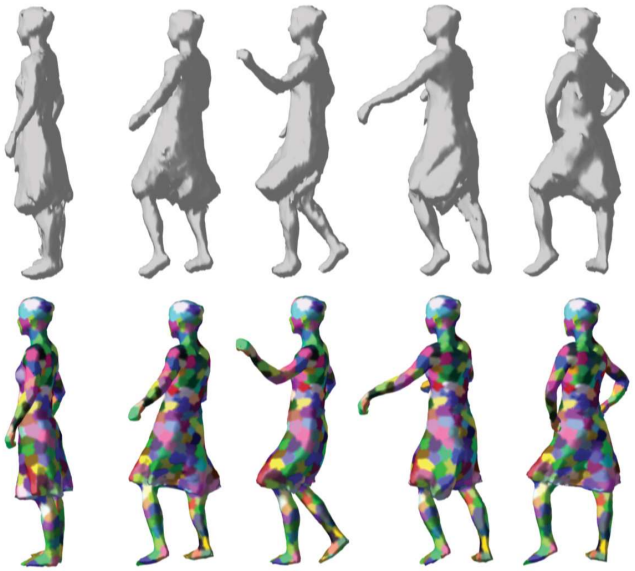


Figure 12.2: Illustration of sequential tracking: (top) input observations; (bottom) output tracked reference models.

over time are independent 3D meshes that are obtained with a multi-camera acquisition system and a silhouette-based reconstruction [Franco and Boyer 09]. The reference model is a triangle mesh that is usually one of the observed meshes. The approach adopts a locally rigid patch-based deformation model where vertices are grouped into N_p patches (Figure 12.2). While the approach is generic with respect to the shape category, it can be specialized through the prior (regularization) term to track a human surface model and its associated skeleton [Huang et al. 13]. In that case the triangle mesh model is combined with a skeleton model that is a tree structure of N_j nodes (3D joints) with the root set at the pelvis. It is rigged into the mesh using the `Pinocchio` software [Baran and Popović 07] that gives the associations between vertices and joints.

MAP Estimation

As explained earlier, the problem can be expressed as a MAP estimation of the motion parameters Θ given the observations $\mathcal{O}$. Here these parameters decomposed into mesh deformation parameters $\Theta_m = \{(\mathbf{R}_k, \mathbf{c}_k)\}_{k=1:N_p}$ that are the orientations and the locations of the k patches; and, when specialized to human shapes, skeleton pose parameters $\Theta_s = \{\mathbf{x}_j\}_{j=1:N_j}$

that are the 3D locations of the joints. Thus:

$$\hat{\Theta} = \underset{\Theta_m, \Theta_s}{\arg\max}\ P(\mathcal{O}|\Theta_m)\ P(\Theta_m, \Theta_s). \tag{12.3}$$

The skeleton is not observed in practice and the approach assumes that, given the mesh deformation parameters Θ_m, the joint locations Θ_s and the observations $\mathcal{O}$ are independent, i.e., $P(\mathcal{O}|\Theta_m, \Theta_s) = P(\mathcal{O}|\Theta_m)$. This means that the motion is parameterized with the patches only and that the skeleton is solely used as a regularization constraint. Taking the negative log of Eq.(12.3) the optimization becomes:

$$\hat{\Theta} = \underset{\Theta_m, \Theta_s}{\arg\min}\ E_{dt}(\mathcal{O}, \Theta_m) + E_r(\Theta_m, \Theta_s), \tag{12.4}$$

with $E_{dt}(\mathcal{O}, \Theta_m) = -\log P(\mathcal{O}|\Theta_m)$ and $E_r(\Theta_m, \Theta_s) = -\log P(\Theta_m, \Theta_s)$. The tracking is achieved by deforming the model on a frame-by-frame basis. That is, using $\hat{\Theta}^{t-1}$ as the initialization to solve Eq.(12.4) at frame t.

Data Term

The data term is not specialized and involves the patch parameterization only. Following [Cagniart et al. 10] the likelihood $P(\mathcal{O}|\Theta_m)$ is expressed using Gaussian mixture models (GMM) where an observed point p_i is associated to patches with respect to Gaussian distributions:

$$P(\mathcal{O}|\Theta_m) = \prod_i P(p_i|\Theta_m) = \prod_i \sum_{k=1}^{N_p} \Pi_k P(p_i|z_i = k, \Theta_m). \tag{12.5}$$

Here, z_i is the latent variable for each p_i: $z_i = k$ means that p_i is generated by the mixture component associated with the patch k. $\Pi_k = P(z_i = k|y_i, \Theta_m)$ are the mixture weights that are estimated alternatively with Θ_m using the expectation-maximization method [Dempster et al. 77]. The likelihood that p_i is generated by the k-th component is modeled as a multivariate Gaussian with mean located at the compatible vertex v_i^k (i.e., a vertex with a normal similar to p_i) on patch k closest to p_i and isometric covariances σ; and as a negligible value ϵ if there is no such compatible vertex:

$$P(p_i|z_i = k, \Theta_m) = \begin{cases} \mathcal{N}(\mathbf{p}_i|v_i^k, \sigma) & \text{if } v_i^k \text{ exists} \\ \epsilon & \text{otherwise.} \end{cases}$$

Interestingly, this model allows the algorithm to explicitly handle outliers through an additional virtual $N_p + 1$ component equipped with a uniform distribution $P(p_i|z_i = N_p + 1, \Theta_m) = cst$ [Cagniart et al. 10]. In this case, each observation has a constant probability to be generated from the virtual outlier patch which favors the association of distant observations to this outlier patch.

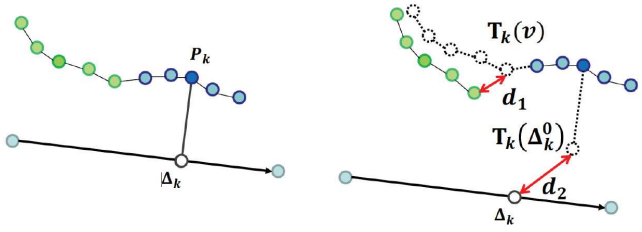


Figure 12.3: Illustration of the regularization: (left) original configuration; (right) distances being minimized in the optimization. d_1 impacts the mesh rigidity term E_m while d_2 impacts the bone-binding energy E_s.

Regularization Term

$P(\Theta_m, \Theta_s)$ decomposes into $P(\Theta_s|\Theta_m)P(\Theta_m)$. The regularization energy $E_r = -\ln P(\Theta_m, \Theta_s)$ is therefore a combination of a generic term $E_m = -\ln P(\Theta_m)$ that enforces rigidity between neighboring patches, and of an optional specialized term for humans $E_s(\Theta_s, \Theta_m) = -\ln P(\Theta_s|\Theta_m)$ that enforces the patches and the skeleton to satisfy binding constraints.

Mesh Rigidity E_m applies to the rigid transformation parameters of the patches and is defined over the set $\{(P_i, P_j)\}$ of neighboring patches [Cagniart et al. 10]:

$$E_m(\Theta_m) = \sum_{\{(P_i,P_j)\}} \sum_{v \in P_i \cup P_j} w_{ij} \| T_{\Theta_m(i)}(v) - T_{\Theta_m(j)}(v) \|,$$

where $T_{\Theta_m(k)}(v)$ is the predicted position of vertex v as transformed with patch k and w_{ij} weights the contribution of v with respect to its distances to the patch centers of P_i and P_j. This term favors similar rigid transformations between neighboring patches (Figure 12.3). Combined with the previous data term it allows to track arbitrary mesh model under the locally rigid assumption only (Figures 12.1 and 12.2). Additional prior knowledge about the scene, e.g., a human skeleton, can be exploited, when available, to further constraint the model evolution as explained in what follows.

Skeleton Binding Assuming a known skeleton for the tracked shape, a bone-binding energy $E_s(\Theta_s, \Theta_m) = -\ln P(\Theta_s|\Theta_m)$ is defined that builds on the approach introduced in [Straka et al. 12]. For each patch k the orthogonal projection of its center onto its associated bone is denoted Δ_k

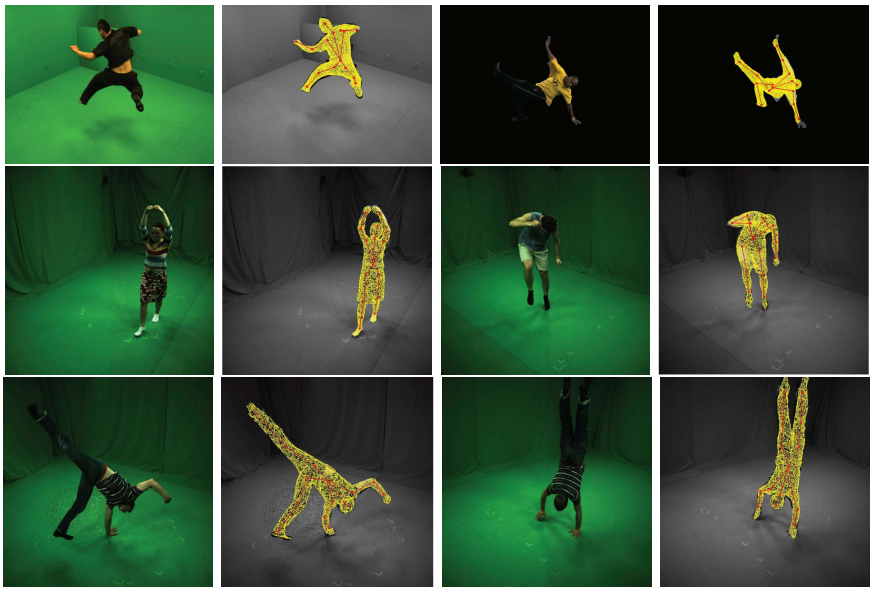


Figure 12.4: Illustration of the sequential approach on standard datasets: example frames of the input videos and the overlaid tracked models that include skeleton poses here.

(Figure 12.3). Let Δ_k^0 be the location of Δ_k on the reference model, then:

$$E_s(\Theta_s, \Theta_m) = \sum_{k=1}^{N_p} w_k \| T_{\Theta_m(k)}(\Delta_k^0) - \Delta_k \|,$$

where $T_{\Theta_m(k)}(\Delta_k^0)$ is the predicted position of Δ_k^0 with the transformation of patch k and Δ_k is the location of the patch center projection onto the skeleton with parameters Θ_s. The term w_k weighs the patch contribution such that patches close to joint locations have less influence. This term involves both the transformation parameters Θ_m and the skeleton joint locations Θ_s. It favors rigid motions of all the patches associated to the same bone in the skeleton. The approach involving the skeleton binding term is illustrated in Figure 12.4 for various standard datasets.

12.4 Non-Sequential Mesh Tracking

Sequential frame-to-frame mesh tracking, as presented in the previous section, is prone to drift over time, especially when large displacements occur. Consequently, non-sequential alignment strategies have been introduced to limit the accumulation of errors associated with sequential frame-to-frame

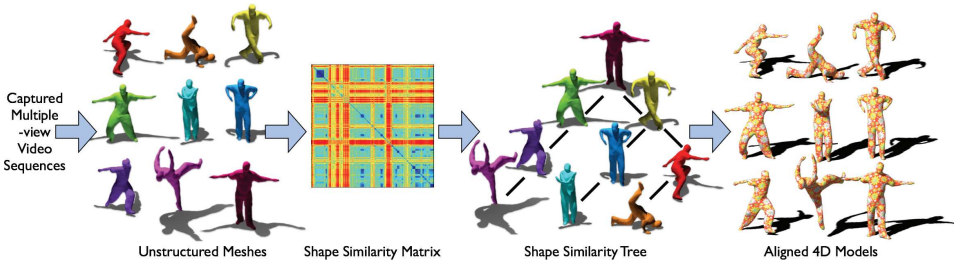


Figure 12.5: Overview of non-sequential global temporal mesh sequence alignment [Budd et al. 13].

mesh tracking by exploiting the similarity of frames across the entire sequence [Klaudiny et al. 12, Budd et al. 13, Beeler et al. 11]. Interestingly, this also allows the alignment across different sequences to produce 4D models from performance capture of different motions. Non-sequential approaches were first proposed in the context of reconstruction from video using structure-from-motion to reduce drift in reconstruction. In this case reconstruction is performed over sub-sequences of the video and fused into a single scene reconstruction using a hierarchical tree structure. This non-sequential reconstruction improves efficiency and reduces accumulation of reconstruction errors across the sequence.

For non-rigid mesh tracking Beeler et al. [Beeler et al. 11] presented an approach based on manually selected anchor frames to reduce drift for alignment of reconstructed non-rigid face sequences. The approach assumes that anchor frames are similar to a manually selected reference expression and are distributed across the sequence. Pairwise alignment of anchor frames with the reference reduces accumulation of errors by sequential alignment on shorter subsequences. This approach implicitly uses a non-sequential alignment based on a tree with branches from the reference pose to each of the anchor frame.

Shape similarity trees [Budd et al. 13] link frames with similar shape and motion providing a general representation for non-sequential alignment over one or more sequences. Non-sequential alignment changes the order in which frames are aligned to maximize the similarity of adjacent frames allowing existing pairwise sequential alignment algorithms to be applied while reducing drift and alignment failure. The branches of the shape similarity tree define the shortest path in similarity space between frames. This defines a non-sequential ordering of frames for alignment. Figure 12.5 presents an overview of the alignment process from raw unstructured mesh sequences to a single temporally coherent representation. Non-sequential alignment of a database of reconstructed mesh sequences based on the

Figure 12.6: Example frames for non-sequential alignment across multiple sequences of a StreetDancer dataset comprising 1800 frames.

shape similarity tree is performed in three stages:

1. **Shape Similarity Evaluation:** Evaluate the shape and motion similarity between all pairs of frames across all sequences.

2. **Shape Similarity Tree Construction:** Construct the minimum spanning tree given by the shortest path in shape similarity space for all frames.

3. **Global Non-rigid Alignment:** All frames for all sequences are aligned and re-meshed to have a single connectivity based on the shortest paths defined by the shape similarity tree and using, for instance, the patch based method presented in the previous section.

Figure 12.6 presents an example of non-sequential alignment to produce a 4D model from six sequences of a street dancer performing fast motion with loose clothing. The non-sequential alignment reduces drift and provides robustness to rapid motion by identifying optimal paths for tracking of mesh sequences. The approach links similar frames across the database rather than purely temporal adjacency. A potential drawback of non-sequential alignment is the independent accumulation for sequential alignment across different tree branches. The tree structure can be further optimized by clustering similar frames [Klaudiny et al. 12] to avoid short branches reducing alignment jitter due to fragmentation.

12.5 Motion Fields

As mentioned earlier, some approaches do not constrain motion through parameterization and, instead, estimate full motion fields over primitives, as in [de Aguiar et al. 07, Varanasi et al. 08, Petit et al. 11], and more recently in Blache et al. [Blache et al. 14]. In the latter approach the input is a temporal sequence of multi-view silhouettes transformed into a set of binary digital volumes using a visual hull reconstruction algorithm [Lucas et al. 13]. The visual hull is defined as the 3D intersection of the viewing cones associated to 2D silhouettes [Laurentini 94, Franco and Boyer 09].

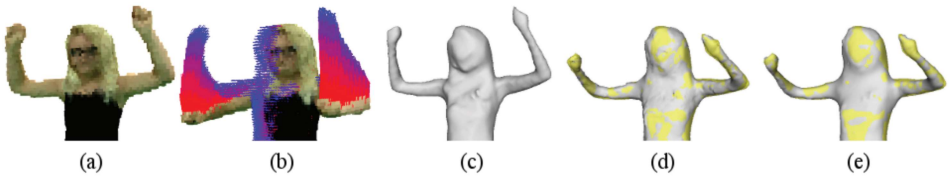


Figure 12.7: Starting from a colored volume sequence *(a)*, the motion flow is estimated *(b)*. The mesh of the first frame is used as a template *(c)*. After the deformation of this template by the motion flow *(d)*, the mesh is smoothed by a regularization algorithm *(e)* which matches the reconstruction with visual hull of the next pose (shown in yellow).

These binary digital volumes represent the observed shape at each video frame (Figure 12.7(a)). The method starts by computing a 3D motion flow between two consecutive frames, that extends the idea of optical flow described in Chapter 8 to 3D. This motion field is used in a subsequent step to deform a mesh model. Such a model can easily be obtained from any frame in the sequence by applying a surface reconstruction algorithm (such as marching cubes, Chapter 10) on the estimated volume at that frame. The tracking is then performed by deforming the mesh model at each time frame using the estimated flows.

Given two consecutive volumes V^t and V^{t+1} in the sequence, a motion field between the surface voxels $v_i^t \in V^t$ and $v_j^{t+1} \in V^{t+1}$ is determined. This 3D vector field is obtained by minimizing a distance function based on three criteria: a **proximity criterion** that accounts for the Euclidean distance between matching voxels, an **orientation criterion** that measures the difference between the normal vectors of matching voxels, and a **colorimetric criterion** that compares the photometric information between matching voxels. These criteria allow to match the voxels that correspond to the same part of the observed surface based on their proximity, orientation and appearance on the shape surface.

This estimation is performed backward and forward in time and each pair of matched voxels defines a 3D vector. The resulting 3D vector field describes the volumetric motion of the shape between t and $t+1$. However, the obtained motion field is corrupted by several inconsistent matches that remain. In order to reduce their effect, a Gaussian filtering step is applied on the 3D vectors which helps to obtain a coherent motion flow. This filtering is equivalent to a regularization of the motion field based on a smoothness assumption. In the final displacement field, each surface voxel is associated to a single motion vector (Figure 12.7(b)). In the second step of the approach, the template mesh is displaced along the motion field and each vertex is moved with respect to the closest vector. Already in previous sections, it was discussed that drift could lead to a deterioration

of the result which yields an irregular mesh (Figure 12.7(d)). Therefore, a regularization algorithm is also applied here to obtain a proper mesh that fits the reconstructed volume at the next frame (Figure 12.7(e)). The mesh is considered to be a deforming mechanical system and each vertex is subject to a set of forces as follows:

- A **spring force** where each incident edge applies a force on the vertex to enforce similar edge lengths. This tends to regularize the vertex distribution.
- A **smoothing force**, based on a Laplacian operator, that tends to smooth the mesh surface.
- A **matching force** that uses the Euclidean distance field to keep each vertex close to the object's surface.

When compared to more traditional 3D optical flow approaches, this strategy proves to be more robust over time, failing only when strong topological changes occur. Using a 3D uniform grid to represent volumes facilitates the motion flow estimation. Such estimation would be tedious with 3D meshes reconstructed independently over time, and therefore inconsistent (i.e., with different number of vertices and faces). The evaluation of the results demonstrated that introducing the photometry in the matching criteria significantly increases the accuracy. This algorithm fails when topological changes occur (such as body parts in contact) and several reference meshes should be considered in those cases. Having topological changes in 4D reconstruction is one of the big open research questions.

12.6 Summary

The chapter discusses methods to capture the motions and deformations of general shapes recorded by multi-view video systems. Strategies that exist differ with respect to three main criteria: (i) the observations taken into account, (ii) the motion parameterization, and (iii) the regularization over these parameters. One of the most popular related strategies that is illustrated in the chapter consists in tracking a known reference template model of the scene, usually a mesh obtained with a reconstruction method from one of the video frames. The success of this tracking strategy, either the sequential or the non-sequential variants that are discussed, is most certainly due to its ability to produce time consistent shape models, known as 4D models. However, while good and robust results have been obtained, many issues are still unresolved. This includes, for instance, the capture of dynamic scenes with evolving topologies or complex scenes with many interacting objects.

Part III

Modeling Reality

13
Rigging Captured Meshes

Kiran Varanasi and Edilson de Aguiar

13.1 Introduction

Performance capture methods discussed in the earlier chapters reconstruct the geometry of real-world objects in motion. For realizing any computer graphics application, practical user controls for editing and modifying the scene content are essential. Thus, a second step of developing a *control rig* for the captured geometry is necessary, through which the 3D geometry can be easily posed and manipulated.

In practice, computer animation artists commonly use three different types of rigs depending on the motion they wish to control. Non-rigid deformations over an approximately rigid bone-structure such as facial expressions are manipulated using a set of example *blendshapes* that are provided by the modeling artist. New facial expressions can then be generated through blending between different examples. Highly fluid deformations dependent on external forces such as cloth folds are generated using physics-based simulation. These simulations can be controlled by the artist through positional or force-based constraints. Finally, if the captured geometry is that of an articulated character model such as a human actor, an intuitive means for editing the pose of that character is through a skeleton rig. In principle, performance capture methods that reconstruct the 3D geometry of real-world deformations can be applied to each of these settings. In this chapter, the focus is on rigged articulated models such as human characters.

The skeleton rig for controlling the pose corresponds only loosely to the physiological skeleton of the bone joints. Instead, it is an abstraction for driving the surface deformation through a set of geometric transformations. Essentially, the skeleton is a hierarchy of joints around which a 3D rotation can be performed with respect to a pre-defined axis of rotation. The movements of a parent joint are cascaded down, affecting all the joints below in the hierarchy. For example, the movements of the shoulder affect the position of the elbow, the arm, and so on. The overall motion of the character is thus a superposition of the various joint movements. In character animation, forward kinematics refers to how the motion encoded by a set of 3D rotations around joints in the skeleton rig is decoded into

the overall motion of surface vertices of the character. Equivalently, inverse kinematics refers to how the 3D rotations around the joints in the skeleton rig can be inferred from the surface motion of vertices. To perform either of these steps, knowledge of the exact topology (i.e., hierarchy) of the character skeleton and of the relative positions of the joints in a neutral *rest* pose is required, as well as the mechanism through which the various surface vertices are affected by the skeleton joints. Conventionally in the character animation community, the first step is referred to as *rigging* and the second step as *skinning*. In this chapter, details about these steps are described to process 3D meshes captured from the real world.

Traditionally, for 3D meshes hand-sculpted by 3D artists, the rigging step involves bringing the input mesh into a neutral *rest* pose and carefully positioning the skeleton joints inside the interior volume of the 3D mesh, such that articulations around these joints generate plausible surface deformation results for the character. The skinning step involves partitioning the surface of the character into a set of body parts that correspond to *bones* linking joints in the skeletal hierarchy, as well as painting of *skinning weights* for each surface vertex that blend the effect of different joint transformations on that vertex. Typically, a surface vertex is affected by not more than $2-3$ bones. A skeleton rig offers an artistically intuitive and computationally efficient means for controlling the character motion. Despite its simplicity, plausible results for character animation can be achieved, although with immense artistic skill required in both the rigging and the skinning steps. Performance capture methods can greatly simplify these steps.

In this chapter, assuming that a performance capture system is deployed to produce a sequence of temporally aligned triangle meshes with the same number of vertices and mesh topology, three methods are described for building a rigged character out of such meshes.

1. Fitting a skeleton into a static mesh such that the bone joints fall along the medial axis of the 3D shape and are aligned with plausible articulation points (Section 13.3).

2. Converting a mesh animation sequence into a skeletal animation sequence by optimizing for the joint transformations and skinning weights (Section 13.4).

3. Building a deformable model by learning the residual deformations of each surface vertex observed in the captured meshes, which are beyond the skinned deformations due to rigid transformations of the bone joints (Section 13.5).

In the following, the problem context of editing 3D surface deformations

is introduced and viable solutions to the three problems mentioned above are presented.

13.2 Overview

Directly editing mesh animations It is possible to edit mesh animations without constructing a rig, by directly editing vertex positions. Kircher and Garland [Kircher and Garland 06] propose various such edits that align selected point trajectories to user-given locations. They parameterize the surface through dihedral angles between surface triangles, and deform the rest of the mesh surface such that local surface curvature and details are preserved. Sumner and Popovic [Sumner and Popović 04] propose an alternative motion parameterization through deformation gradients of mesh triangles. They are able to transfer the deformations across two different meshes with certain shape similarity. The deformation gradients can also be embedded into an automatically computed deformation graph structure [Sumner et al. 07] that can be edited. Hildebrandt et al. [Hildebrandt et al. 12] propose certain space-time editing operations that respect the modal energies of the surface. However, directly editing a mesh animation typically requires the user to place constraints on several vertices and thus needs a lot of edit-time. The range of edits that can be performed is also typically limited. In practical scenarios, an additional control rig that adapts to the motion of the object, and not just to its shape, is highly useful.

Surface deformation by rigs Skeleton rigs are a popular control structure for deforming articulated 3D objects [Lewis et al. 00]. Each vertex on the object's surface is associated to one or more skeletal bones and its motion is interpolated from the motion of the bones. The most common method for interpolation is known as *linear blend skinning*, where the vertex motion is computed as a weighted linear combination of rigid transformations of the bones [Magnenat-Thalmann et al. 88]. James and Twigg [James and Twigg 05] propose to use affine transformations on the bones to represent arbitrary surface motion, such as cloth deformation. Mean shift clustering is used to aggregate the vertex deformations into a set of bones. Kavan et al. [Kavan et al. 10] decompose the mesh animation into the relatively simpler linear blend skinning framework, and make use of the sparsity of the vertex weights to generate a fast animation. They use alternating least squares to optimize for the bone positions and skinning weights. Instead of a set of interior skeletal joints, the constraints for deforming a mesh can be provided through the vertices of an enclosing

cage. The mesh is then deformed according to how the volume enclosed by the cage compresses or expands. The mesh vertices are typically represented through generalized barycentric coordinates inside the polyhedra of the cage. Cages are sometimes more intuitive to edit for artists than interior skeletal handles, e.g., to show subtle non-rigid deformations on an object. For smooth deformation, the weighting functions that relate the mesh vertices to the cage should vary continuously and smoothly. Cauchy-green coordinates and bounded biharmonic weights are proposed as options for such weighting functions [Weber et al. 09]. Jacobson et al. [Jacobson et al. 12] propose a general framework for mesh deformation through a combination of control rigs: skeletons, cages, or point control handles. The user specifies only a subset of the deformation space and their method automatically infers the remaining degrees of freedom and estimates geometrically appropriate skinning weights that connect the surface vertices to the control rig.

Surface deformation by blendshapes Artists have traditionally handcrafted target meshes, known as blendshape targets, for representing various facial expressions. Novel facial expressions are then synthesized on the virtual face by interpolating between these blendshape targets or by mapping direct surface manipulation to blendshape interpolation [Lewis and Anjyo 10]. New facial dialogue is synthesized by interpolating between specific blendshape targets, known as visemes, that correspond to a selection of speech phonemes. Facial rigs are typically limited to synthesizing rigid motions such as the rotation of the head and the jaw movement, while the rest of the facial motion is synthesized exclusively through the skill of the artist who sculpts facial expressions at vertex-level detail. This has been necessary because human visual perception is highly attuned to artefacts in facial expressions. Indeed, the virtual face is deemed less likeable by human observers even as the fidelity of its facial geometry and expressions increases, in a widely understood phenomenon known as the *uncanny valley*. However, with increased resolution of facial scans and high-fidelity performance capture, this barrier is being overcome. Vlasic et al. [Vlasic et al. 05] propose a multi-linear model for varying facial expressions across the axes of identity, emotion, and visemes. They align the data from a set of static 3D scans of human faces to a common geometric template. It is now also possible to capture dynamic facial geometry at high resolution on a single mesh topology. Beeler et al. [Beeler et al. 11] propose a high-resolution passive facial performance capture system from multi-camera recordings. Data-driven deformable models can exploit this data to synthesize virtual facial expressions. Neumann et al. [Neumann et al. 13b] automatically decompose an input mesh animation into a set of sparse localized deformation

components, or *splocs*, that loosely correspond to blendshape targets that can be edited by artists for novel expression synthesis. Capturing and editing facial deformations for real-world visual effects is discussed in detail in Chapter 20. This chapter focuses on building rigged 3D models for articulated characters.

13.3 Fitting a Skeleton into a Static Mesh

Given a static 3D mesh as input, a skeleton structure can be extracted by successively thinning the shape [Gagvani and Silver 99]. For the purpose of character animation, the skeleton is a support structure of rigid line segments in the interior of the shape around which articulated movements can be executed. Teichmann and Teller [Teichmann and Teller 98] propose a method for semi-automatic recovery of the animation skeleton by simplifying the network structure of the Voronoi centers of the 3D shape using some user assistance. Since the precise motion semantics are not necessarily apparent from a static 3D mesh, automatically recovering an animation skeleton is an ill-posed problem. However, with certain assumptions, plausible animation results can be achieved. An informative geometric structure that matches our intuition about an underlying shape skeleton is the *medial axis* of the shape, which is the set of 3D points that are centers to medial balls which are equidistant to at least two different points on the shape's surface. The medial axis of a given shape is unique and completely informative about the shape. Several methods have been proposed for fast and robust computation of the medial axis given an input 3D shape [Chazal and Lieuter 05, Giesen et al. 09]. Although the medial axis is continuous for 2D shapes, it can be a set of disconnected sheets for 3D shapes. To remedy this problem, the curvilinear skeleton of the shape is proposed as a 3D curved line that connects a set of interior points [Sharf et al. 07], which can be built from approximations of the medial axis. Wade [Wade 00] uses the discontinuities in the distance field from the surface to approximate the medial surface and extracts the skeleton from this. Katz and Tal [Katz and Tal 03] propose a shape partitioning algorithm which can be used for skeleton extraction. Other methods for 3D surface segmentation can be similarly adapted for this purpose [Lien et al. 06]. Liu et al. [Liu et al. 03] use a different approximation by repulsive force-fields to estimate the skeleton. Tagliasacchi et al. [Tagliasacchi et al. 09] use cutting planes of the 3D shape to recover the set of interior points, and estimate a curve skeleton for incomplete point clouds. Among the various alternatives for shape approximation, methods based on sphere packing such as the medial axis have remained popular due to their simplicity, and current methods can robustly deal with surface noise.

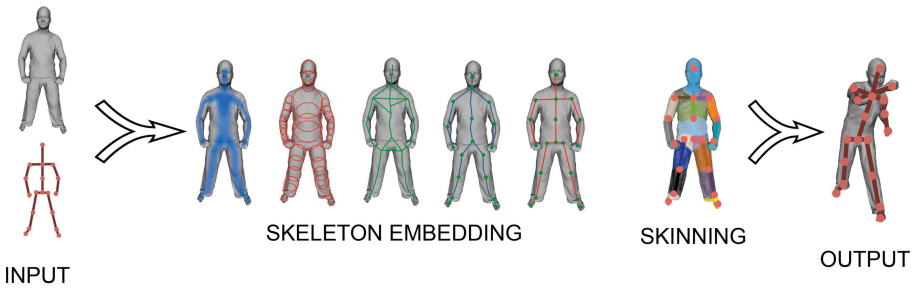


Figure 13.1: The *Pinocchio* system receives as input a character mesh and a skeleton. In the first step, the skeleton is embedded into the character mesh as follows: first, the surface medial axis distance is calculated, packed spheres are found, and the graph is constructed. Thereafter, the optimal embedding of the skeleton is found via discrete optimization and the resulting skeleton is refined. The second step computes skinning weights based on the embedded bones. The result is an automatic rig of the input character.

Instead of relying exclusively on 3D shape geometry, when one has prior information on the type of motion one expects from the input 3D shape, this can be used for the estimation of the skeleton rig. For example, a 3D shape resembling a humanoid can be expected to move in a human-like fashion with similar skeletal dynamics. When a skeleton rig is created with this prior, human motion, such as that captured from an actor's performance, can be easily retargeted to the input shape. Given a static surface mesh, biologically plausible animation skeleton can be extracted using constraints from prior knowledge based on the anatomy of humans or animals [Aujay et al. 07, Schaefer and Yuksel 07]. Baran and Popović [Baran and Popović 07] propose the *Pinocchio* system to embed a human skeletal rig into a static surface mesh of a character using a variety of priors such as the relative lengths of bones, the relative position of legs, etc. Their method also estimates automatically the skinning weights of the vertices from the bones. In comparison with skeleton extraction methods, skeleton embedding is more suitable for automatically animating a character. Extracted skeletons may have different topologies for similar character meshes, which may hinder its use with similar motion data. In addition, embedding a given skeleton provides information about the expected structure of the character, which can be difficult to obtain from just the character geometry.

As shown in Figure 13.1, the *Pinocchio* system [Baran and Popović 07] consists of two main steps: rigging (skeleton embedding) and skinning (skin attachment). The first component, skeleton embedding, computes the joint positions of the input generic skeleton inside the character. Afterwards, the skin attachment is performed by assigning bone weights to the vertices of the character model.

Intuitively, the joint positions need to be computed such that the embedded skeleton fits inside the character correctly and looks like the given skeleton as much as possible. This is achieved by first embedding the skeleton into a discretization of the character's interior and then by refining this embedding using continuous optimization. In summary, the skeleton embedding algorithm works as follows:

- Adaptive distance field is used to compute a sample of points approximately on the medial surface of the input character.
- A graph is constructed where the vertices represent potential joint positions and edges are potential bone segments. The graph is constructed by packing spheres centered on the approximate medial surface into the character and by connecting sphere centers with graph edges.
- The system finds the optimal embedding of the skeleton into this graph with respect to a discrete penalty function considering a variety of penalties like short bones, improper orientation between joints, length differences in bones marked symmetric, bone chains sharing vertices, etc.
- The resulting skeleton found by discrete optimization usually has the general character shape, but typically, it does not fit correctly inside the character. Therefore, embedding refinement is used to correct the embedded skeleton by minimizing a new continuous function that penalizes bones that do not fit inside the surface, bones that are too short, and bones that are oriented differently from the given skeleton.

As a result, the skeleton embedding step resizes and positions the input skeleton to fit inside the character. However, the character and the embedded skeleton are disconnected until the deformations of the skeleton to the character mesh are specified.

Deforming the mesh surface (or skin) by an underlying skeleton structure [Magnenat-Thalmann et al. 88] is the second step of the system: skin attachment or skinning. Briefly, the general standard linear blend skinning (LBS) method [Lewis et al. 00] works as follows: if $\mathbf{v}_i$ is the position of the vertex i, θ_j is the transformation of the j-th bone, and w_{ij} is the weight of the j-th bone for vertex i, the standard linear blend skinning (LBS) approach gives the position of the transformed vertex i as $\mathbf{v}'_i = \sum_j w_{ij}\theta_j(\mathbf{v}_i)$. Therefore, the skin attachment problem is to find bone weights $\mathbf{w}$ for all vertices, indicating how each bone affects each vertex.

Assigning bone weights purely based on proximity to bones will often fail because they ignore the character's geometry. An alternative approach is to use the analogy to heat equilibrium to find the weights. If the character

volume is treated as an insulated heat-conducting body and the temperature of bone j is forced to be 1 while keeping the temperature of all of the other bones at 0, the equilibrium temperature at each vertex on the surface can be considered the weight of bone j at that vertex.

For simplicity, the *Pinocchio* system solves the equilibrium equation over the surface. The equilibrium over the surface for bone j can be written as $-\Delta \mathbf{w}_j + \mathbf{H}\mathbf{w}_j = \mathbf{H}\mathbf{p}_j$, where Δ is the discrete surface Laplacian, calculated with the cotangent formula [Meyer et al. 02], $\mathbf{p}_j$ is a vector with $p_{ji} = 1$ if the nearest bone to vertex i is j and $p_{ji} = 0$ otherwise, and $\mathbf{H}$ is the diagonal matrix with H_{ji} being the heat contribution weight of the nearest bone to vertex i.

The performance of *Pinocchio* has been demonstrated in a variety of papers and applications. The system can be used to automatically rig a character. It allows a user to go from a static mesh to an animated character automatically. As a result, users can animate many different characters using a generic skeleton with quality compared to modern video games.

13.4 Converting a Mesh Animation into a Skeletal Animation

It is popular to represent animations not by means of a classical skeleton-based model, but in the form of deforming mesh sequences, e.g., as a result of performance capture methods. The reason for this is that novel mesh deformation methods as well as surface-based scene capture techniques offer a great level of flexibility during animation creation. Unfortunately, the resulting scene representation is less compact than skeletal ones and there is a limited number of tools available which enables easy post-processing and modification of mesh animations. Several methods have been described to automatically rig a model using an animated mesh sequence [Schaefer and Yuksel 07, de Aguiar et al. 08a, Hasler et al. 10b]. The key idea of these methods is to first perform a motion driven clustering step to extract rigid bone transformations, then estimate the joint locations and bone lengths, and finally optimize the bone transformations and skinning weights.

In this section, a variant of this general algorithm will be described. A method to automatically extract a plausible kinematic skeleton, skeletal motion parameters, as well as surface skinning weights from arbitrary mesh animations will be presented [de Aguiar et al. 08a]. By this means, deforming mesh sequences can be fully automatically transformed into fully rigged virtual subjects.

An overview of the approach is shown in Figure 13.2. The input is an animated mesh sequence comprising N frames. An animated mesh sequence

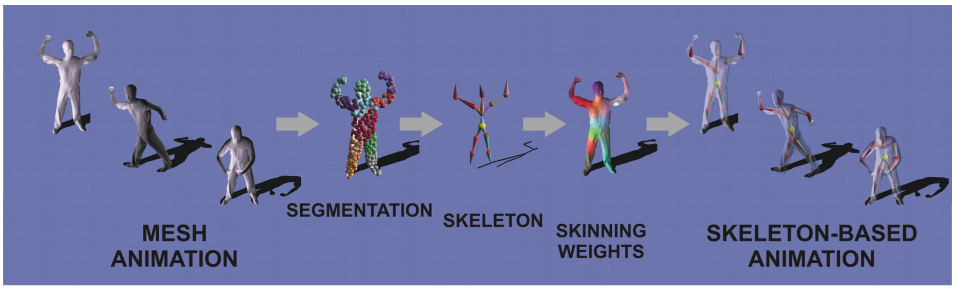


Figure 13.2: Using an animated mesh as input, the approach segments the model into plausible approximately rigid surface patches, estimates the kinematic skeleton and its motion parameters, and calculates the skinning weights connecting the skeleton to the mesh. The output is a skeleton-based version of the input mesh animation.

is represented by a mesh model M consisting of vertices V and triangulation T with positional data $\mathbf{p}_t(\mathbf{v}_i) = (x_i, y_i, z_i)_t$ for each vertex $\mathbf{v}_i \in V$ at all time steps t. In the first step of the algorithm, spectral clustering is used to group seed vertices on the mesh into approximately rigid segments. By using the clustered seed vertices it is possible to segment the moving mesh into kinematically meaningful approximately rigid patches. Thereafter, adjacent body parts are determined and the topology of the kinematic structure of the mesh is found. Using the estimated topology, joint positions between interconnecting segments are calculated over time. In the last step, appropriate skinning weights are calculated to attach the learned skeleton to the surface.

The first step of the algorithm segments the animated input mesh (given by M and $\mathbf{p}_t$) into spatially coherent patches that undergo approximately the same rigid transformations over time. The approach is initialized by selecting a subset of l seed vertices that are distributed evenly over the mesh M using a curvature-based segmentation method [Yamauchi et al. 05]. The motion trajectories of the seed vertices throughout the whole sequence form the input to a spectral clustering approach [Ng et al. 02] which automatically groups the l seeds into k approximately rigidly moving groups. The idea behind this clustering is to capitalize on the invariant that mutual distances between points on the same rigid part should only exhibit a small variance while the mesh is moving. Using the k optimal vertex clusters, the triangle clusters are created. The resulting clusters divide the mesh into k approximately rigid surface patches.

Given the list of body segments, their associated seed vertices and triangle patches, the kinematic skeleton structure is found by first finding its kinematic topology (i.e., find which body parts are adjacent). To deter-

mine which body segments are adjacent, the triangles at the boundaries of the triangle patches are analyzed. Body parts A and B are adjacent if they have mutually adjacent triangles in their respective patch boundaries. Unfortunately, in practice a patch may be adjacent to more than one other patch. Taking a heuristic approach, the method considers only those patches to be adjacent that share the longest common boundary (in terms of the number of adjacent boundary triangles). Note that the system assumes that the body part in the center of gravity of the mesh is the root of the hierarchy. A good estimate for the correct sequence of joint positions is the sequence of locations that minimizes the variance in joint-to-vertex distance for all seed vertices of the adjacent parts at all frames. Therefore, after aligning the segment poses to a reference time step, the algorithm proposed in [Anguelov et al. 04a] is used to calculate all joint positions. Additionally, the bone lengths are enforced to be constant over time.

In order to infer joint motion parameters, i.e., a rotational transformation for all joints at all times, a cyclic-coordinate-descent (CCD)-like algorithm [Luenberger 73, Badler et al. 87] is employed to calculate all motion parameters using Euler angle parameterization. The translation of the root is stored as an additional parameter for each frame. In the last step, the approach described in the previous (Section 13.3) is used to determine the skinning weight distribution for each bone considering the entire mesh sequence and the reconstructed skeleton poses.

The performance of the described system has been validated on a large variety of mesh animations. The fully-automatic approach is able to extract a kinematic skeleton, joint motion parameters, and surface skinning weights from a mesh animation. The original input can then be quickly rendered based on the new compact bone and skin representation, or it can be easily modified using the full repertoire of already existing animation tools.

13.5 Building a Deformable Model

A disadvantage of using a static motion template such as a skeleton or an enclosing cage is that the deformation priors induced by such a template may not correspond to the actual motion characteristics of a real-world object. Simulating the exact physiology and elastic properties of muscles and tendons is possible [Lee et al. 09], but this simulation is costly and time-consuming. Alternatively, real-world datasets of 3D motion can be used to build data-driven models for synthesizing new virtual deformations. These datasets can be acquired either from motion capture systems, or from artist-given example deformations in order to achieve an artistic effect or animation style. Kry et al. [Kry et al. 02] propose the *Eigenskin* system that synthesizes new deformations from examples. Weber et al. [Weber et al. 07]

propose context-aware skeletal shape deformation that uses artist-given examples to enhance the realism of articulated deformations.

Motion capture systems, either based on optical markers or on visual feature tracks over high-resolution multi-view photography, can estimate the general 3D non-rigid motion of an object at a high detail. Such spatio-temporal data can be used to build dynamic 3D deformation models that mimic real-world motion. Park and Hodgins [Park and Hodgins 06] placed a set of 300 optical markers on an actor's body and recorded various muscle deformations. They used this data to build a deformation model for synthesizing physically realistic muscle deformations. They represent the 3D motion of the optical markers on the surface as residual angular transformations from an underlying skeleton. Another related research effort is to parameterize variation of human body proportions from a database of real-world 3D scans of people [Allen et al. 02, Allen et al. 06, Anguelov et al. 05, Hasler et al. 09c] (Chapter 14 gives a more thorough review). As the resolution of these 3D scans increases, surface deformation is parameterized independently from the underlying rigs, such as through deformation gradients [Sumner and Popović 04]. The surface deformation can then be represented in layers: that given by a control rig for editing pose, and that representing residual deformations either due to shape variation across people, or due to pose-specific variation.

The dynamic muscle deformation model of Park and Hodgins [Park and Hodgins 08] can be considered as a demonstrative example. This model is built as an extension of earlier work [Park and Hodgins 06] and able to synthesize dynamic effects such as the jiggling of muscles and body fat. A set of $400 - 450$ reflective markers are placed on an actor and his motion is captured in a studio with 16 near infrared Vicon MX-40 cameras running at 120 frames per second. Thus, an actor-specific database of roughly 10,000 frames of different motions are captured: flexing, twisting, running, jumping, punching, etc. Each motion is captured at different speeds to study the effect of dynamics. The motion capture data is then cleaned and smoothed to remove noise artifacts and to fill holes. The actor's body is segmented into a set of 17 near-rigid parts and each marker is placed in a local coordinate frame. The rigid motion of the body part is then removed and the residual displacement of the marker in the local co-ordinate frame is analyzed by decomposing it into static and dynamic components.

$$\mathbf{d} = \mathbf{d_s} + \mathbf{d_d} \tag{13.1}$$

where $\mathbf{d}, \mathbf{d_s}, \mathbf{d_d} \in \mathbb{R}^{3N}$ with N being the number of markers.

The static component is due to pose-specific muscle bulging, and is estimated using a locally weighted linear regression model.

$$\mathbf{d_s} = A\Theta \tag{13.2}$$

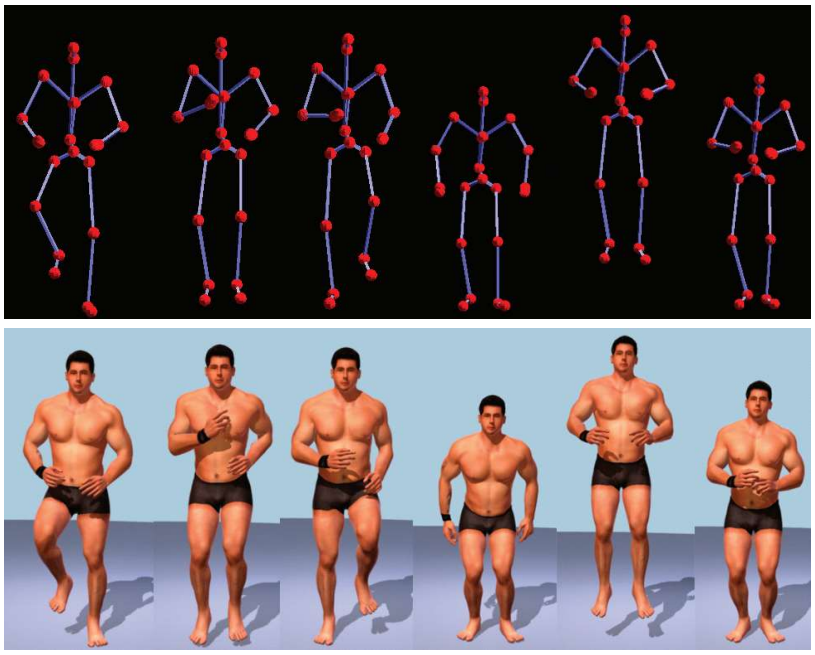


Figure 13.3: Surface deformation results from the data-driven deformation model of [Park and Hodgins 08]. The top row shows skeletal pose input; the bottom row shows detailed surface deformation. The data-driven model is built from dynamic motion capture data of $400 - 450$ reflective markers placed on the actor's body.

where $\Theta = [\theta_1^\top \theta_2^\top 1] \in \mathbb{R}^7$ is a vector denoting the pose of the part (θ_1, θ_2 denote the angular displacement vectors between the body part and its inboard and outboard neighbors from the skeletal data). The regression matrix A is estimated by weighted linear least squares from ground-truth residual displacements. This regression model is not yet able to capture dynamic effects such as jiggling of the muscle and fat. These dynamics are captured by $\mathbf{d_d}$ using a second-order differential model.

$$m_k \ddot{d}_k + C_k \dot{d}_k + K_k d_k = f_k \tag{13.3}$$

where d_k is the k-th component of $\mathbf{d_d} \in \mathbb{R}^{3N}$. C_k and K_k are damping and stiffness coefficients, f_k is the component of the net external force $\mathbf{f}$ contributing to the dynamic local deformation component d_k, and m_k is an imaginary mass to provide inertia effects. By approximating $\ddot{d}_k, \dot{d}_k$ with numerical derivatives, the above equation can be linearized and solved using least-squares (the regression model can be made smoother and more robust by dimensionality reduction using principal component analysis (PCA) on $\mathbf{d_d}$ and solving for the projected components in a lower dimensional space).

Given a new body pose, the final skin deformation is synthesized by estimating the static and dynamic residual deformations for each vertex, and adding them to the rigid motion of the body part. High-quality animations of muscle and skin deformations are thus generated by exploiting the motion data (Figure 13.3).

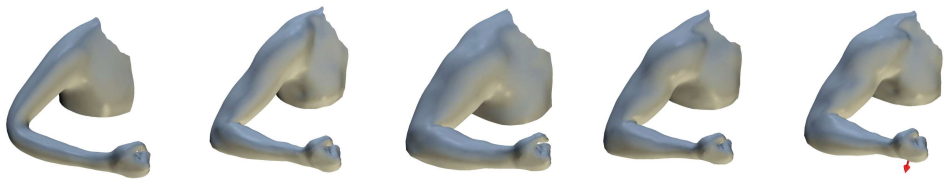
One limitation of this work is the crude segmentation into rigid body parts, which was rectified in a later work by Hong et al. [Hong et al. 10], who proposed a method for automatically estimating the underlying skeleton rig for the shoulder-arm complex through a data-driven segmentation of the marker trajectories.

De Aguiar et al. [de Aguiar et al. 10] propose a similar dynamic model for learning stable spaces for cloth deformation (Chapter 15). De Aguiar and Ukita [de Aguiar and Ukita 14] proposed an approach to represent and manipulate a mesh-based character animation preserving its time-varying details by decomposing the input mesh animation into coarse and fine deformation components. A model for the coarse deformations is constructed by an underlying kinematic skeleton structure and blending skinning weights and a non-linear probabilistic model based on Gaussian processes is used to encode the fine time-varying details of the input animation.

Using multi-view camera recordings of real-world muscle exercises, Neumann et al. [Neumann et al. 13a] built a data-driven model for the shoulder-arm complex that synthesizes realistic deformation not only according to pose (Θ), but also with respect to the body shape parameters of the actor (β) as well as external forces acting on the arm (γ).

$$\mathbf{d}_s = \Psi(\Theta, \beta, \gamma) \tag{13.4}$$

They model only the static residual deformations $\mathbf{d_s}$ while the actors perform slow and natural movements, where dynamic effects can be neglected within the so-called *kinesostatic* assumption. They capture the surface deformations of the arm by applying a makeup of random dot patterns on human actors and imaging real-world muscle exercises at a high resolution through a multi-camera system. These muscle exercises are repeated with the subjects holding different weights in their arms. In this way, reactionary forces in the muscles and the resultant 3D surface bulging is captured from a passive multi-camera setup. Different arm muscles, such as biceps and triceps, get activated as a result of changes in the direction and magnitude of the external force on the hand. They propose a hybrid model, by fitting a linear regression model for variations due to body shape and external forces, and a non-linear regression model for more complex variations due to body pose. Figure 13.4 shows the variation of surface deformations produced by the model by varying the body shape parameters β and the addition of external force vector γ.



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

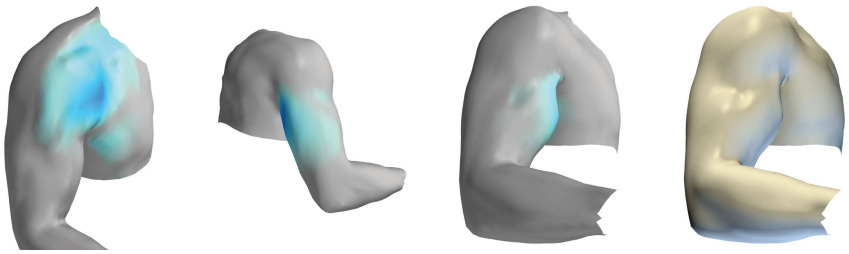
Figure 13.4: Statistical deformation model of the human arm. A simple pose-space deformation produces skinning artifacts (a) which are corrected by residual transformations learned by the model. The body shape parameters are varied between normal (b), high BMI (c), and high muscularity ratio (d). In (e), an external force vector is added to the muscular arm in (c) that sharply shows muscle-bulges. Figures adapted from [Neumann et al. 13a].

It is possible to extend such deformation models to account for specific muscle groups. In a later work [Neumann et al. 13b], Neumann et al. perform a sparse-matrix factorization on the meshes showing the second layer of residual transformations, after pose-specific deformation is subtracted. By concatenating these residual displacement vectors into an animation matrix $\mathbf{M}$, a basis of deformation components $\mathbf{C}$ is discovered by factoring the matrix $\mathbf{M}$.

$$\mathbf{M} = \mathbf{WC}. \tag{13.5}$$

It is desirable to obtain the deformation components $\mathbf{C}$ as corresponding to the actions of specific muscle groups that act in a localized manner on the surface. One option to achieve such an effect is to take the appropriate basis $\mathbf{C}$ from user-input. However, it is possible to achieve an approximate effect by giving appropriate priors on $\mathbf{C}$ and $\mathbf{W}$ for the decomposition. Specifically, the components $\mathbf{C}$ can be expected to be sparse and localized on specific surface regions. Neumann et al. [Neumann et al. 13b] use the sparsity inducing $L1$-norm as a regularizer on $\mathbf{C}$ in Eq.(13.5). Their paper discusses using other priors for locality and well-behaving weights. When run on a subset of the captured arm-muscle dataset of [Neumann et al. 13a] (bicep-curl movements of one subject), the decomposition components of their method loosely correspond to physical muscle-groups such as biceps, triceps, etc. This model can be used to enhance specific muscle effects (Figure 13.5).

In conclusion, data-driven deformation models learned from mesh sequences captured from the real world are a powerful artistic tool. They not only reduce the modeling time of the artists but also achieve more physically realistic rigs that can be manipulated easily.



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

Figure 13.5: Sparse localized deformation components found on a subset of the muscle exercise dataset. (a,b,c) show localized effects on the deltoid, triceps, and biceps regions, respectively. In (d), the biceps bulge is exaggerated by amplifying the component in (c).

13.6 Summary

In this chapter, various methods for building virtual 3D characters from performance-captured mesh sequences were discussed, such that they can be controlled and edited by artists. With rising trends in modern 3D sensing technology and large-scale machine learning from big data, the deformation quality in the virtual characters can be expected to improve greatly. Historically, animation editing methods based on geometric templates and on statistical deformation models have evolved along different paths, taking inspiration from the mathematical disciplines of differential geometry and probabilistic modeling, respectively. In the future, these methods are likely to converge into a single unified discipline. Statistical deformation models built from real 3D motion datasets will also be used increasingly by professionals for real-world visual effects and computer games. These models will be built not only on traditional artistic toolkits such as bone skeletons, but also on more anatomically accurate skeletons and muscle models. In conclusion, capturing real-world 3D motion and building data-driven deformation models will help a professional 3D artist or computer programmer to create more realistic virtual characters.

14
Statistical Human Body Modeling

Stefanie Wuhrer, Leonid German, and Bodo Rosenhahn

14.1 Introduction

Human body models are required in many applications, such as creating realistic animations for gaming and movie productions, or creating accurate simulations for quality control in ergonomic design. In these application scenarios, the user wants to control the body shape and posture using a small number of intuitive parameters. To achieve this goal, application-dependent, manually generated control parameters can be designed. This is a common technique when rigging a character for a movie production (Chapter 13). However, generating these parameters manually is time-consuming and expensive.

A fully automatic alternative to this manual approach is to learn a small set of control parameters from a large database of human body scans using machine learning techniques. This method offers the advantage of learning application-dependent shape and posture variations without requiring extensive manual input.

However, performing statistics on a set of 3D scans is a challenging problem. To statistically analyze the shapes, a distance measure between pairs of shapes needs to be defined. This is challenging even in the case where the same subject is scanned multiple times in the same posture, because the scans are corrupted by noise and missing data, and because different scans contain different numbers of points. This problem is further complicated, because scans of different subjects scanned in different postures exhibit a large variation in body shape. Figure 14.1 depicts some noisy body scans that have very different shapes. This chapter outlines some methods to process these scans and to learn parametric models of body shape and posture.

To analyze 3D models of human bodies used in product design applications and entertainment, databases of 3D scans of different subjects are

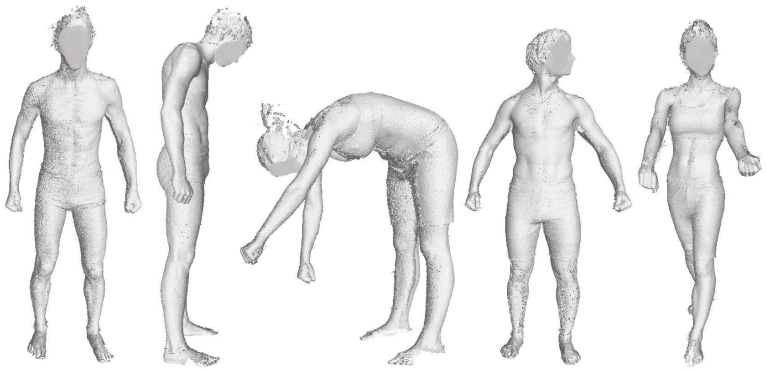


Figure 14.1: Human body scans of different subjects in different postures may contain large shape variation and be corrupted by noise and holes. Models from the MPI database [Hasler et al. 09c].

required. Therefore, several large databases have been collected. The first large 3D database of human models was the Civilian American and European Surface Anthropometry Resource (CAESAR) [Robinette et al. 99], whose goal was to provide accurate anthropometric data for various design applications. This commercially available database contains over 4500 subjects in three postures each and was collected at multiple locations across North America and Europe. Similar datasets have been subsequently compiled all over the world in order to find typical, representative shapes of a population to improve the sizing systems used for product design (e.g., the SizeGERMANY survey [SizeGermany 07]). All of these surveys have the goal to analyze variation in body shape only and, therefore, collect scans in very few standard postures. To use statistical models for animation, subjects need to be scanned in many different postures in order to allow for the automatic extraction of posture variations from the database. For these applications, Hasler et al. [Hasler et al. 09c] compiled a database of over 100 subjects in up to 35 postures each. Unlike the previously discussed databases, it is freely available for research purposes. Figure 14.1 shows some scans from this database.

14.2 Overview

This section gives a brief overview of the challenges related to human body modeling. The goal is to learn a small number of parameters to control the body shape and posture of a human body model from a large database of human body scans acquired in different postures.

Once a 3D model database of human body scans has been acquired, each model is represented by a set of points that are possibly connected by

triangles. At this stage, the different body scans cannot be directly compared, because different models can have different numbers of vertices, and they are corrupted by noise and missing data. Furthermore, no information is available about which points or triangles correspond to which body part. To compare the models, intrinsic correspondence information needs to be established. Computing dense correspondences between 3D models is a challenging problem that continues to receive considerable attention in the computer vision and computer graphics research communities [van Kaick et al. 11, Tam et al. 13].

To compute correspondences between human models, *templates* are commonly used. Templates describe the rough shape and deformation behavior of a human model and are usually represented by a rigged triangle mesh (Chapter 13). This technique was first proposed by Allen et al. [Allen et al. 03] to compute correspondences between human models in a standard posture and subsequently extended in numerous works to allow for posture variation [Anguelov et al. 05, Hasler et al. 09c]. To initialize the template fitting, anatomical markers corresponding to joint positions are commonly used. Acquiring these positions is time-consuming and expensive. For instance, acquiring the CAESAR database which includes 73 anatomical marker positions per scan took four years and cost $6 million.[1] To avoid this cost, data-driven methods learn localized information about the geometry of the marker positions and use this learned information to predict marker positions on a newly available scan automatically [Azouz et al. 06, Wuhrer et al. 10]. These automatically predicted landmarks have been shown to be sufficiently accurate to allow for the computation of reliable correspondences for most scans [Wuhrer et al. 11]. Section 14.3 discusses methods that predict marker positions and methods that fit a template to a scan in more detail.

Once a database of human body models is in correspondence, shape variations across different subjects and postures can be analyzed using machine learning techniques. To this end, each shape is represented as a vector consisting of the x-, y-, and z-coordinates of its n vertices in an arbitrary but fixed order. This representation of each model in $\mathbb{R}^{3n}$ allows to compute distances between shapes as vector distances. Machine learning techniques aim to represent each human model in a low-dimensional *shape space*. Allen et al. [Allen et al. 03] analyze a database of processed human scans captured in a standard posture using principal component analysis (PCA), which finds a shape space as the orthogonal linear sub-space of $\mathbb{R}^{3n}$ that captures the largest proportion of the shape variability present in the database.

[1] http://www.sae.org/standardsdev/tsb/cooperative/caesumm.htm

To facilitate the simultaneous analysis of shape and posture variations, Anguelov et al. [Anguelov et al. 05] couple this shape space with an additional linear shape space that models the deformation of a single subject captured in multiple postures to allow for posture variation. The advantage of this method, which is called SCAPE, is that the shape and posture spaces can be modeled using linear mappings, and shape and posture can be analyzed independently. Section 14.4 discusses the commonly used SCAPE model that represents human shape and posture changes in more detail. The main disadvantage of this model is that the posture variation is learned from a single subject.

In order to extend the analysis of posture variations to a population of subjects, Allen et al. [Allen et al. 06] propose the use of multiple subjects captured in multiple postures to learn a correlated shape space capturing body shape and posture variations. A substantial drawback is that transforming a model from $\mathbb{R}^{3n}$ to this shape space requires a non-linear transform. This was addressed by Hasler et al. [Hasler et al. 09c] with the use of a rotation-invariant encoding of the models to find a linear shape space that encodes human body shape and posture in a correlated way. This method allows to model muscle deformations accurately using a linear shape space, which has applications in fitting human models to scans acquired with clothing [Hasler et al. 09b]. However, this model does not allow to control human shape and posture variations independently. To obtain more control, Hasler et al. [Hasler et al. 10a] propose the use of a bilinear shape space to estimate the 3D shape of a human model from a single image. Recently, Chen et al. [Chen et al. 13b] combined such a bilinear model with the SCAPE model to allow for intuitive parameters to control posture variation while achieving realistic shape deformations near joints.

In some applications, such as in ergonomic design, it is desirable to analyze the variations due to body shape differences independently of posture changes of the models. To achieve this goal, Wuhrer et al. [Wuhrer et al. 12] propose to perform PCA on a shape representation based on localized Laplace coordinates of the mesh.

Statistical models offer the advantage of allowing the control of realistic human models through few parameters. Applications include reconstructing 3D human models from possibly dynamic RGB or depth image data [Balan et al. 07, Balan and Black 08, Guan et al. 09, Hasler et al. 10a, Weiss et al. 11, Helten et al. 13], controlling human models using a small set of semantic parameters, such as one-dimensional measurements or body weight [Allen et al. 03, Wuhrer and Shu 13, Rupprecht et al. 13], and editing image and video data [Zhou et al. 10, Jain et al. 10].

14.3 Parameterization of Human Body Models

This section discusses how to process raw human body scans, such that they can subsequently be used for statistical analysis. This is a difficult problem, where the goal is to correspond two surfaces T and S containing $n^{(T)}$ and $n^{(S)}$ vertices, respectively. When trying all possible combinations, the search space for solving the correspondence problem has size $O\left(n^{(T)}n^{(S)}\right)$. In practice, this problem is further complicated by acquisition artifacts, such as noisy and incomplete data. Usually it is difficult to filter the noise and fill the holes in the acquired data, because the geometry of the data can be complex.

To remedy these problems, a template T consisting of a rigged triangle mesh that represents the shape and deformation behavior of a typical human body is often used to reduce the search space. The task of computing a correspondence between a population of human body shapes $S^{(1)}, \ldots, S^{(m)}$ represented by possibly noisy and incomplete point clouds can now be solved by deforming T to each shape $S^{(i)}$. After this step, there are m deformed versions of T (and hence, correspondences are known), and the i-th deformed version of T is close to $S^{(i)}$. This approach has the advantage of leading to m complete surfaces in correspondence even when the original acquisitions are incomplete, without solving the challenging task of filling holes in scan data explicitly.

This template fitting approach uses a non-rigid iterative closest point (ICP) [Besl and McKay 92] algorithm and was first proposed by Allen et al. [Allen et al. 03]. Non-rigid ICP approaches proceed by repeatedly computing the current correspondences between T and a scan S using nearest neighbors in $\mathbb{R}^3$ and use these correspondences to smoothly deform T to S. The methods stop when the alignment no longer changes significantly. Hence, these approaches need a good initialization to avoid getting trapped in local minima.

To find a good initial shape alignment, anthropometric landmarks are commonly used. In a first step, the landmarks are used to roughly align the initial shape with the data by using a skeleton model. Afterwards, the landmarks give a small set of corresponding points on T and on S which can be used to guide the deformation from T to S. To fit T to S, two fitting steps are commonly used. First, the landmarks are used to refine the posture of the skeleton of T, such that the deformed version of T is close to S. Second, the vertices of T are deformed to be close to S using non-rigid ICP.

Regarding the first step, to fit the skeleton of T such that the posture of T is close to the posture of S, the method takes advantage of the rigging information of T. That is, a skeleton model of T consisting of b bones along with a set of rigging weights describe how the vertices of T move with

respect to the skeleton. Let $\mathbf{A}_k$ denote the 3×4 transformation matrix that encodes the rigid transformation of the k-th bone. Note that $\mathbf{A}_k$ depends on few parameters since most joints only allow a rotation. Let $\mathbf{l}_j^{(T)}$ and $\mathbf{l}_j^{(S)}$ denote the corresponding landmarks of T and S, respectively. Furthermore, in the following, let $\tilde{\mathbf{p}}$ denote the homogeneous coordinates of the 3D point $\mathbf{p}$. Landmark $\mathbf{l}_j^{(T)}$ can be deformed to $\sum_{k=0}^{b-1} w_{jk}\mathbf{A}_k \left(\tilde{\mathbf{l}}_j^{(T)}\right)$, where w_{jk} is the rigging weight for the k-th bone and the j-th landmark of T. A commonly used approach to fit the posture of T to the posture of S is to use a variational approach to solve for the bone transformations $\mathbf{A}_k$, such that

$$E_{\text{lnd}}(\mathbf{A}_k) = \sum_j \left(\sum_{k=0}^{b-1} w_{jk}\mathbf{A}_k \tilde{\mathbf{l}}_j^{(T)} - \tilde{\mathbf{l}}_j^{(S)} \right)^2 \tag{14.1}$$

is minimized. This is also referred to as *skeleton transfer*.

After the posture of T has been initialized to be close to the posture of S, in the second fitting step, the vertices of T are deformed to be close to the surface of S using non-rigid ICP. This step is also called *shape alignment*, and it deforms each vertex $\mathbf{p}_j$ of T using a 3×4 transformation matrix $\mathbf{A}_j$.

Different methods allow for different classes of transformation matrices $\mathbf{A}_j$. To allow for elastic deformations, affine transformation matrices are often allowed. Alternatively, to restrict the deformations to be piecewise rigid, rigid transformations are allowed per vertex. The goal is to fit T to S while preserving the overall shape of the template surface. This is achieved by minimizing the energy

$$E_{\text{fit}} = \omega_{\text{data}} E_{\text{data}} + \omega_{\text{smooth}} E_{\text{smooth}}, \tag{14.2}$$

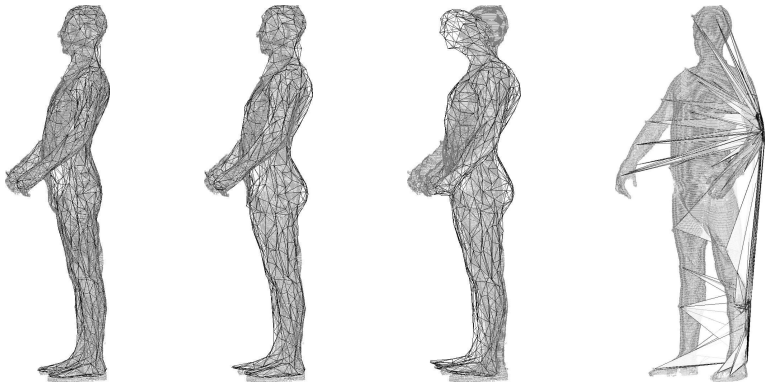
with respect to the deformations $\mathbf{A}_j$. The term

$$E_{\text{data}} = \sum_j \left(NN^{(S)}\left(\mathbf{A}_j \tilde{\mathbf{p}}_j\right) - \mathbf{A}_j \tilde{\mathbf{p}}_j \right)^2 \tag{14.3}$$

is an energy that pulls the vertices of T toward the sensor data S, where $NN^{(S)}\left(\mathbf{A}_j \tilde{\mathbf{p}}_j\right)$ is the nearest neighbor of the transformed vertex $\mathbf{A}_j \tilde{\mathbf{p}}_j$ on S. The term E_{smooth} encourages close-by vertices to have similar deformations, e.g., rotation around a common axis as opposed to translation by different distances. Thereby transformations which incur low E_{smooth} penalties are more likely to preserve the overall shape of the template. $E_{\text{smooth}} = \sum_j \sum_{(j,k)\in E} |\mathbf{A}_j - \mathbf{A}_k|_F^2$ is commonly used, where E is the set of edges of T, and where $|\cdot|_F^2$ denotes the squared Frobenius norm.

It is noteworthy that the location of the minimum is influenced only by the ratio of the weighting factors, e.g.,

$$E_{\text{fit}} = \omega_{\text{data}} E_{\text{data}} + \omega_{\text{smooth}} E_{\text{smooth}} \propto E_{\text{data}} + \frac{\omega_{\text{smooth}}}{\omega_{\text{data}}} E_{\text{smooth}}. \tag{14.4}$$



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

Figure 14.2: The scan is shown in gray and the fitting result as wireframe. (a) Successful fit with normal values for ω_{data} and ω_{smooth}. (b) A fit with normal smoothness, data not considered. (c) A fit with very high ω_{smooth}. (d) A fit with very high ω_{data}.

The influence of the two fitting terms E_{data} and E_{smooth} is demonstrated in Figure 14.2. Figure 14.2(a) shows a successful template fitting result; ω_{data} and ω_{smooth} were well-balanced in this case. Figure 14.2(b) shows the result if E_{data} is not considered during the fitting. The mesh as a whole was affinely transformed using landmarks, and the pose adjusted to the scan, but the mesh was not fitted to the scan surface. Figure 14.2(c) shows a fitting with an exaggerated value for ω_{smooth}. A translation and centering occurred, but the pose of the mesh is unchanged, as that would increase the smoothness term which is penalized strongly in this case. Minimizing the distance between landmarks on the scan and the template, the mesh was shrunk slightly and transformed in other ways that do not increase E_{smooth}, leaving it a little warped. Figure 14.2(d) shows the result if a disproportionately large value is chosen for ω_{data}. The mesh has completely lost its form, as most vertices are simply attracted to the closest points on the scan. Certain vertices, however, are still attracted to landmarks; their adjacent faces are seen overlapping the scan surface.

One remaining question is how to find the anthropometric landmarks $\mathbf{l}_j^{(S)}$ on a scan S automatically. This can be achieved by taking advantage of the observation that most anthropometric landmarks represent skeletal features and correlate with geometric surface features. This allows the use of machine learning techniques to learn statistical properties about the geometry of these landmark locations from a training database of human models with annotated landmark positions.

To this end, the landmarks $\mathbf{l}_j$ are considered as nodes V of a graph and are linked by edges E in a skeleton-like structure. For each landmark, a node descriptor is computed, and a probability distribution is then fitted to the descriptor values of the landmark location over all shapes of a training set. Commonly used node descriptors include surface curvatures, spin images [Johnson and Hebert 97], and measures related to the lengths of or the areas enclosed by geodesic isolines at fixed distances from the landmark [Sun and Abidi 01]. Similarly, for each edge, an edge descriptor is computed, and a probability distribution is then fitted to the descriptor values of the edges over all shapes of a training set. Commonly used edge descriptors include the length or orientation of the edge measured in a pre-aligned pose of the shape.

This learned information is then used to model a Markov random field (MRF) [Duda et al. 01], where a random variable $\mathbf{x}$ associated with node $\mathbf{l}_j$ is considered conditionally independent of all other variables given the variables associated with the neighbors of $\mathbf{l}_j$. A MRF models the joint probability of a set of random variables $\mathbf{x}_j$ corresponding to potential locations of $\mathbf{l}_j$ as

$$p(X) = \frac{1}{Z} \prod_{\mathbf{l}_j \in V} \phi_j(\mathbf{x}_j) \prod_{(\mathbf{l}_j, \mathbf{l}_k) \in E} \psi_{j,k}(\mathbf{x}_j, \mathbf{x}_k), \tag{14.5}$$

where $\phi_j(\mathbf{x}_j)$ is the likelihood that vertex $\mathbf{x}_j$ corresponds to landmark $\mathbf{l}_j$ (computed using the previously learned probability distribution over node descriptors), $\psi_{j,k}(\mathbf{x}_j, \mathbf{x}_k)$ is the joint likelihood that vertices $\mathbf{x}_j$ and $\mathbf{x}_k$ correspond to landmarks $\mathbf{l}_j$ and $\mathbf{l}_k$ (computed using the previously learned probability distribution over edge descriptors), and Z is a normalizing factor. This model allows to find the vertices $\mathbf{x}_j$ on a new scan that maximize the joint probability $p(X)$. These are the most likely positions of the landmarks $\mathbf{l}_j$ on this scan according to the geometric descriptors used to model the nodes and edges.

Figure 14.3 shows some manually picked landmarks in red and automatically predicted landmarks in blue. The algorithm used to compute these landmarks used 200 models of the MPI database in 35 different postures for training and is described in more detail in Wuhrer et al. [Wuhrer et al. 10].

14.4 Statistical Analysis of Human Body Models

This section discusses methods to analyze shape variations of a database of human scans that are in correspondence. Analyzing shape variations allows to find a small number of parameters that can be used to control human body shape and posture changes.

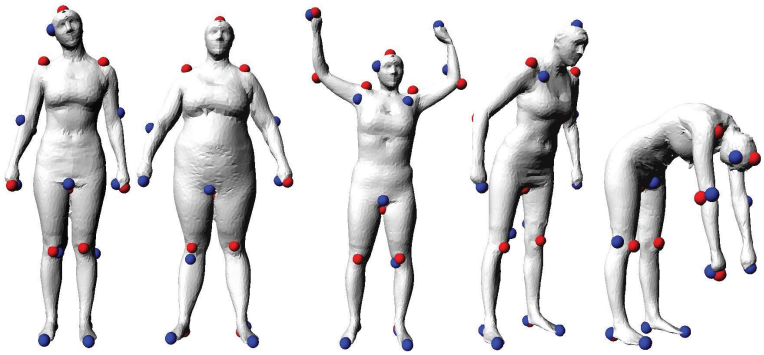


Figure 14.3: Manually picked (red) and automatically predicted (blue) landmarks on models from the MPI database. For correctly predicted vertices, only the blue landmark is visible.

Let $S^{(1)}, \ldots, S^{(m)}$ denote a database of human body shapes consisting of n vertices that are each in correspondence and that have been aligned rigidly to be located in the same coordinate system. To align a population of shapes rigidly, generalized Procrustes analysis (described in more detail in Dryden and Mardia [Dryden and Mardia 02]) can be used. The shapes $S^{(i)}$ can be represented as vectors $\mathbf{s}^{(i)} = \left[x_1^{(i)}, y_1^{(i)}, z_1^{(i)}, \ldots, x_n^{(i)}, y_n^{(i)}, z_n^{(i)}\right]^T$ in $\mathbb{R}^{3n}$, where $(x_j^{(i)}, y_j^{(i)}, z_j^{(i)})$ are the coordinates of the j-th vertex of $S^{(i)}$. It is straightforward to convert between the representations $S^{(i)}$ and $\mathbf{s}^{(i)}$, and the new representation $\mathbf{s}^{(i)}$ allows to compute distances between pairs of shapes as Euclidean distances in $\mathbb{R}^{3n}$.

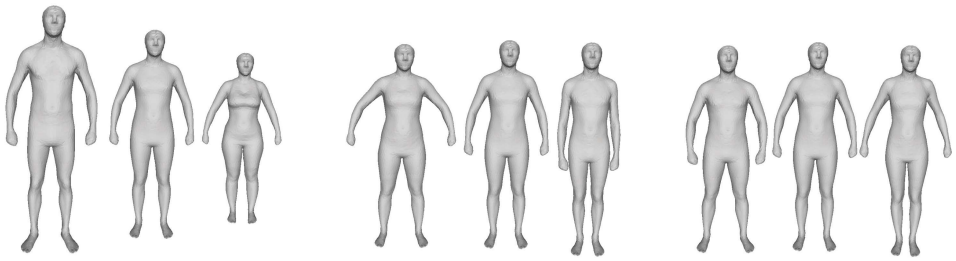
To analyze variations in body shape only, PCA can be used. PCA is an approach that aims to reduce the dimensionality of a dataset given as vectors in $\mathbb{R}^{3n}$ to $\mathbb{R}^d$ with $d < \min(m, 3n)$ using a linear transformation, while aligning the reduced coordinate system along the main modes of variation of the data. More specifically, the aim is to express the data vectors $\mathbf{s}^{(i)}$ as

$$\mathbf{s}^{(i)} = \left(\sum_{j=1}^{d} a_{ij} \mathbf{e}_j \right) + \bar{\mathbf{s}}, \tag{14.6}$$

where $\mathbf{e}_j$ are a set of basis vectors, a_{ij} are scalar weights, and $\bar{\mathbf{s}} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{s}^{(i)}$ is the mean body shape. This way of reducing the dimensionality can be expressed in matrix form as

$$\mathbf{S}^{(\text{cent})} = \mathbf{A}\mathbf{E}, \tag{14.7}$$

where $\mathbf{S}^{(\text{cent})}$ is a $(m \times 3n)$ matrix that contains the centered data $\mathbf{s}^{(i)} - \bar{\mathbf{s}}$ as its rows, $\mathbf{A}$ is a $(m \times d)$ matrix with entries a_{ij}, and $\mathbf{E}$ is a $(d \times 3n)$ matrix that contains the basis vectors $\mathbf{e}_j$ as its rows.



(a) First PC (b) Second PC (c) Third PC

Figure 14.4: Shape variations along the first three principal components for models of the MPI database.

It remains to compute the basis vectors $\mathbf{e}_j$ and the weight matrix $\mathbf{A}$. PCA computes the basis vectors with the help of an eigendecomposition of the $3n \times 3n$ sample covariance matrix

$$\mathbf{\Sigma} = \frac{1}{m-1} \sum_{i=1}^{m} (\mathbf{s}^{(i)} - \overline{\mathbf{s}})(\mathbf{s}^{(i)} - \overline{\mathbf{s}})^T. \tag{14.8}$$

Let $\lambda_1, \ldots, \lambda_d$ denote the largest eigenvalues of Σ in non-increasing order. The eigenvectors corresponding to λ_j are used as new basis vectors $\mathbf{e}_j$. These basis vectors are orthogonal and have the property that $\mathbf{e}_j$ corresponds to the j-th largest direction of data variability. The coordinate axes $\mathbf{e}_j$ are called the principal components (PC) of the data. Given $\mathbf{S}^{(\text{cent})}$ and $\mathbf{E}$, $\mathbf{A}$ can be computed. Dryden and Mardia [Dryden and Mardia 02] and Duda et al. [Duda et al. 01] give more extensive discussions of PCA and its properties.

PCA can be used to visualize the main variations of a population of shapes by sampling points, denoted by $\mathbf{a}$, along the coordinate axes in the reduced space $\mathbb{R}^d$ and by computing and visualizing the shape vectors $\mathbf{s}$ as $\mathbf{s} = \mathbf{aE} + \overline{\mathbf{s}}$. Figure 14.4 shows the shape variations computed along the first three principal components for all models of the MPI database captured in standard standing posture. This approach was first applied to analyze the shape variations of 3D human body shapes by Allen et al. [Allen et al. 03].

PCA is a global method, that is, each principal component influences each vertex coordinate. This is not always desirable. For instance, to analyze both body shape and posture variations, it is better to segment the body into several parts and to perform shape analysis separately on different parts.

Anguelov et al. [Anguelov et al. 05] proposed SCAPE, which is an extension of PCA that allows to analyze both shape and posture variations independently. Shape variations are analyzed by performing PCA over a

dataset of different human subjects captured in a standard posture. Posture variations are analyzed using a dataset $P^{(1)}, \ldots, P^{(k)}$ of a single human subject captured in k different postures. The two types of variations are then linearly combined to allow for both body shape and posture changes.

To analyze posture variations, SCAPE decomposes the human body into different body parts. Let $r_j^{(i)}$ denote the j-th body part of $P^{(i)}$, and let $P^{(1)}$ denote the model acquired in the same posture that was used to analyze body shape variations. SCAPE computes for each training posture and each body part a rigid transformation that aligns $r_j^{(1)}$ to $r_j^{(i)}$. This rigid alignment is not sufficient to deform $r_j^{(1)}$ to $r_j^{(i)}$, as posture changes cause non-rigid deformations, such as muscle bulging. To model these effects, an affine transformation is computed for each triangle in $r_j^{(1)}$ that deforms this triangle in $r_j^{(1)}$ to its corresponding triangle in $r_j^{(i)}$. It remains to link the body shape differences that occur due to posture changes to a small set of skeleton parameters. This is achieved by computing a linear regression between a small set of skeleton parameters and the computed rigid and affine transformations.

PCA does not necessarily give semantically meaningful variations. The variations are merely the most significant ones in terms of data variability. To remedy this, a regression between the PCA space and semantically meaningful parameters, such as body measurements, is commonly learned [Allen et al. 03]. This regression can then be used to generate synthetic human body models with desired measurements provided as input.

In the following, some typical results are shown that can be obtained by applying such a regression method within the learned PCA-space. Here, the intuitive input parameters are body height and weight. All results are obtained using an iterative regression with the mean shape as starting position.

Figure 14.5 shows the regression results of synthesizing a typical male body shape with input parameters height 180 cm and weight 80 kg, and a typical female body shape with input parameters height 170 cm and weight 50 kg. Note that realistic body shapes are generated.

Figure 14.6 shows the influence of changing the height and weight parameters separately. The left side shows the influence of changing the height for male body shapes. Results are synthesized for heights of 160 cm and 180 cm when keeping the weight fixed at 70 kg. The right side shows the influence of changing the weight parameter for female body shapes. Results are synthesized for weights of 40 kg and 80 kg when keeping the height fixed at 170 cm. Note how these input parameters lead to realistic changes in body proportions for all synthesized body shapes.

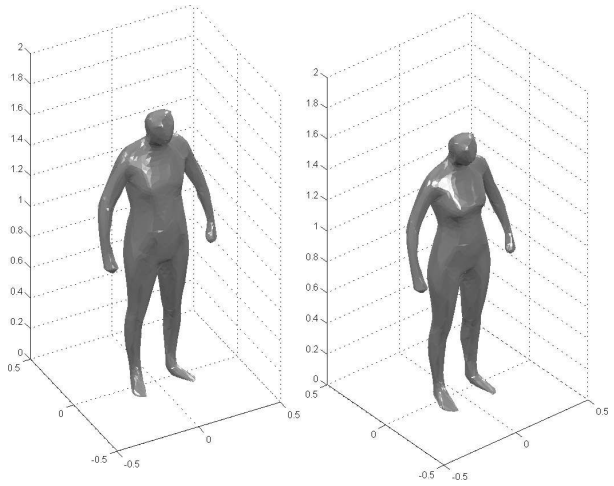


Figure 14.5: Results of regression based on body height and weight. From left to right: Male with height/weight of (180 cm, 80 kg) and female with height/weight of (170, 50 kg).

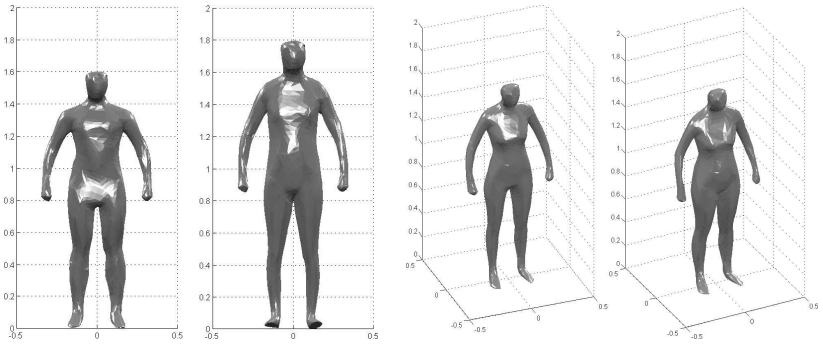


Figure 14.6: Influence of the height and weight parameters on regression results. Left: Regression of a male body shape with input parameters height 160 cm and 180 cm, respectively, and weight 70 kg. Right: Regression of a female body shape with input parameters height 170 cm and weight 40 kg and 80 kg, respectively.

14.5 Summary

This chapter presented commonly used methods to parameterize and statistically analyze human body models. To parameterize the body models, a template model can be deformed to the data using a variational technique that jointly optimizes a data fitting energy and a deformation smoothness energy. To use a template fitting method, the template needs to be aligned to the data. This can be achieved with the help of a sparse set of anthropometric landmarks, which can be predicted automatically with the help of machine learning techniques.

PCA, which is a commonly used statistical method to analyze a population of human body models, was discussed. It finds the main modes of variation of the population. The result of this analysis can be used in various applications, such as the presented application of predicting synthetic body models from a sparse set of intuitive measurements.

An interesting direction for further research is to statistically analyze human body shape and posture changes as the subjects perform different types of motion, such as walking or running.

15
Cloth Modeling

Anna Hilsmann, Michael Stengel, and
Lorenz Rogge

15.1 Introduction

Realistic looking garments and clothes are an important component when it comes to modeling reality and producing photorealistic images including people. In modern movies, many scenes contain completely virtual actors driven by motion capture data, or real actors are visually augmented with virtual clothes in a post-process. Thus, realistic virtual clothing and proper models for fabric behavior and appearance are very important. Cloth modeling and simulation is a challenging task, because cloth deformation and drapery exhibit many degrees of freedom, and wrinkles produce complex deformations and shading. Furthermore, due to the complex structure of fibers, modeling realistic reflection properties of cloth is very difficult. Yet, these complex details are essential for realistic appearance of the virtual clothes. Different approaches exist for modeling and rendering realistic looking pieces of clothing with correct or plausible behavior and appearance, ranging from sophisticated and very complex methods, accurately simulating all fine details with physics-based methods, to real-time methods that focus on producing visually plausible rather than accurate results.

The field of cloth modeling and simulation has a very long tradition in computer graphics and there exist a number of surveys and state-of-the-art reports [Gibson and Mirtich 97, House and Breen 00, Magnenat-Thalmann et al. 04, Nealen et al. 05]. This chapter presents an overview of cloth modeling and simulation techniques as well as realistic appearance modeling of clothing, focusing on recent trends, especially on methods that are directly inspired by the *real world* to properly model physical characteristics, deformation behavior as well as appearance of clothing. The chapter follows the pipeline of modeling and rendering photorealistic pieces of cloth (Figure 15.1). Section 15.2 starts with recent advances in modeling the geometry as well as mechanical parameters from real-world cloth samples, followed by Section 15.3 describing cloth simulation and deformation modeling approaches. Section 15.4 focuses on modeling appearance and reflection properties of cloth.

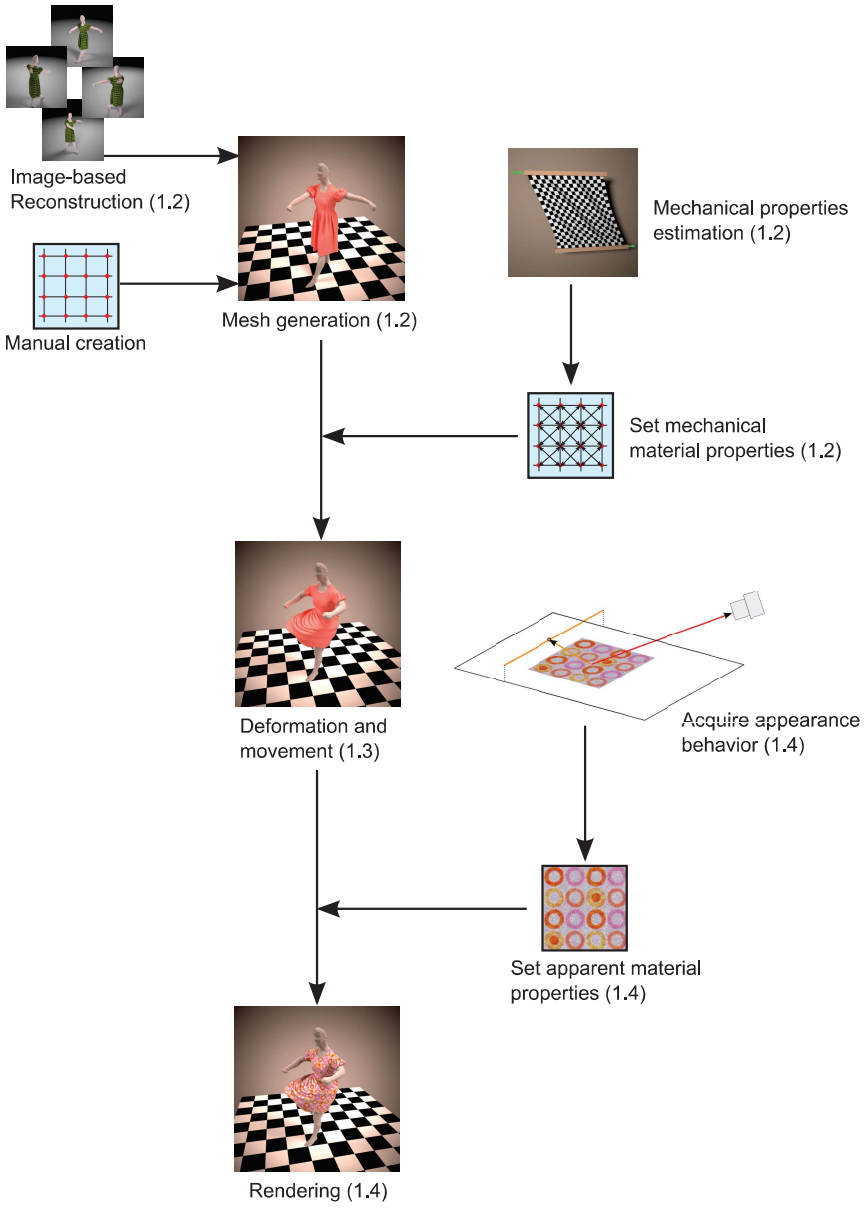


Figure 15.1: Data-driven cloth simulation pipeline.

15.2 Cloth Geometry and Mechanics Modeling

The main goal of cloth modeling is to simulate a piece of cloth as realistically as possible, ideally indistinguishable from real cloth. Usually, modeling clothes relies on a 3D model, e.g., a polygonal mesh, and folds as well

as dynamics of clothes are synthesized. Traditionally, the parameters for the synthesis process are calculated with physics-based models based on a character's pose, motion, external forces as well as material properties (Section 15.3). In these models, the mechanical properties of a specific material are modeled by a number of parameters, describing its resistance to bending, stretching, and shearing. To fully exploit the capabilities of these methods, and to realistically simulate the specific characteristics of different cloth materials, the parameters must be tuned with great care, which is a tedious process if done manually by a professional artist. To achieve a realistic modeling of cloth, several *real-world*-driven (or data-driven) methods have been proposed as an alternative to pure simulation. These methods are directly inspired by the real world to extract the geometry of cloth as well as intrinsic material parameters from visual and/or other sensors. This section gives an overview on modeling cloth geometry and physical properties from real-world samples.

Geometry Modeling

Early approaches focused on capturing and modeling the geometry of moving garments and clothing at high detail from multi-view image or video data (Section 8.4). These approaches generate a detailed and temporally consistent animated mesh model from the input data. For this purpose, reliable correspondences must be established between the different camera views and time steps. One approach to establish these correspondences is to use markers on the piece of clothing [Guskov et al. 03, Scholz et al. 05, White et al. 07] (Figure 15.2). A similar approach is presented in Section 13.5 to reconstruct the shape of a human actor. The nature of the marker layout allows a unique identification of surface patches across the different camera views and timesteps. In order to remove the constraints of a markered cloth sample, which needs to be manufactured specifically for the capturing process, other researchers rely on natural image features, e.g., SIFT or SURF features (Section 7.2), to track surface points over different views and time in multi-view video setups [Pritchard and Heidrich 03, Hasler et al. 06]. However, relying on image features still puts constraints on the captured piece of clothing, as these features require a strongly and preferably anisotropically textured material. This limits the applicability of feature-based approaches to the capturing of highly textured pieces of clothing. Geometry modeling from untextured *off-the-shelf* clothing has been addressed by Bradley et al. [Bradley et al. 08]. By combining imperfectly reconstructed mesh models of every frame in a video sequence with a template mesh, a consistent and complete mesh animation of the captured clothing can be modeled.

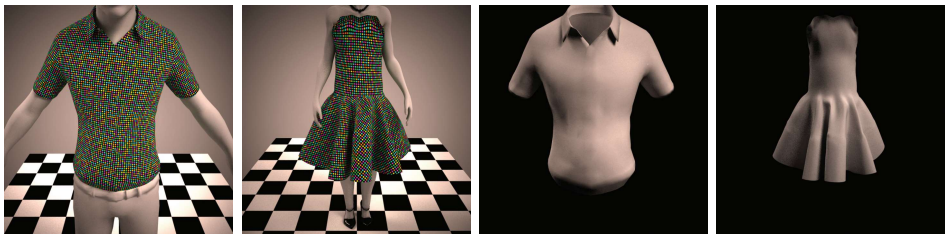


Figure 15.2: Marker-based garment reconstruction according to [Scholz et al. 05]. Left: Input images of one camera in a multi-view setup. Right: Reconstructed 3D models.

Modeling Mechanical Properties

Besides geometry, other approaches try to explicitly extract cloth material parameters describing the physical properties of a piece of cloth directly from visual data [Bhat et al. 03, Kunitomo et al. 10, Bouman et al. 13]. For example, Bhat et al. [Bhat et al. 03] captured video data of a person wearing different pieces of clothing to estimate the location of cloth folds from gradient vector fields in the video frames. The retrieved information is used to determine stiffness parameters of a mass-spring model [Baraff and Witkin 98] (Section 15.3). Similarly, Bouman et al. [Bouman et al. 13] proposed to automatically analyze videos of fabrics moving under various unknown wind forces to estimate stiffness and area weight parameters.

While purely vision-based approaches appeal through a simple setup, it is generally very difficult to measure the true material properties accurately without physical contact and control over external forces. Moreover, it is difficult to separate internal from external parameters. Therefore, recently, data-driven methods have emerged in the literature that measure cloth parameters by applying physical forces to a sample piece of cloth in very controlled conditions [Wang et al. 11a, Miguel et al. 12, Miguel et al. 13]. The external forces are usually measured by force sensors, and deformation is captured by computer vision systems. The information on deformation under controlled forces is then used to fit the parameters of a cloth deformation model to the data. Since these models rely on real-world test data, they can mimic the fabric behavior very realistically during simulation. For example, Wang et al. [Wang et al. 11a] presented a data-driven and piecewise linear model with 39 material parameters to approximate the non-linear, anisotropic deformation behavior of cloth in a continuum-based deformation model [Baraff and Witkin 98] (Section 15.3). By applying external forces to the cloth samples and capturing the planar deformations from camera images, they estimate stretching and bending parameters for ten different samples of cloth strongly varying

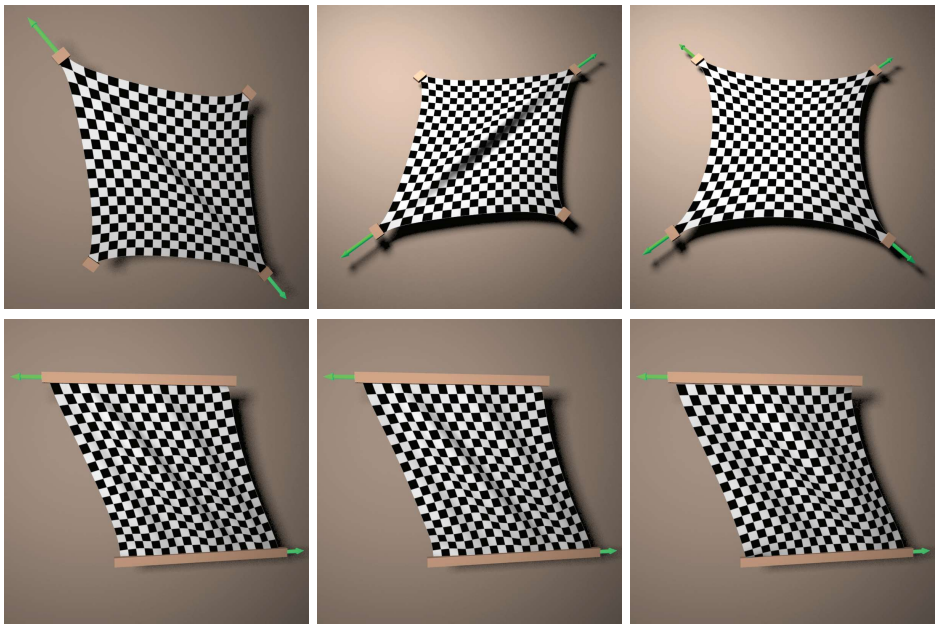


Figure 15.3: Different stretching and bending states of a cloth sample according to the measurement setup of Miguel et al. [Miguel et al. 13]. Top row: Different diagonal forces applied to a cotton sample. Lower row: Horizontal forces applied to cotton, leather and polyester (left to right).

in their stretching and bending behavior. Similarly, Miguel et al. [Miguel et al. 12] developed a system for the estimation of non-linear material parameters of different coefficients of cloth. In contrast to Wang et al., they reconstructed 3D deformation instead of 2D planar deformation using a stereo vision system (Chapter 2, and Figure 15.3). They later extended their work by developing a vision-based technique to estimate internal friction parameters from cloth samples [Miguel et al. 13].

15.3 Cloth Deformation Modeling and Simulation

Having a specific cloth model at hand, a physically correct simulation of garments requires sophisticated simulation techniques to model the deformation behavior. Physics-based methods aim at an accurate and physically correct modeling of deformation as well as wrinkling behavior. However, these methods usually need substantial computation time because highly detailed cloth simulation requires a model with a large number of degrees of freedom. In contrast to that, hybrid and example-based techniques

focus on producing visually plausible and realistic deformation and wrinkling with lower computational complexity.

Physics-Based Deformation Models

A number of different physics-based cloth models have been developed, which estimate the motion and drapery of cloth based on its intrinsic material parameters, as well as external forces, e.g., induced by a character's pose, motion, gravity, collisions, etc. [Terzopoulos et al. 87, Breen et al. 94, Eberhardt et al. 96, Baraff and Witkin 98, Choi and Ko 02]. These models have been widely studied in the past decades, and detailed surveys have been presented in [House and Breen 00, Magnenat-Thalmann et al. 04, Magnenat-Thalmann and Volino 05, Nealen et al. 05, Choi and Ko 05]. The most widely used models are discrete particle models. These models represent a piece of cloth as a discrete set of particles (e.g., vertices in a polygonal mesh) forming the shape of its surface [Breen et al. 94, Eberhardt et al. 96, Baraff and Witkin 98, Bridson et al. 03, English and Bridson 08]. The particle positions are determined by forces applied to their topological neighborhood. One of the most popular particle-based models is the mass-spring model [Provot 95, Choi and Ko 02], which models the interaction between the particles as linear massless springs. The state of the system, i.e., the positions of all particles, is defined by their current positions $\mathbf{x}_i$, their masses m_i as well as their velocities $\mathbf{v}_i$. The forces $\mathbf{f}_i$ on each particle are computed based on external as well as internal forces, defined by the mesh topology. The motion of each particle is then driven by Newton's second law $\mathbf{f}_i = m_i\ddot{\mathbf{x}}_i$. Hence, calculating the position of all particles results in solving a system of ordinary differential equations $\mathbf{M}\ddot{\mathbf{x}}_i = \mathbf{f}(\mathbf{x}, \mathbf{v})$. The matrix $\mathbf{M}$ is a $3n \times 3n$ diagonal mass matrix, where n denotes the number of particles in the system. The solution has to be found by advanced numerical integration methods, which determine the computation performance. The various methods differ in the way the forces are calculated and the time integration is performed [Magnenat-Thalmann and Volino 05, Nealen et al. 05]. While particle systems are an intuitive and simple model for deformation, they are not necessarily accurate and need a large number of particles to model fine-scale deformations. Moreover, these models are known to suffer from *post-buckling instability* when wrinkles are shaped [Choi and Ko 02]. An assumption that compressing forces lead to buckling rather than compression (*immediate buckling assumption*) can solve this problem and lead to improved stability [Choi and Ko 02].

An alternative to particle models are continuum models, which are built on elasticity theory. Very popular models in this class are finite element models [Etzmuss et al. 03, Thomaszewski et al. 09, Volino et al. 09]. These models are based on energy-functions defined on a continuous model, which

are approximated by polynomial patches. Compared to particle-based models, continuum-based models have the advantage that they are parameterization independent, i.e., they can theoretically be applied to arbitrary shaped unstructured meshes, and can reproduce the anisotropic deformation behavior of cloth more accurately. However, these advantages come at the cost of substantially more numerical operations than particle systems.

Hybrid and Data-Driven Deformation Models

With physics-based approaches, modeling and simulating very fine detailed folds and wrinkles requires a very large number of triangles and substantial computation times for both simulation and rendering [Choi and Ko 05]. Therefore, with these methods, usually there is trade-off between speed and quality. Thus, the challenge is to reduce the amount of numerical operations while maintaining visual quality. Exploiting the fact that fine wrinkles and buckles usually appear in local regions on the surface, adaptive remeshing techniques can be used to dynamically refine and coarsen polygonal meshes so that they automatically conform to the geometric and dynamic detail of the simulated cloth [Narain et al. 12]. This can reduce computation time to a fraction of the original time while keeping fine-scale wrinkles.

Various other techniques have been developed to improve the computational speed, giving up the mechanical and physical accuracy of the model and rather focusing on visual realism. Hybrid approaches tackle the trade-off between computation time and simulation quality by combining a low-resolution physical model with geometric detail synthesis. These approaches use a coarse deformable model with reduced detail to quickly generate a plausible simulated animation. To improve the final result, the animated model is augmented with synthetic geometric details, such as folds and wrinkles, which are typically very difficult to simulate in real-time. The position and shape of these synthetic details can for example be based on a deformation analysis of the low-resolution mesh [Hadap et al. 99, Decaudin et al. 06, Rohmer et al. 10, Müller and Chentanez 10], or directly on the image data, e.g., by extracting prominent folding edges [Popa et al. 09].

Recently, data-driven deformation models have become very popular to learn wrinkling and deformation models [Cordier and Magnenat-Thalmann 05, Kim and Vendrovsky 08, Wang et al. 10, Feng et al. 10, Guan et al. 12, Zurdo et al. 13]. The idea of these methods is to establish a mapping between low-resolution and highly detailed cloth models, in such a way that during rendering, a low-resolution mesh can be augmented with predicted fine details that have been learned a priori from highly detailed examples. For this purpose, an appropriately chosen parameterization needs to be defined, describing characteristic wrinkling behavior of the piece of clothing. For example, Kim and Vendrovsky [Kim and Vendrovsky 08] as

well as Wang et al. [Wang et al. 10] focus on tight-fitting clothing for which they assume that wrinkling mainly depends on the articulated pose of a character. Hence, they model the shape of the clothing mesh as a function of pose, i.e., as a function of skeleton or animation parameters (Chapter 11). Similarly, Guan et al. [Guan et al. 12] addressed not only pose- but also body shape-dependency of wrinkling and drapery of clothing. Their approach learns a clothing model from highly detailed physics-based models of clothing on bodies of different shapes and in different poses. During rendering, based on a parameterized model of the human body (Chapter 14), describing its shape and pose, a piece of clothing is synthesized from the learned database. Using more abstract embeddings and directly establishing a relationship between low-resolution and high-resolution cloth deformations instead of assuming pose- or shape-dependence allowed several researchers to develop learning-based models, which are valid also for very loose pieces of clothing, e.g., dresses or skirts, or other types of cloth like flags [Feng et al. 10, Kavan et al. 11, Zurdo et al. 13]. Since temporally consistent cloth models at very high detail together with the required additional information (e.g., pose, body shape, external forces) are difficult to acquire from real cloth samples, these methods rely on highly detailed simulations, giving a known alignment between the test data. However, some efforts have been made to synthesize dynamic wrinkles based on real captured data [Ryan White 07].

Cloth Modeling at the Yarn Level

Common methods approximate cloth as an elastic sheet. In contrast, recent developments model cloth at high detail directly at the yarn level [Kaldor et al. 08, Kaldor et al. 10, Yuksel et al. 12]. Driven by the observation that sheet-based deformation models cannot realistically simulate the complex interactions of yarn loops in real woven material, these methods directly model the material as a complex structure of interwoven yarn. The yarns in the fabric are for example represented as inextensible but flexible spline curves. This allows modeling virtual clothing using common garment layout, exchanging the types of fibers and stitching techniques used in real-world production processes. Single parts of a cloth model can be virtually stitched together, forming a complex piece of cloth that behaves like real woven clothing.

15.4 Cloth Appearance Modeling

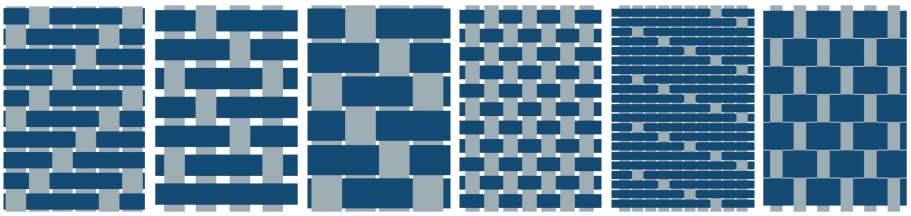
Besides modeling geometric properties and behavior, realistic visual photometric characteristics of the cloth material are important. However,

Table 15.1: Characteristics of different appearance models [Schröder et al. 12].

Model	Surface-based	Volumetric	Explicit
Translucency	✓	✓	✓
Silhouettes		✓	✓
Light Diffusion		✓	✓
Real-time	✓		
Scalability	✓	✓	
Viewing Distance	far/medium	medium	close

modeling the appearance of cloth in high visual quality is a non-trivial task, since the look of a fabric is determined by complex light interaction inside fine surface structures, and often, highly anisotropic single and multiple scattering and self-shadowing effects dominate the appearance. For example, the complex reflection properties of finely structured fabrics like silk or velvet are very difficult to model. This part of the chapter presents approaches which have been inspired by the real world to realistically model the appearance of clothes by reproducing the reflection properties of cloth as closely as possible.

In the last years, different methods for appearance modeling of clothes have been developed, which differ in terms of visual plausibility but also in runtime complexity during rendering. A good overview of appearance models has been presented in [Schröder et al. 12]. The applicability of each method strongly depends on the requirements of the visual quality, runtime and on the viewing distance. Fast methods for high-quality real-time applications or rendering from a far viewing distance may use precomputed bidirectional reflectance distribution functions (BRDFs) or bidirectional texture functions (BTFs). However, with these methods, a specific BRDF or BTF needs to be built for each individual type of fabric.Volumetric models enable a wider field of fabrics and closer viewing distances, but usually need more time and memory to render. For very close viewing distances, explicit or so-called comprehensive models can be used, which allow a visualization of lighting effects on the detailed geometry of the textile on yarn level. This chapter gives an overview of these appearance modeling techniques. The properties of the different models are summarized in Table 15.1.



cotton twill | cotton denim | wool gabardine | polyester lining | silk charmeuse | silk tung | shan-

Figure 15.4: Examples of weaving patterns as used by Irawan [Irawan 08].

Surface-Based Models

Much work for modeling reflection properties of cloth has focused on statistical surface-based models such as bidirectional reflectance distribution functions (BRDFs) and bidirectional texture functions (BTFs) [Müller et al. 04] (Chapter 20). These functions model the reflection properties of a material as a function of viewing and illumination direction and offer a simplified representation of a material. BTF- or BRDF-based methods can be heuristic or data-driven. Among the heuristics models, microfacet models describe the surface as a number of flat micromirrors (facets) which reflect light only in one direction. The BRDF is then determined by the number of visible microfacets at the respective orientation reflecting light to the camera. A generation process for 2D microfacet orientation distributions, for example to model satin and velvet, is described by Ashikmin et al. [Ashikmin et al. 00]. Also, weaving information available from textile computer aided design data can be used to generate specific BRDFs for each fabric material [Adabala et al. 03]. Another heuristic approach is to derive BTFs and BRDFs at the yarn level, as it is done for woven fabrics by Irawan [Irawan 08]. The basis of this approach is a complex empirical model for light interacting with threads of fabric (Figure 15.4). This complex and precise model achieves plausible results for a variety of materials. However, it needs substantial computing time, making it more appropriate for offline ray-tracing methods than for real-time visualization techniques.

Data-driven approaches measure the BRDF or BTF directly from real cloth samples, e.g., by capturing a dataset of images of a surface under different viewing and illumination directions [Sattler et al. 03, Wang et al. 08]. Illumination effects like occlusion, global illumination, self-shadowing, and parallax effects at the yarn level are captured by the images and thereby implicitely incorporated in the measurement data without explicitly modeling them (Figure 15.5). This drastically simplifies the underlying model but requires a defined BTF for every kind of fabric. A publicly available BTF

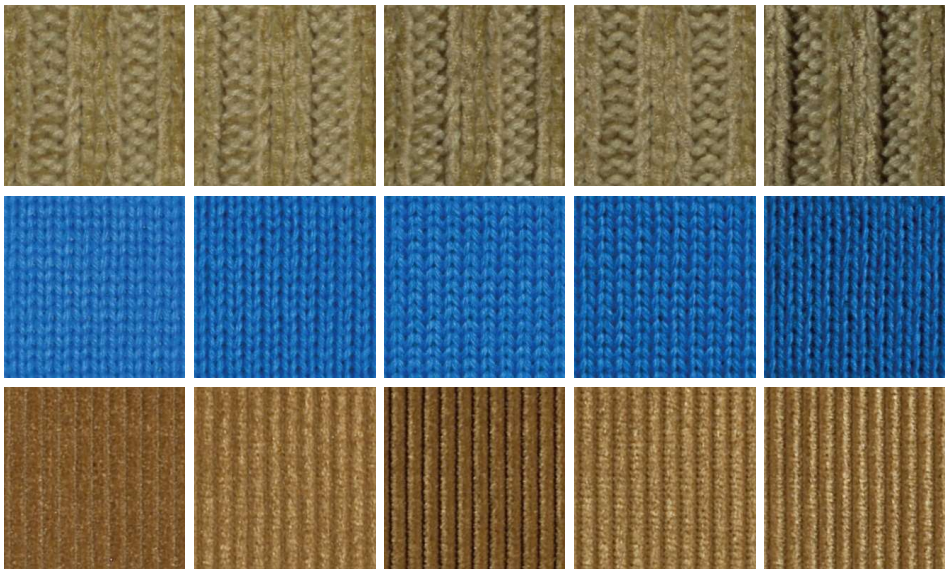


Figure 15.5: Example images of the BTF Database Bonn for three different kinds of cloth under 5 different illumination directions [Sattler et al. 03]. For each sample, 81 views and 81 illumination directions are captured in total. Note the differences in the images regarding self-shadowing and inter-reflections.

dataset has been presented by Sattler et al. [Sattler et al. 03] (Figure 15.5). They captured planar patches of different materials under 81 viewing and 81 illumination directions. In their BTF representation, all images are registered such that a complete set of reflection values is assigned to one 2D image coordinate. To reduce the size of the data, they performed a principal component analysis (PCA) for each of the viewing directions. In a single view approach, Wang et al. [Wang et al. 08] acquired reflectance data using a setup of a single fixed camera and a linear light source. The light source moves above a sample of cloth, and the camera captures the intensity of the reflected light (Figure 15.6). A microfacet-based BRDF is derived by interpolating between the scattered data. The basic assumption for this single view approach is that for any surface point, there is another surface point with similar but rotated microstructure and that, therefore, a single view contains different slices of the BRDF at surface points with similar reflectance. Since this process is related to example-based texture synthesis [Efros and Freeman 01, Kwatra et al. 03], the method is also called example-based microfacet synthesis.

BRDF- or BTF-based appearance modeling can be extremely efficient for simple lighting setups, and for real-time graphics applications, BTFs are

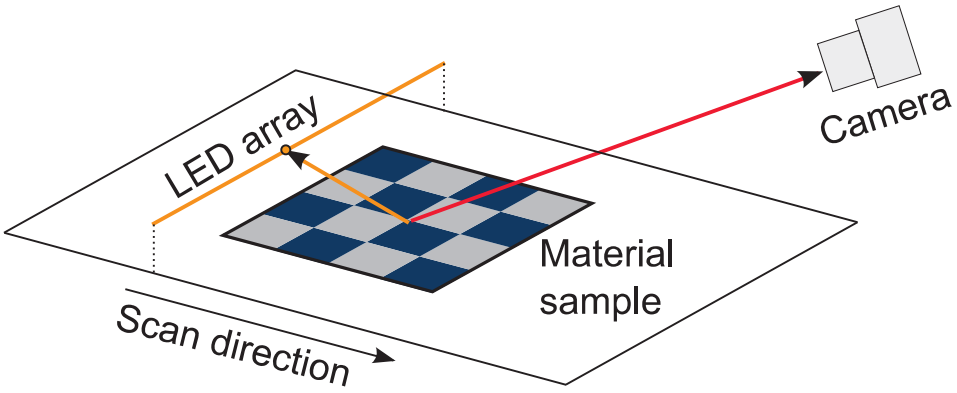


Figure 15.6: The setup of lighting unit, camera, and material sample used for example-based microfacet synthesis [Wang et al. 08].

currently among the most popular techniques. Limitations of BTFs include light diffusion at shadow boundaries, which cannot be reproduced properly. Also, silhouette effects are commonly ignored and handling of transparency is difficult.

Volumetric Models

Volumetric models overcome some of the above-mentioned limitations by additionally capturing the thickness of the fabric. This improves the visual quality when viewing a piece of cloth from a medium or close distance. In the micro-flake model, scattering events in the volume are computed by introducing a directional flake distribution that describes the interaction between particles [Schröder et al. 11, Zhao et al. 11]. This distribution represents the orientation of idealized mirror flakes at any point in the volume. These models can also be driven by real-world measurements of real cloth samples. For example, Zhao et al. [Zhao et al. 11] used CT scans of fabric samples for volumetric rendering. Their approach is able to generate renderings in high quality. However, very sophisticated hardware is needed to record the data for the model.

Explicit Models

In contrast to surface- or volumetric models, explicit models describe the reflection properties by precisely computing light intersections with actual fiber geometry. For example, Irawan and Marschner [Irawan and Marschner 12] presented a comprehensive model to reproduce the look of woven cloth in both small scale and large scale. Their model is based on measurements of the light interaction with fiber threads and can reproduce a wide range of cloth appearances, e.g., black cotton twill, denim,

silk, or polyester. Since the numerical fit to the measured data within their approach is very time-consuming, the number of possible thread directions is reduced. However, this also reduces the number of specular highlights. Sadeghi et al. [Sadeghi et al. 13] extend this model to render more complex highlights. They use a microcylinder appearance model to render anisotropic highlights and color shifts by combining BRDF measurements with scattering profiles. The scattering is represented by a scattered radiance distribution function (BSDF) measured by a spherical gantry. By providing the measured parameters as well as weave definitions for a variety of individual threads of fabrics, e.g., linen, silk, polyester, and velvet, realistic visualization is possible for these types of cloth.

Image-Based Models

An alternative to following the complete pipeline from physically modeling mechanical material parameters over deformation simulation to reflection modeling is to model all these characteristics purely by appearance in image-based representations. These representations use a large database of images of a piece of clothing, e.g., captured from different viewpoints and in different body poses. Similar to image-based BTFs, where the images implicitly capture all reflection properties at fine scale, these images capture geometric properties, like wrinkling behavior and deformation, as well as appearance, e.g., reflection and shading properties, at a larger scale. Hence, a database of images showing a piece of clothing can be used as a database of examples to model a piece of clothing. Early methods that followed this idea focused on retexturing. These methods extract 2D texture deformation and shading from a single image or video frame to render a new piece of clothing under the same deformation and shading conditions [Scholz and Magnor 06, White and Forsyth 06, Hilsmann et al. 10, Hilsmann et al. 11]. While these methods are restricted to the conditions present in the original image (regarding deformation and shading), recent methods learn a mapping between the pose of a human body and appearance of clothing captured by a large number of images [Zhou et al. 12, Hauswiesner et al. 11, Hauswiesner et al. 13, Hilsmann et al. 13]. This way, deformation and appearance of clothes is modeled in a pose-dependent way, and images for new pose configurations can be synthesized and merged from the database. The direct use of real images of clothing allows a photorealistic modeling and rendering with very fine details without the need of computationally demanding simulation (Figure 15.7). However, a dataset must be captured for each individual piece of clothing. Also, as these methods interpolate images from precaptured data, the variety of possible animations and simulations strongly depends on the number and variety of examples in the database.

Figure 15.7: Modeling geometric as well as photometric characteristics of clothing in appearance [Hilsmann et al. 13]. Images of clothes are synthesized based on pose information from a set of pre-recorded database images (below). Different body parts, e.g. the left and the right parts are synthesized from different example images and the final result is merged via image blending.

15.5 Summary

This chapter gives an overview on recent work in cloth modeling and simulation. The ultimate goals of cloth modeling are (i) to produce realistic looking and behaving cloth models that are indistinguishable from real cloth and (ii) to achieve a simulation at an acceptable frame rate, ideally in real-time. Modeling realistic cloth comprises modeling of cloth geometry and mechanical properties, deformation behavior as well as appearance. For each of these steps, different approaches exist, which differ in accuracy and complexity, and for each specific application, different methods are suitable. This chapter focused on recent approaches that try to infer information on cloth properties and appearance from the real world in data-driven and example-based methods. These approaches determine realistic geometry and mechanical properties as well as reflection and appearance properties, to model a piece of cloth as realistically as possible.

16 Video-Based Character Animation

Dan Casas, Peng Huang, and Adrian Hilton

16.1 Introduction

Current interactive character authoring pipelines commonly consist of two steps: modeling and rigging of the character model which may be based on photographic reference or high-resolution 3D laser scans (Chapters 13 and 14); and move-trees based on a database of skeletal motion capture together with inverse dynamic and kinematic solvers for secondary motion. Motion graphs [Kovar et al. 02] and parameterized skeletal motion spaces [Heck and Gleicher 07] enable representation and real-time interactive control of character movement from motion capture data. Authoring interactive characters requires a high-level of manual editing to achieve acceptable realism of appearance and movement.

Chapters 11 and 12 introduced recent advances in performance capture [Starck and Hilton 07b, Gall et al. 09] that have demonstrated highly realistic reconstruction of motion using a temporally coherent mesh representation across sequences, referred to as 4D video. This allows replay of the captured motions with free-viewpoint rendering and compositing of performance in post-production while maintaining photo-realism. Captured sequences have been exploited for retargeting surface motion to other characters [Baran et al. 09] and analysis of cloth motion to simulate novel animations through manipulation of skeletal motion and simulation of secondary cloth movement [Stoll et al. 10]. However, these approaches do not enable authoring of interactive characters which allow continuous movement control and reproduction of secondary motion for clothing and hair.

This chapter presents a framework for authoring interactive characters based on actor performance capture. A Surface Motion Graph representation [Huang et al. 09] is presented to seamlessly link captured sequences, allowing authoring of novel animations from user specified space-time constraints. A 4D parametric motion graph representation [Casas et al. 13] is described for real-time interactive animation from a database of captured

4D video sequences. Finally, a rendering approach referred to as 4D video textures [Casas et al. 14] is introduced to synthesize realistic appearance for parametric characters.

16.2 Surface Motion Graphs

Multiple view reconstruction of human performance as a 3D video has advanced to the stage of capturing detailed non-rigid dynamic surface shape and appearance of the body, clothing, and hair during motion [Starck and Hilton 07b, de Aguiar et al. 08b, Vlasic et al. 08, Gall et al. 09] (Chapters 11 and 12). Full 3D video scene capture holds the potential to create truly realistic synthetic animated content by reproducing the dynamics of shape and appearance currently missing from marker-based skeletal motion capture. There is considerable interest in the reuse of captured 3D video sequences for animation production. For conventional skeletal motion capture (MoCap), *motion graph* techniques [Molina-Tanco and Hilton 00, Kovar et al. 02, Arikan and Forsyth 02, Lee et al. 02] are widely used in 3D character animation production for games and film. This section presents a framework that automatically constructs motion graphs for 3D video sequences, called *surface motion graphs* [Huang et al. 09], and synthesizes novel animations to best satisfy user specified constraints on movement, location, and timing. Figure 16.1 shows an example novel animation sequence produced using a surface motion graph with path optimization to satisfy the user constraints.

Character Animation Pipeline

A frame-to-frame similarity matrix is first computed between all frames across all 3D video motion sequences in the 3D video database. Potential transitions between motions are automatically identified by minimizing the total dissimilarity of transition frames. The idea behind this is to minimize the discontinuity that may be introduced by transitions when transferring

Figure 16.1: An example of a synthesized 3D character animation (10 transitions).

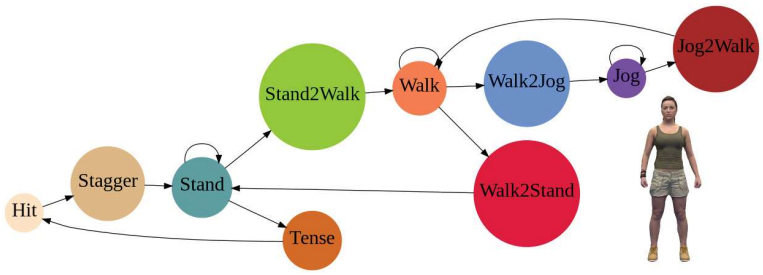


Figure 16.2: An example of a surface motion graph for a game character.

within or across different motions. A surface motion graph is then constructed using these transitions. Once the graph structure is computed, a path on the graph is optimized according to user input such as key-frames, global timing, and distance constraints. Finally, concatenative motion synthesis and rendering is performed to produce video-realistic character animation.

Graph Representation

A surface motion graph represents possible inter- and intra-sequence transitions for 3D video sequences, analogous to motion graphs [Kovar et al. 02] for skeletal motion capture sequences. It is defined as a directed graph: each node denotes a 3D video sequence; each edge denotes one or multiple possible transitions. A path on the graph then provides a possible motion synthesis. Figure 16.2 shows an example of surface motion graph constructed from multiple 3D video motion sequences for the actor shown. The method to automatically identify transitions is described in Section 16.2.

Graph Optimization

Graph optimization is performed to find the path through the surface motion graph which best satisfies the required animation constraints. Intermediate key-frames selected by the user provide hard constraints defining the desired movement. Start and end key-frame locations specify the target traverse distance d_V and the target traverse time t_V chosen by the user. Both target traverse distance and time are used as soft constraints, which define global constraints on the animation. The cost function for graph path optimization to satisfy the constraints is described as follows:

Combined Cost. The cost function for a path P through the surface motion graph between a pair of key frames is formulated as the combination of three costs, C_{tran} representing cost of transition between motions, soft

constraints on distance C_{dist} and time C_{time},

$$C(P) = C_{tran}(P) + w_{dist} \cdot C_{dist}(P) + w_{time} \cdot C_{time}(P). \tag{16.1}$$

where w_{dist} and w_{time} are weights for distance and time constraints, respectively. Throughout this chapter, the parameters are set to $w_{dist} = 1/0.3$ and $w_{time} = 1/10$, which equates the penalty for an error of 30 cm in distance with an error of 10 frames in time [Arikan and Forsyth 02].

Distance Cost. $C_{dist}(P)$ for a path P with N_f frames on the surface motion graph is computed as the absolute difference between the user-specified target distance d_V and the total travelled distance $dist(P)$, given the 3D frames on the path of P is $\{M(t_f)\}$,$f = [0, N_f - 1]$,

$$C_{dist}(P) = |dist(P) - d_V|. \tag{16.2}$$

$$dist(P) = \sum_{f=0}^{N_f-2} |center(M(t_{f+1})) - center(M(t_f))|. \tag{16.3}$$

where function $center()$ computes the projection of the centroid of the mesh onto the ground.

Timing Cost. $C_{time}(P)$ for a path P with N_f frames is evaluated as the absolute difference between the user-specified target time t_V and the total travelled time $time(P)$,

$$C_{time}(P) = |time(P) - t_V|. \tag{16.4}$$

$$time(P) = N_f \cdot \Delta t. \tag{16.5}$$

where Δt denotes the frame rate (e.g., 25 frames per second).

Transition Cost. $C_{tran}(P)$ for a path P is defined as the sum of distortion for all transitions between concatenated 3D video segments. If the index for concatenated 3D video segments is denoted $\{f_i\}$, $i = 0, ..., N_{f-1}$, the total transition cost $C_{tran}(P)$ is computed as

$$C_{tran}(P) = \sum_{i=0}^{N_P-2} D(S_{f_i \to f_{i+1}}). \tag{16.6}$$

where N_P denotes the total number of transitions on path P and $D(S_{f_i \to f_{i+1}})$ from Eq. (16.9) is the distortion for transition from motion sequence S_{f_i} to motion sequence $S_{f_{i+1}}$.

Path Optimization. Finally, the optimal path P^{opt} for a given set of constraints is found to minimize the combined cost $C(P)$, as defined in Equation 16.1,

$$P^{opt} = \arg\min_{P} C(P). \tag{16.7}$$

An efficient approach using interger programming to search for the optimal path that best satisfies the user-defined soft constraints can be found in [Huang et al. 09].

Transitions

A transition of the surface motion graphs $S_{i\to j}$ is defined as a seamless concatenation of frames from two 3D video sequences S_i and S_j. If m, n denotes the central indices for the overlap, the length of overlap as $2L + 1$, the blending weight for the k^{th} transition frame is computed as $\alpha(k) = \frac{k+L}{2L}$, $k \in [-L, L]$. The k^{th} transition frame $M_{i\to j}(t_k) = G(M_i(t_{m+k}), M_j(t_{n+k}), \alpha(k))$ will be generated by a non-linear 3D mesh blend [Tejera et al. 13]. The distortion measure of a transition frame $M_{i\to j}(t_k)$ is then defined as the weighted 3D shape dissimilarity (Equation 16.11) between frame $M_i(t)$ and $M_j(t)$,

$$d(M_{i\to j}(t_k)) = \alpha'(k) \cdot c(M_i(t_{m+k}), M_j(t_{n+k})). \tag{16.8}$$

where $\alpha'(k) = \min(1 - \alpha(k), \alpha(k))$. The total distortion for a transition sequence $S_{i\to j}$ is then computed as the sum of the distortion of all transition frames,

$$D(S_{i\to j}) = \sum_{k=-L}^{L} d(M_{i\to j}(t_k)). \tag{16.9}$$

The optimal transition $S^{opt}_{i\to j}$ is then defined to minimize the distortion cost,

$$S^{opt}_{i\to j} = \arg\min_{S_{i\to j}} D(S_{i\to j}). \tag{16.10}$$

3D Shape Similarity. To measure the similarity of the 3D mesh geometry within each frame of the 3D video database, shape histograms are used. This has previously demonstrated to give good performance for measuring non-rigid deformable shape similarity for 3D video sequences of human performance [Huang et al. 10]. The volumetric shape histogram $H(M)$ represents the spatial occupancy of the bins for a given mesh M. A measure of shape dissimilarity between two meshes M_r and M_s can be defined by optimizing for the maximum overlap between their corresponding radial bins with respect to rotation about the vertical axis.

$$c_{shape}(M_r, M_s) = \min_{\phi} \|H(M_r) - H(M_s, \phi)\|. \tag{16.11}$$

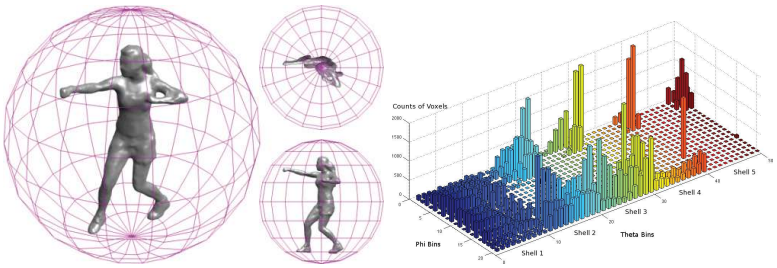


Figure 16.3: A 3D shape histogram of 5 shells, 10 bins for θ, and 20 bins for ϕ together with the space partitions on the 5th shell is illustrated.

The volumetric shape histogram $H(M)$ partitions the space which contains a 3D object into disjoint cells and counts the number of occupied voxels falling in each bin to construct a histogram as a signature for this 3D object. The space is represented in a spherical coordinate system (R, θ, ϕ) around the center of mass. An example of a 3D shape histogram is shown in Figure 16.3.

A frame-to-frame static similarity matrix $\mathbf{C} = [c_{r,s}]_{N \times N}$, where N denotes the total number of frames, between all frames across all 3D video sequences in the 3D video database is pre-computed according to Eq.16.11,

$$c_{r,s} = c_{shape}(M_r, M_s). \tag{16.12}$$

Adaptive Temporal Filtering. Each transition is determined by a tuple (m, n, L). The global optimization is then performed by testing all possible tuples (m, n, L) and so finding optimal arguments for the minimum,

$$(m^{opt}, n^{opt}, L^{opt}) = \arg \min_{m,n,L} \sum_{k=-L}^{L} \alpha'(k) \cdot c_{m+k,n+k}. \tag{16.13}$$

This equates to performing an adaptive temporal filtering with window size $2L + 1$ and weighting $\alpha'(k)$ on the pre-computed static similarity matrix $\mathbf{C}$. The process is computationally efficient. To obtain multiple transitions between a pair of sequences, top T best transitions are preserved. T is pre-determined by the user. An example of the frame-to-frame 3D shape similarity matrix and transition frames (marked in yellow) evaluated using a temporal window $L = 4$ and $T = 4$ are shown in Figure 16.4.

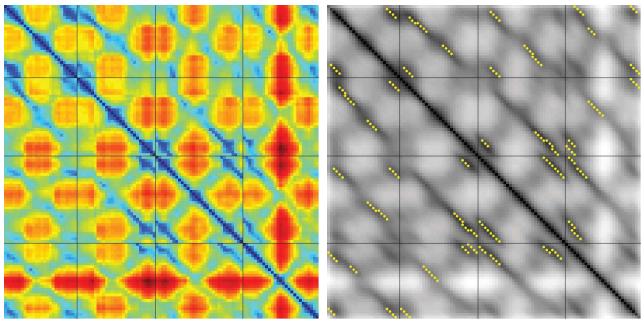


Figure 16.4: Shape similarity matrix and identified transitions for motions *Jog*, *Jog2Walk*, *Walk*, and *Walk2Jog* (left to right and top to bottom) .

16.3 4D Parametric Motion Graphs

Parametric Control of Mesh Sequences

Parametric animation requires the combination of multiple captured mesh sequences to allow continuous real-time control of movement with intuitive high-level parameters such as speed and direction for walking or height and distance for jumping. Methods for parameterization of skeletal motion capture have previously been introduced [Kovar et al. 02, Rose et al. 98] based on linear interpolation of joint angles. Analogously, as described by Casas et al. [Casas et al. 12b], parametric animation of mesh sequences can be achieved by interpolation between mesh sequences.

This requires temporal alignment of frames across multiple 3D video mesh sequences, such that all frames have a consistent mesh connectivity and vertices correspond to the same surface point over time. The set of temporally aligned 3D video sequences are referred to as 4D video [Budd et al. 13] (Chapter 11).

Given a set of N temporally aligned 4D mesh sequences $\mathbf{M} = \{M_i(t)\}_{i=1}^{N}$ of the same or similar motions (e.g., high jump and low jump), some sort of parametric control is sought by blending between multiple mesh sequences

$$M_B(t, \mathbf{w}) = b(\mathbf{M}, \mathbf{w}) \tag{16.14}$$

where $\mathbf{w} = \{w_i\}_{i=1}^{N}, w_i \in [0..1]$ is a vector of weights for each input motion and $b()$ is a mesh sequence blending function. This function must perform at online rates, ≥ 25 Hz, and the resulting mesh $M_B(t, \mathbf{w})$ to maintain the captured non-rigid dynamics of the source $\{M_i(t)\}_{i=1}^{N}$ meshes.

Three steps are required to achieve high-level parametric control from mesh sequences: time-warping to align the mesh sequences; non-linear mesh

blending of the time-warped sequences; and mapping from low level blending weights to high-level parameters (speed, direction, etc.).

Sequence Time-Warping. Each 4D video sequence of related motions (e.g., walk and run) is likely to differ in length and location of corresponding events, for example foot-floor contact. Thus, the first step for mesh sequence blending is to establish the frame-to-frame correspondence between different sequences. Mesh sequences $M_i(t)$ are temporally aligned by a continuous time-warp function $t = f(t_u)$ [Witkin and Popovic 95] which aligns corresponding poses of related motions prior to blending such that $t \in [0, 1]$ for all sequences. Temporal sequence alignment helps in preventing undesirable artifacts such as foot skating in the final blended sequence.

Real-Time Mesh Blending. Previous research in 3D mesh deformation concluded that linear-methods for mesh blending, despite being computationally efficient, may result in unrealistic results [Lewis et al. 00]. Non-linear methods based on differential surface representation [Botsch and Sorkine 08] overcome this limitation, achieving plausible surface deformation. However, the price paid is a significant increase in processing requirements, hindering their use for online applications. Piecewise linear blending methods [Casas et al. 13] have demonstrated successful results for online applications by precomputing a set of non-linear interpolated meshes. At run time, any requested parametric mesh is computed by linearly blending the relevant offline interpolated meshes. This results in an approximation to the robust non-linear blending approach with a computational cost similar to the linear approach.

High-Level Parametric Control. High-level parametric control is achieved by learning a mapping function $f(\mathbf{w})$ between the blend weights $\mathbf{w}$ and the user specified motion parameters $\mathbf{p}$. A mapping function $\mathbf{w} = f^{-1}(\mathbf{p})$ is learned from the user-specified parameter to the corresponding blend weights required to generate the desired motion because the blend weights $\mathbf{w}$ do not provide an intuitive parameterization of the motion. Motion parameters $\mathbf{p}$ are high-level user specified controls for a particular class of motions such as speed and direction for walk or run, and height and distance for a jump. As proposed by Ahmed et al. [Ahmed et al. 01], the inverse mapping function f^{-1} from parameters to weights can be constructed by a discrete sampling of the weight space $\mathbf{w}$ and evaluation of the corresponding motion parameters $\mathbf{p}$.

Figure 16.5 presents three examples of mesh sequence parameterization, enabling control of speed, jump length, and jump height, respectively. Change in color represents change in parameterization. In each example,

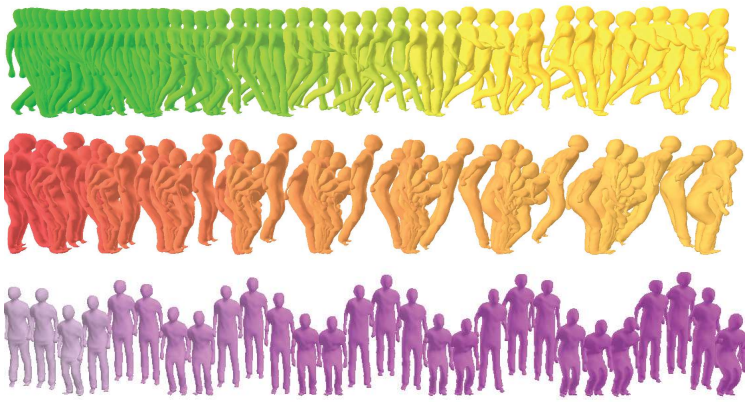


Figure 16.5: Mesh sequence parameterization results [Casas et al. 12b]. In each row, two input motions are interpolated, generating in-between parametric motion control. Top row, speed control; middle row, jump length control; bottom row, jump height control.

two motions are interpolated to generate in-between poses, enabling interactive parametric control of the motion.

4D Parametric Motion Graphs

Given a dataset of mesh sequences for different movements with a consistent mesh structure at every frame, referred to as 4D video, parametric control of the motion can be achieved by combining multiple sequences of related motions (Section 16.3). This gives a parameterized motion space controlled by high-level parameters (for example walk speed/direction or jump height/length). However, parametric motions can only be synthesized by combining semantically similar sequences (i.e., walk and run), whereas the interpolation of non-similar sequences, such as jump and walk, would fail in generating a human-realistic motion. Therefore, in order to fully exploit a dataset of 4D video sequences, methods for linking motions performing different actions are required.

An approach referred to as 4D parametric motion graph, 4DPMG, [Casas et al. 12a, Casas et al. 13] tackles this problem by employing a graph representation that encapsulates a number of independent mesh parametric spaces as well as links between them, to enable real-time parametric control of the motion. Figure 16.6 presents an illustration of a 4DPMG created using 10 different motions used to create 4 parametric spaces.

The nodes of a 4DPMG are created by the combination of similar motions. Each node can be considered as an independent parametric motion space created (Section 16.3). The problem is then how to find transitions

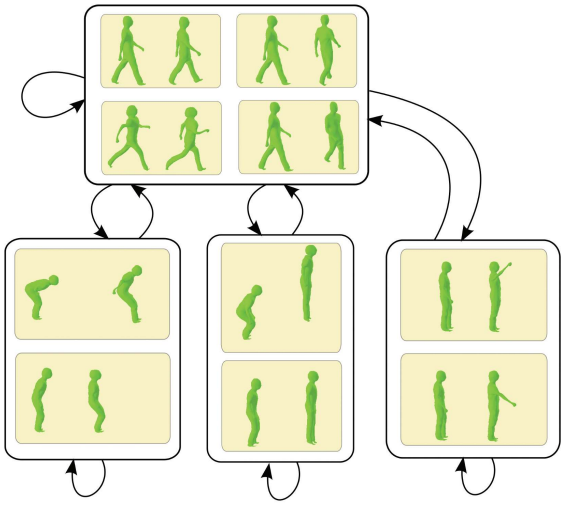


Figure 16.6: Illustration of a 4D parametric motion graph. Ten different motions are combined to create a 4-node 4DPMG, enabling speed, direction, jump, and reach parametric control. Each node represents an independent parametric space; edges represent links between them.

between parametric motions at run time. Natural transitions require a similar shape and non-rigid motion between the linked meshes, otherwise the resulting animation will not look realistic due to sudden change of the character's speed and pose. In the literature, parametric transitions for skeletal data have been approached by precomputing a discrete set of *good* transitions, evaluating the similarity in pose and motion between pairs of the linked motion spaces [Heck and Gleicher 07]. However, precomputation of a fixed set of transition points may result in a relatively high latency due to the delay between the current pose and the next pre-computed good-transition pose.

In a 4DPMG, to evaluate the best transition path P_{opt} between two parametric points, a cost function representing the trade-off between similarity in mesh shape and motion at transition, $E_S(P)$, and the latency, $E_L(P)$, or delay in transition for a path P, is optimized

$$P_{opt} = \arg\min_{P \in \Omega} \left(E_S(P) + \lambda E_L(P) \right) \tag{16.15}$$

where λ defines the trade-off between transition similarity and latency. The transition path P is optimized over a trellis of frames as illustrated in Figure 16.7, starting at the current frame $M^s(t^s, \mathbf{w}^s)$ in the source motion space and a trellis ending at the target frame $M^d(t^d, \mathbf{w}^d)$ in the target motion space, where M^s and M^t are interpolated meshes as defined by Eq. (16.14).

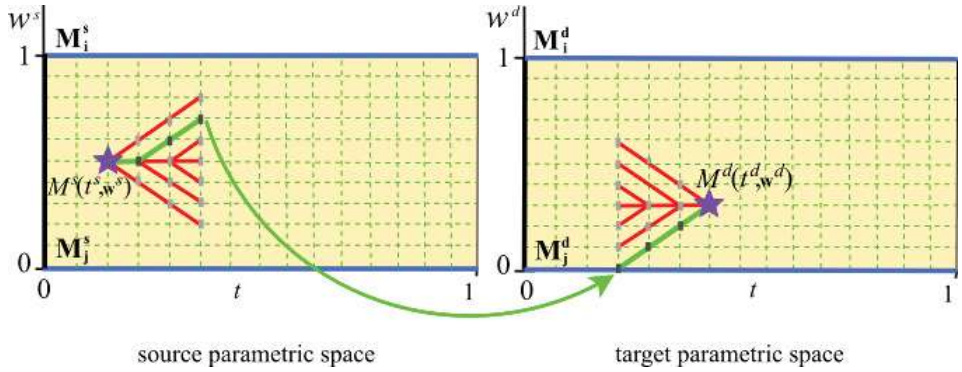


Figure 16.7: Diagram depicting the process of finding a transition between two parametric meshes, $M^s(t^s, \mathbf{w}^s)$ and $M^d(t^d, \mathbf{w}^d)$, here arbitrarily selected for illustration purposes, and marked with a purple star inside their corresponding parametric spaces. Two trellises, depicted in red in each parametric space, are built to find the sets of transition mesh candidates. Optimal transition path P_{opt}, also arbitrarily selected in this diagram, is highlighted in green over the trellis.

The trellis is sampled forward in time at discrete intervals in time Δt and parameters $\Delta \mathbf{w}$ up to a maximum depth l_{max} in the source space. Similarly from the target frame a trellis is constructed going backward in time. This defines a set of candidate paths $P \in \Omega$ with transition points between each possible pair of frames in the source and target trellis.

For a path P, the latency cost $E_L(P)$ is measured as the number of frames in the path P between the source and target frames. Transition similarity cost $E_S(P)$ is measured as the similarity in mesh shape and motion at the transition point between the source and target motion space for the path P. Online mesh similarity computation is prohibitively expensive for large sets of transition candidates. To overcome this, Casas et al. [Casas et al. 12a] proposed a method based on precomputing a set of similarities between the input data, and interpolated them at runtime to approximate any requested parametric pose similarity.

Figure 16.8 presents a parametric character interactively synthesized combining 10 different motions. Qualitative and quantitative results using 4DPMG have demonstrated [Casas et al. 12a] realistic transitions between parametric mesh sequences, enabling interactive parametric control of a 3D-mesh character.

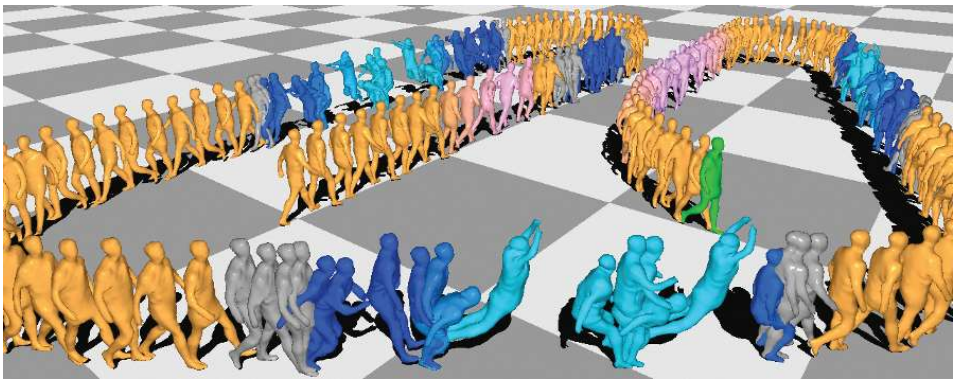


Figure 16.8: An interactively controlled character generated using 4DPMG, combining walk, jog, left turn, right turn, short jump, and long jump motions. Grey meshes indicate transitions between parametric spaces.

4D Video Textures

The 4D parametric motion graph allows real-time interactive control of a character's motion to produce novel animation sequences. However, this is missing the realistic appearance of the source video for free-viewpoint rendering. To achieve video-realistic rendering of the dynamic appearance for 4D parametrically controlled animation a representation referred to as 4D video textures (4DVT) is used [Casas et al. 14].

With 4DVT, appearance for an intermediate motion is produced by aligning and combining multiple-view video from the input examples to produce plausible video-realistic dynamic appearance corresponding to the modified movement. As the character motion changes, so does the dynamic appearance of the rendered view reflecting the change in motion.

A 4D video $F(t) = \{V(t), M(t)\}$ combines multiple view video sequences $V(t) = \{I_c(t)\}_{c=1}^{C}$ with C camera views with a 4D proxy of the dynamic surface shape represented as a mesh sequence $M(t)$, where vertices correspond to the same surface point over time. This form of representation has previously been employed for free-viewpoint video rendering of dynamic scenes [Carranza et al. 03, Starck and Hilton 07b, de Aguiar et al. 08b, Vlasic et al. 08]. Free-viewpoint video renders a novel video sequence $I(t, \mathbf{v})$ for a viewpoint $\mathbf{v}$ from the set of input videos $V(t)$ using the mesh sequence $M(t)$ as a geometric proxy [Buehler et al. 01]. The objective of free-viewpoint video is to maintain the visual realism of the source video while providing the flexibility to interactively control the viewpoint. However, free-viewpoint video is limited to the replay of the captured performance and does not allow for any change in scene motion.

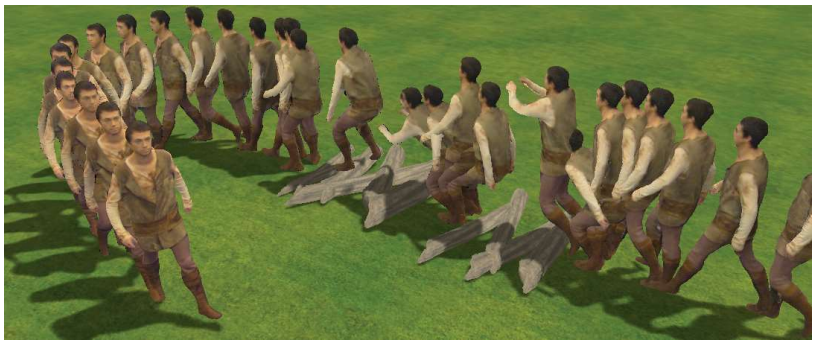


Figure 16.9: A character animated using 4D Video Textures jumping over obstacles of varying length. Notice the texture detail in face and clothing.

4DVT overcomes this limitation [Casas et al. 14]. Given a set of motion control parameters $\mathbf{w}$ and viewpoint $\mathbf{v}$, the aim is to render a novel video $I(t, \mathbf{w}, \mathbf{v})$:

$$I(t, \mathbf{w}, \mathbf{v}) = h(F_1(t), ..., F_N(t), \mathbf{w}, \mathbf{v}), \tag{16.16}$$

where $h(.)$ is a function which combines the source 4D videos according to the specified motion parameters $\mathbf{w}$ and viewpoint v. The rendered video $I(t, \mathbf{w}, \mathbf{v})$ should preserve the visual quality of both the scene appearance and motion.

A two-stage approach is used to synthesize the final $I(t, \mathbf{w}, \mathbf{v})$ video. First, a 4D shape proxy $M(t, \mathbf{w})$ is computed by the combination of the input mesh sequences using the approach presented in Section 16.3. Finally, exploiting the known vertex-to-vertex correspondence across sequences, view-dependent rendering of the source videos $V_i(t)$ is performed using the same 4D shape proxy $I(t, \mathbf{w}, \mathbf{v})$. The output video $I(t, \mathbf{w}, \mathbf{v})$ is generated based on real-time alignment of the rendered images.

Qualitative and quantitative results have demonstrated that 4DVT succesfully enables the creation of video characters with interactive video and motion control and free-viewpoint rendering which maintain the visual quality and dynamic appearance of the source videos [Casas et al. 14]. Intermediate motions are rendered with plausible dynamic appearance for the whole-body, cloth wrinkles, and hair motion. Figure 16.9 presents an animation interactively created combining 4D video textures and 4D parametric motion graph; a character combines different styles of jumps and walks to avoid the obstacles, and finally performs a sharp left turn.

16.4 Summary

Recent research on multi-camera performance capture has enabled detailed reconstruction of motions as 3D mesh sequences with a temporally coherent geometry. This chapter has presented a set of methods to allow the reutilization of reconstructed motions, with the goal of authoring novel character animation while maintaining the realism of the captured data.

A representation referred to as surface motion graph has been introduced to synthesize novel animations that satisfy a set of user-defined constraints on movement, location, and timing. A graph optimization technique is used to find transitions between originally captured sequences that best satisfy the animation constraints. A 3D shape similarity descriptor is used to evaluate the transitions between different motions.

A parametric approach referred to as 4D parametric motion graph has been presented to synthesize novel interactive character animation by concatenating and blending a set of mesh sequences. This enables real-time control of a parametric character that preserves the realism of the captured geometry. Video-realistic appearance for novel parametric motions is generated at run-time using 4D video textures, a rendering technique that combines multi-camera captured footage to synthesize textures that changes with the motion while maintaining the realism of the captured video.

Part IV

Authentic Rendering, Display, and Perception

17
Image- and Video-Based Rendering

Christian Lipski, Anna Hilsmann, Carsten Dachsbacher, and Martin Eisemann

17.1 Introduction

The purpose of image- and video-based rendering (IVBR) is to be able to synthesize photo-realistic new, virtual views of real-world scenes and events from no more than a set of conventional photographs or videos of the scene. Purely image-based, or plenoptic, approaches densely sample scene appearance using a large number of images (Sections 17.2 and 5.3). New views are generated by simply re-sampling the captured image data. In contrast, geometry-assisted methods require much less input images. Here, 3D scene geometry is either a priori known, reconstructed from the captured imagery, or acquired separately by some other means, e.g., ranging imaging (Chapter 4). With (approximate) scene geometry available, views from arbitrary viewpoints are synthesized using the acquired images as geometry texture (Section 17.3). Over the years, various IVBR methods have been proposed that can all be categorized in-between these two limiting cases [Lengyel 98] (Figure 17.1). Their respective advantages and limitations have been discussed in several IVBR surveys [Shum and Kang 00, Smolic et al. 09, Linz 11, Germann 12].

17.2 Plenoptic Approaches

Light Fields and Lumigraphs

The idea underlying light field rendering is to represent the plenoptic function of a real-world scene, i.e., its appearance from any direction, as a four-dimensional lookup table [Levoy and Hanrahan 96] (Figure 17.2). By assuming the space around the scene to be transparent, each light ray can be parameterized by four scalar values (u, v, s, t). (u, v) are the intersection

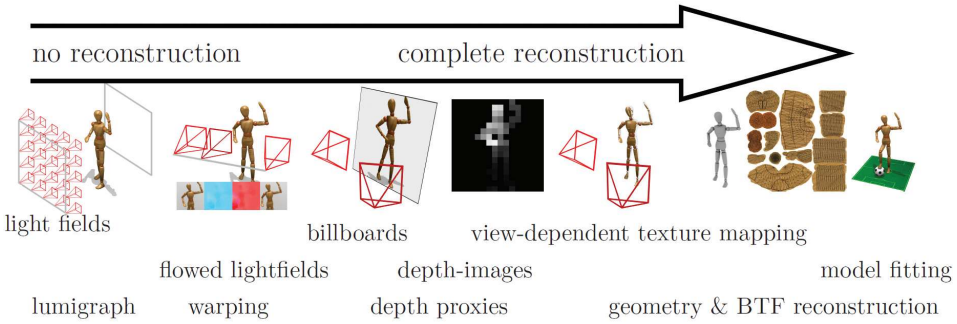


Figure 17.1: Overview of video and image-based rendering systems. While some approaches are based on densely sampling scene appearance with many images (far left), others rely on having available high-quality 3D scene geometry (far right). Numerous techniques can be located somewhere between these two limiting cases.

coordinates of the light ray with the camera plane, while (s, t) are the ray's intersection coordinates with the fronto-parallel image plane. Light field acquisition consists of sampling the uv plane by taking images from regularly spaced uv grid positions (Figure 17.3). For high-quality light field acquisition, a rectifying homography transformation is typically applied to align the image plane of all captured images with the st plane [Kim et al. 13].

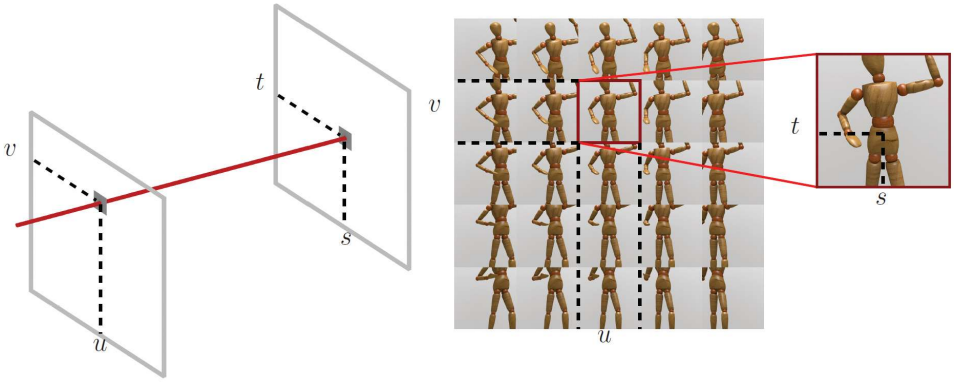


Figure 17.2: Light field rendering: views of the scene from arbitrary vantage points are obtained by tracing a view ray for each pixel and determining the intersection coordinates with the uv- and the st-planes (left). The $uvst$ coordinates are used to look up pixel color from the 4D light field dataset (right).

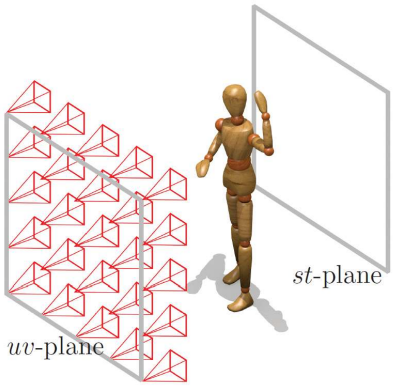


Figure 17.3: Light field acquisition: in the two-parallel-plane parameterization, the plenoptic function of a scene is regularly sampled by camera positions on the uv plane and coinciding image plane st. To acquire the light field of a static scene, a motorized camera gantry can be used (Section 5.2), while for dynamic scenes an array of video cameras is needed.

For static scenes, a mechanical gantry enables sequentially capturing many light field images with one camera. The capture hardware has to be precisely calibrated, however, and the capture process may take a long time. Alternatively, single-chip light field cameras have been propsed [Rodriguez-Ramos et al. 11, Lytro, Inc. 12, Wietzke 12] that employ lenslet arrays to acquire the entire light field simultaneously (Chapter 5). With single-chip systems, however, there is a trade-off between image resolution (st plane) and viewpoint range (uv plane).

Several modifications and extensions to light field rendering have been proposed. In Lumigraph rendering, capturing the scene is simplified by allowing for non-regular placement of cameras [Gortler et al. 96]. Before rendering, the image data is re-parameterized to the (u, v, s, t) representation in a rebinning step. Following a similar approach, light fields can also be captured using mobile phones [Davis et al. 12].

Light field rendering requires neither scene geometry nor image correspondence information. In theory, photo-realistic, high-quality rendering results can be obtained. For aliasing-free rendering, however, unrealistically high sampling rates are required [Chai et al. 00]. To reduce discretization artifacts when light field-rendering from undersampled data, filtering schemes can be employed [Stewart et al. 03, Eisemann et al. 07]. On the other hand, light field data is highly redundant which allows for efficient compression, storage, and streaming [Vaish and Adams 12].

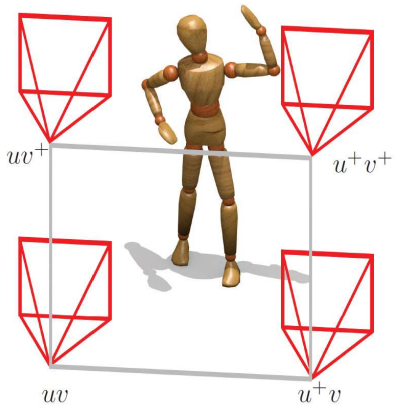


Figure 17.4: Sparse light field capture: with increasing distance between camera positions, uv plane sampling becomes less dense. To avoid ghosting artifacts during rendering, disparity between light field images must be compensated.

Flowed Light Field Rendering

To render from only sparsely sampled light fields, image warping can be employed to synthesize in-between camera positions on the uv-plane [Shade et al. 98, Heidrich et al. 99, Einarsson et al. 06] (Figure 17.4). Prior to rendering, dense image correspondences are established between adjacent light field images by estimating the optical flow fields. During rendering, each ray is intersected with the uv-plane, but instead of just looking up pixel color in the closest-by light field image or linear blending, backward warping is applied to correct for parallax. In backward warping, the pre-computed flow fields are first individually forward-warped to the desired location [Shade et al. 98]. For each ray intersection with the uv plane, the flow vectors to all input images are then looked up and the corresponding light field image pixels are weightedly blended (Figure 17.5).

Flowed light field rendering requires considerably fewer input images than standard light field rendering. For a full 360° surround capture of an actor, 3×30 images are sufficient to obtain convincing rendering results [Einarsson et al. 06]. In addition, the warping step can correct for small errors, e.g., misaligned images due to calibration errors or unsynchronized cameras. This way, even dynamic scenes may be light field-captured sequentially from different vantage points [Einarsson et al. 06].

The main limitation of flowed light field rendering is its dependency on correct, dense flow fields. With increasing distance between neighboring camera positions or for complex scenes, however, optical flow estimation algorithms tend to fail. Another limitation is that backward warping cannot

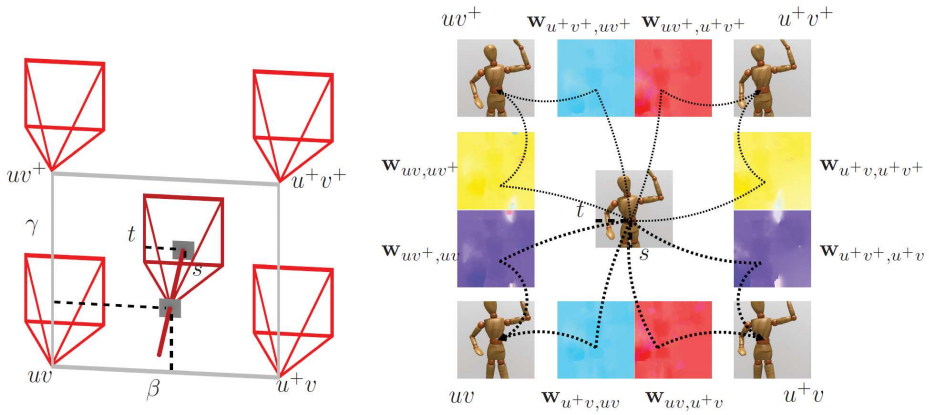


Figure 17.5: Flowed light field rendering. For each viewing ray that intersects the uv plane, the four surrounding images $u[^+]v[^+]$ are determined. In a pre-processing step, optical flow fields $\mathbf{w}_{u[^+]v[^+],u[^+]v[^+]}$ have been computed between adjacent light field images. During rendering, backward warping $\mathbf{w}$ is applied to obtain the ray's corresponding pixel color in each of the four images. The rendered pixel is then computed as a weighted sum from the four light field images.

cope with object occlusions, causing ghosting and other artifacts during rendering.

Warping-Based Approaches

In contrast to flowed light field rendering where for each viewing ray adjacent light field images are locally queried, in warping-based rendering, also known as image morphing or correspondence-based rendering, the acquired light field images are being warped completely using flow fields (Figure 17.6). Warping-based approaches are not restricted to light fields but have a long-standing tradition in view interpolation and the creation of smooth transitions between similar images [Stich et al. 08]. For example, warping has been used to create transitions between different actors who are performing an identical choreography [Beier and Neely 92]. An extension to more than two images has been proposed by Lee et al. [Lee et al. 98]. Chen and Williams [Chen and Williams 93] proposed to use forward image warping for viewpoint interpolation based on previously estimated flow fields. Image warping is applied to each input image, and the final result is obtained by weighted blending of the warped images. The combination of image warping and blending is also frequently referred to as image morphing.

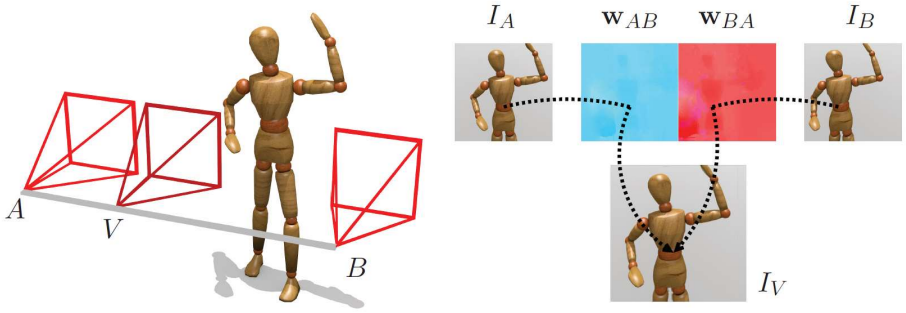


Figure 17.6: Warping-based rendering: to interpolate viewpoint I_V, each pixel in the input images I_A and I_B is forward-warped according to its flow vector $\mathbf{w}_{AB}$, $\mathbf{w}_{BA}$. The final image is weightedly blended from the two warped images.

For real-world images, perceptually convincing dense correspondence fields can be estimated either automatically [Stich et al. 11] or assisted by additional user input [Ruhl et al. 12a]. For multi-view video footage of dynamic scenes, additionally loop consistency among subsequent video frames from neighboring cameras can be exploited [Sellent et al. 12]. For synthetic scenes, the flow vector for a given pixel location $\mathbf{x}$ can be easily derived from per-pixel depth d and camera matrices $\mathbf{P}_A, \mathbf{P}_B$:

$$\mathbf{w}_{AB}(\mathbf{x}) = \mathbf{P}_B(\mathbf{P}_A{}^{-1}(\mathbf{x}, \mathbf{d})) \tag{17.1}$$

where $\mathbf{P}_A{}^{-1}$ is the inverted projection matrix of camera A (Section 1.6). To ensure geometrically undistorted in-between views during warping-based image interpolation, rectifying homography transforms are applied to the input images prior to warping [Seitz and Dyer 96]. Warping enables convincing viewpoint interpolation of panoramas [McMillan and Bishop 95] (Chapter 3), dynamic light fields [Goldlücke et al. 02], uncalibrated images [Fusiello 07] as well as uncalibrated and unsynchronized multi-view video [Lipski et al. 10a].

Similarities exist to flowed light fields as well as to depth-based rendering (Section 17.3). If backward warping is used, i.e., if each target pixel of the view to be synthesized is queried for its location in the captured light field images, image warping is a special case of flowed light fields: here, the interpolated viewpoint is located on the manifold spanned by all camera positions, as opposed to flowed light fields where the target view does not have to lie on the capture manifold. On the other hand, if forward-warping is used, i.e., if each pixel of a captured light field image gets shifted to its new location in the target view, warping is akin to depth-based rendering:

For rectified input imagery, the flow vectors are scanline-aligned and reduce to one-dimensional *disparity* that is proportional to the inverse scene depth at a given pixel position.

Analogous to flowed light fields, warping-based rendering requires considerably fewer input images than pure light field rendering. The amount of image data can be further reduced if cameras do not have to span a 2D manifold but can be arranged in arc-like setups around the scene. Warping is able to compensate for calibration inaccuracies and works with unsynchronized multi-view video footage. It performs robustly even for complex outdoor scenes [Lipski et al. 10a] and lends itself to creating numerous space-time visual effects [Linz et al. 10b].

In comparison to traditional light field rendering, one limitation of warping-based IVBR is that the virtual viewpoint must lie on the manifold spanned by the capture positions of all input images/videos. The quality of the rendered output depends on the accuracy of the dense correspondence maps. For multi-view acquisition setups of up to about 10° between adjacent cameras, robust correspondence estimation algorithms exist [Lipski et al. 12] (Chapter 8). Still, occlusion effects cannot always be handled correctly by warping alone, motivating the use of geometry proxies in IVBR.

17.3 Geometry-Assisted Approaches

Geometry Proxies

Image-based occlusion detection is an active research area [Ince and Konrad 08, Herbst et al. 09]. Alternatively, actual depth information is needed to render occlusion effects robustly [Chen and Williams 93]. Fortunately, even if actual scene geometry is too complex for faithful reconstruction, simple geometry proxies often already suffice to achieve visually convincing rendering results.

The most basic scene geometry approximation consists of a single fronto-parallel plane located in the middle of the scene, often referred to as a billboard (Figure 17.7). By default, the billboard is always oriented perpendicular to the current viewing direction. For rendering, one or more captured scene images are projected onto the billboard and rendered as textures, cross-blending between projected images [Snavely et al. 06, Snavely et al. 08a]. If the scene consists of several objects, separate billboards may be used, one for each object. An individual object may also be represented by more than one billboard. In microfacet billboarding [Yamazaki et al. 02], the object is divided into many thousand small billboards. Further rendering improvements can be achieved by faithfully reconstructing the boundary colors of neighboring proxies [Germann et al. 10] by

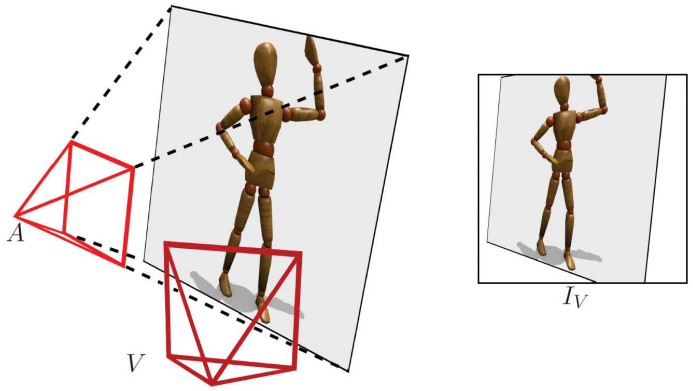


Figure 17.7: In geometry proxy-based rendering, only a coarse geometric representation of the scene may be required. In the depicted case, a single view-dependent, fronto-parallel billboard is used. During rendering, the source image is projected onto the proxy geometry.

merging them in image space [Hornung and Kobbelt 09], or by applying local displacements to billboards [Waschbüsch et al. 07]. The main advantage of billboarding is its modest computational requirements, enabling real-time, on-the-fly rendering from live-captured multi-video footage [Goldlücke and Magnor 03]. Billboard rendering has been successfully applied to free-viewpoint video of actors, sports broadcasts, and architectural scenes [Germann et al. 10, Schwartz et al. 10].

If for some image region no reliable billboard depth can be obtained, to avoid more annoying artifacts such regions may deliberately be visualized in a blurred manner (Section 17.4). In ambient point cloud rendering, for example, random depth values are assigned to pixels of unknown depth so that they form an amorphous, unobtrusive point cloud [Goesele et al. 10].

Geometry proxy-based approaches are able to achieve visually pleasing results without pixel-accurate depth information. Coarse scene representations such as billboards can be estimated very robustly, and computational as well as memory requirements are small due to the limited amount of estimated geometric information. Billboard rendering yields improved rendering results from undersampled light field data. Also, viewpoint position is not restricted to any kind of camera manifold. Still, rendering artifacts may be apparent (Section 17.4). Especially when using a simple billboard proxy, strong ghosting artifacts become visible when cross-blending from widely separated input images. The impact of such artifacts in image-based rendering on perceived image quality has been studied by Vangorp et al. [Vangorp

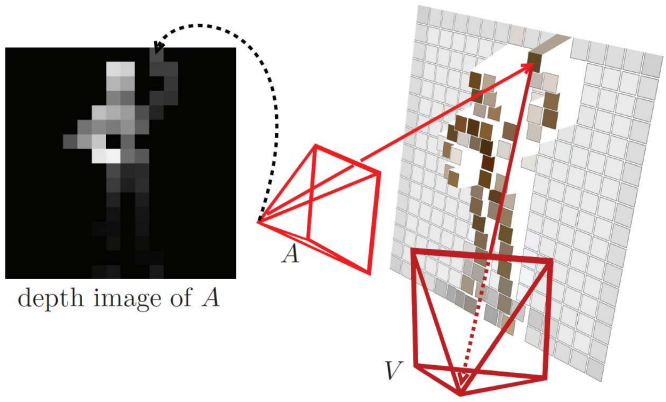


Figure 17.8: In depth-image-based rendering (DIBR), the depth of each pixel is known (visualized by grayscale depth map, left). According to pixel depth and camera matrix of camera image I_A, each pixel is projected to its world space position. Using the projection matrix of the virtual camera I_V, the image I_A is projected to the image plane of I_V.

et al. 11]. Another constraint is that although only coarse depth information is required, the input images must be calibrated.

Depth Image-Based Rendering (DIBR)

Depth image-based rendering relies on dense depth information for every pixel of all input images. To obtain reliable depth, dense per-pixel stereo matching algorithms are typically used (Chapter 8). For rendering, in general each pixel in reference image I_A is reprojected into the world space and re-rendered from the desired viewpoint I_V [Fehn 04] (Figure 17.8). In point cloud rendering [Zabih and Woodfill 94, Addison et al. 95] and point splatting [Hornung and Kobbelt 09], reprojection and re-rendering is accomplished by treating all pixels independently. Alternatively, the source images can be considered as a connected mesh [Zitnick et al. 04, Zheng et al. 09]. To avoid artifacts along object silhouettes, single quads of the mesh that feature large depth discontinuities must be locally discarded. Alpha matting is used to estimate local foreground color and alpha values, and an additional boundary layer is rendered to guarantee smooth transitions between different depth layers.

A very challenging problem is disocclusion handling. Both point splatting and mesh rendering methods may produce holes in the final image [Tauber et al. 07]. One solution is to use two or more source images for rendering that hopefully fill in the holes in the final image [Zitnick et al. 04, Zheng et al. 09]. Another possibility is to use a depth-layered

representation of the scene [Shade et al. 98, Müller et al. 08]. Still, infilled image regions cannot be ruled out and may remain annoyingly visible as empty holes in the synthetized novel view. To remedy the problem, several inpainting techniques have been proposed to assign plausible color information to such unfilled regions [Criminisi et al. 03, Moreno-Noguer et al. 07, Debevec et al. 98]. Common inpainting techniques have also been surveyed and benchmarked for their applicability in image-based rendering [Schmeing and Jiang 11].

If dense and correct depth information can be obtained, depth image-based rendering can be very accurate. Similar to depth proxies, the location of the virtual viewpoint is arbitrary. Also, only few input views are needed to achieve useful rendering results. On the other side, accurate, dense depth reconstruction is notoriously difficult and error-prone for the general case. Acquisition cameras must be calibrated and synchronized accurately, and scene content may not change appearance by too much between different viewpoints, neither due to occlusion effects nor to non-Lambertian reflectance characteristics. In essence, input images may not be separated by more than about 10°, else even state-of-the-art reconstruction methods fail (Chapter 8).

To overcome the problems specific to depth-based and warping-based rendering, a hybrid approach has been proposed [Lipski et al. 14]. By exploiting both dense, pairwise image correspondences as well as depth information simultaneously, convincing rendering results can be obtained even from imprecisely reconstructed depth and inaccurately calibrated, asynchronously captured multi-view footage.

3D Geometry Reconstruction and View-Dependent Texture Mapping

If the scene is not too complex, instead of estimating local per-pixel depth it may be possible to reconstruct a complete, consistent 3D geometry model of the scene prior to rendering (Chapter 12). During rendering, the 3D mesh can then be projectively textured using a view-dependent selection of input images [Debevec et al. 98] (Figure 17.9). If the scene consists of a single object of interest, visual hulls are a common, conservative 3D geometry approximation [Baumgart 74, Potmesil 87, Matusik et al. 00]. While visual hull reconstruction and rendering is very fast and allows for real-time applications [Li et al. 03], due to its inherently limited geometric accuracy attainable rendering quality is limited. Instead, more elaborate, off-line 3D reconstruction schemes can be employed. One approach consists of performing structure-from-motion calibration [Snavely et al. 06] (Chapter 7), prior to applying quasi-dense multi-view reconstruction of surface patches [Furukawa and Ponce 10, Snavely 12, Lipski 12] (Chapter 8). Alternatively,

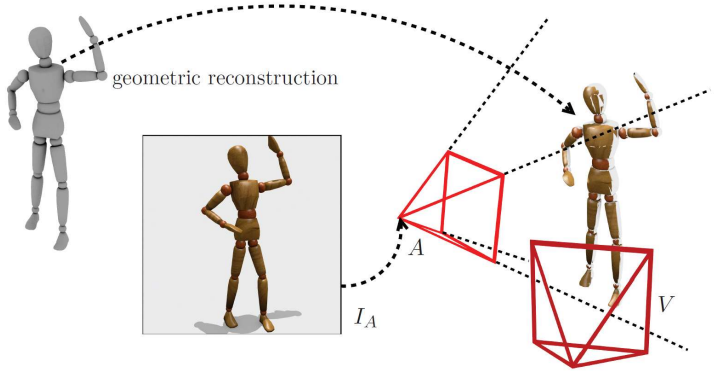


Figure 17.9: View-dependent texture mapping: if an accurate 3D geometry model of the scene is available, it can be projectively textured using only a few captured input images. Typically, more than one input image is used for projective texturing to cover all visible regions of the object. The selection of input images and their respective blending weights are assigned based on the position of the viewpoint.

depth cameras or 3D scanners may be used to obtain partial object geometry [Wood et al. 00] (Chapter 9). Finally, a watertight 3D model is obtained from the acquired point cloud using Poisson surface reconstruction [Kazhdan et al. 06] (Chapter 10). Solutions to estimate 3D geometry exist also for scenes that contain non-Lambertian objects [Vogiatzis et al. 06]. Many reconstruction algorithms rely on silhouette extraction, effectively limiting applicability to scenes consisting of a single, easily segmentable object of interest. If automatic reconstruction is infeasible, scene geometry reconstruction can also be user-guided, e.g., by specifying points and edges of the mesh to determine 3D-positions semi-automatically [van den Hengel et al. 07]. Because of their relevance for popular 3D map services, specific tools have been developed for architectural scenes that allow for user assistance and correction [Debevec et al. 96, google 12]. User-assisted methods are, however, labor-intensive.

For view-dependent texture mapping, only a small number of input views is needed to reproduce highly realistic scene appearance. Given an accurate 3D model, the scene can be rendered from any viewpoint, and the rendered viewpoint is not restricted to any particular area. Having a 3D geometry model available has additional advantages, e.g., it can receive and cast shadows in a virtual scene. On the other hand, exact alignment between projected images and 3D geometry is essential for authentic rendering results. If camera calibration is only slightly off, or if the geometry

Figure 17.10: Model-based IVBR: A priori information about scene content can be exploited to obtain robust modeling results. Instead of reconstructing 3D geometry from scratch, a parameterized 3D model may be fitted to the recorded footage.

model exhibits even small inaccuracies, annoying rendering artifacts occur [Eisemann et al. 08].

Model-Based IVBR

3D geometry can be reconstructed from multi-view imagery based on first principles (Chapter 8). However, scene reconstruction can be considerably improved if knowledge about scene content is exploited and a parameterized 3D model can be provided and a paraterized 3D model can be provided, Figure 17.10. By fitting an a priori 3D model to the recorded data, parameter space is greatly reduced and consistency enforced (Chapter 12). Prominent application areas are human pose estimation and motion capture (Chapter 11). For free-viewpoint video rendering of an actor, for example, joint angles and shape parameters of a 3D human body model as well as time-varying textures may be derived directly from sparse multi-video footage [Carranza et al. 03]. A statistical human body model enables modeling almost any person's physique [Hasler et al. 09a], even from a single video recording [Jain et al. 10]. Alternatively, laser scanning or other depth sensors can be employed to model an individual's 3D geometry precisely [de Aguiar et al. 08b, Ye et al. 11, Kuster et al. 11]. Of course, known 3D geometry and appearance of scene background or other parts of the scene can also be exploited. Many spectator sports, for example, take place on a well-defined playing field. In TV sports broadcasting applications (Chapter 21), this a priori knowledge is used for reliable background segmentation as well as scene augmentation [Hilton et al. 11, Germann et al. 10].

Dynamic Objects and Scenes

Many image-based approaches do not explicitly provide any special means to deal with dynamic scenes recorded with multiple video cameras simultaneously. Although it is always possible to apply image-based techniques independently to each consecutive frame of a multi-video sequence, this is no way to assure temporal coherence. Consequently, independent processing of consecutive time frames can result in flickering artifacts, severely impacting perceived rendering quality. In contrast, video-based rendering approaches are specifically designed for dynamic scene content by both ensuring and exploiting temporal coherence [Magnor 05].

If multi-video acquisition is synchronized across all cameras, an initial geometry model may be estimated for the first frame [Furukawa and Ponce 10] and tracked over successive frames [Furukawa and Ponce 08]. Occlusion of scene parts, however, can lead to holes in the surface model. Mesh completion may be employed to obtain a temporally coherent representation [Li et al. 12a]. For deformable objects like the human face, an initial mesh may be tracked using one or several reconstruction anchor frames [Bradley et al. 10, Beeler et al. 11].

Alternatively, instead of reconstructing a geometry model first and enforcing its temporal consistency in a second step, dynamic geometry may be reconstructed globally as a weighted minimal 3D hyper-surface in 4D space-time [Goldlücke and Magnor 04]. The hyper-surface is defined by the minimum of an energy functional which is given by an integral over the entire hypersurface and which is designed to optimize photo-consistency. A PDE-based evolution derived from the Euler–Lagrange equation maximizes consistency with all of the given video data simultaneously. The result is a globally photo-consistent, closed 3D model of the scene that varies smoothly over time [Goldlücke and Magnor 04].

Besides surface geometry, some IVBR applications also benefit from dense 3D scene motion. This so-called scene flow can be reconstructed from synchronized multi-video footage by estimating the optical flow per camera view and reprojecting the flow fields onto 3D scene geometry [Vedula et al. 05].

While synchronized multi-video recordings considerably simplify subsequent processing, mass-market cameras typically do not provide any technical means for inter-camera synchronization [Hasler et al. 09a]. Even with high-end camera equipment, temporally misaligned frames and complete frame drops can occur [Imre and Hilton 12, Imre et al. 12]. Only a few IVBR approaches explicitly allow for non-synchronized multi-view input imagery. One option is to synchronize dynamic light field recordings prior to rendering via temporal interpolation [Wang et al. 07]. By deliberately offsetting camera recording times, temporal resolution can even be

improved [Li et al. 12b]. For the initial spatial reconstruction, however, still a subset of cameras has to record the dynamic scene in sync.

Besides the technical or practical inability to synchronize multiple video cameras, e.g., in the field, also multi-camera calibration can be a tedious, difficult, and error-prone procedure (Section 2.3). A warping-based approach still enables free-viewpoint video of complex outdoor scenes from completely unsynchronized, uncalibrated, sparse multi-video footage [Lipski et al. 10a]. The underlying idea is to simulate a virtual video camera by interpolating between recorded video frames across space and time. Prior to rendering, dense image correspondences must be estimated between consecutive video frames of each camera sequence as well as between adjacent cameras [Stich et al. 11]. View interpolation takes place in the spatio-temporal domain spanned by all recorded video frames [Stich et al. 08]. By subdividing the interpolation domain into tetrahedrons, with the recorded video frames as vertices and dense correspondence maps along the edges, free viewpoint navigation, slow motion, freeze-and-rotate shots, and many more special effects can be photo-realistically rendered [Linz et al. 10b].

17.4 Advanced Image-Based Methods and Extensions

The classification introduced in the previous sections gives an overview of fundamental IVBR approaches. When looking at some actual rendering systems it is apparent that many of them do not fit precisely into one single category. Some approaches have been proposed that combine different techniques. Unstructured lumigraphs [Buehler et al. 01], for example, generalize both lumigraphs and view-dependent texture maps. Depending on the level of detail of the proxy geometry, they behave like one of the extremes, or a mixture of them. The view-dependent texture mapping approach [Debevec et al. 96] also employs depth-based rendering at a fine level. For each reconstructed facade, the original textures can be projected onto the geometry, and local depth maps are computed to compensate local projection errors. View-dependent textured splatting [Yang et al. 06c], on the other hand, constitutes a mixture between view-dependent texture mapping and point splatting.

For many practical applications it proves to be beneficial to segment the scene into different regions (e.g., actors and background) and to treat them differently. In outdoor sports scenarios, for example, players are separated from the field at an early stage of the processing pipeline [Hilton et al. 11, Germann et al. 10]. While billboard representations or 3D surfaces are reconstructed for the individual players, it is often sufficient to represent the playing field by a single plane. Alternatively, user-supervised segmentation and billboard rendering is used for the foreground person

while the background is reconstructed in high detail and rendered using view-dependent texture mapping [Ballan et al. 10].

Error-Concealed Rendering

As each image-based method has its own advantages, it also has its particular limitations and failure cases. In some scenarios, the user can assist the reconstruction process to obtain pleasing results. Several approaches have been suggested that require user input for geometry reconstruction [Debevec et al. 96, van den Hengel et al. 07]. Other approaches require sparse user input for scene segmentation [Ballan et al. 10, Guillemaut et al. 10] and view interpolation [Chaurasia et al. 11]. Floating textures provide an automatic correction mechanism at render time that does not require any manual intervention [Eisemann et al. 08]. Prior to the blending stage in image-based rendering, the different source image projections on the geometry proxy are locally aligned based on optical flow estimated in real-time. Alternatively, the 3D geometry may be aligned with the images using sparse feature matches, circumventing dense optical flow estimation [Germann et al. 12].

Comprehensive Reconstruction

For free-viewpoint video, i.e., rendering a dynamic, real-world scene from arbitrary vantage points, 3D scene geometry must be available in some form. To allow also for illumination changes of the scene or for augmentation, in addition surface reflectance properties must be estimated, Figure 17.11. If the scene is diffusely reflecting, the reconstruction of one consistent texture map suffices. Different approaches exist to estimate a consistent diffuse texture atlas from multi-view imagery given 3D geometry, illumination, and camera parameters [Wang et al. 01, Lempitsky and Ivanov 07, Gal et al. 10]. 3D geometry and consistent texture may also be estimated simultaneously [Matsuyama et al. 04, Starck and Hilton 07b, Liu et al. 10b, Schwartz et al. 11b, Autodesk 12, Agisoft 12, Hypr3D Development Team 12, Nguyen et al. 12].

In constrast to Lambertian objects, comprehensive reconstruction of scenes containing specular, glossy, semi-transparent, or mirroring surfaces is considerably more difficult [Ihrke et al. 10]. For non-Lambertian surfaces, the ability to recover normal directions accurately varies greatly with both actual surface BRDF and illumination pattern and may even be ill-posed [DŹmura 91]. If 3D scene geometry is known, inverse rendering allows recovering illumination and/or BRDF, represented in spherical harmonics basis functions [Ramamoorthi and Hanrahan 01a]. Alternatively, parameterized reflection models may be fitted to match captured multi-view scene

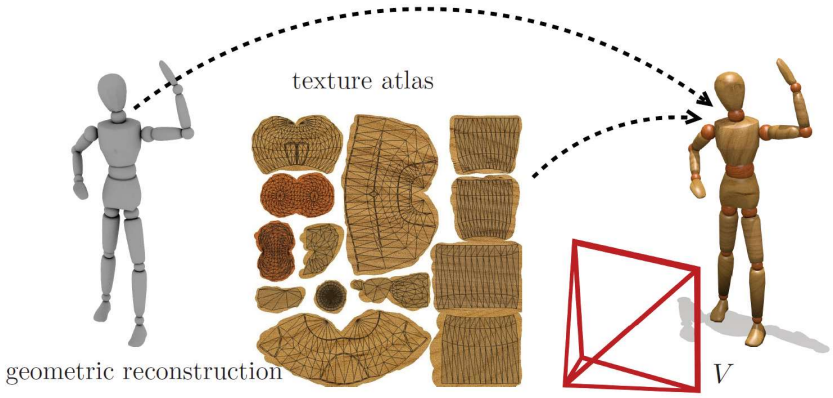


Figure 17.11: Comprehensive reconstruction: if both surface geometry and reflectance properties of the scene can be modeled from the input imagery, the traditional 3D rendering pipeline can be utilized to render the scene from any viewpoint and under arbitrary illumination.

appearance, either for known [Theobalt et al. 07] or unknown [Li et al. 13] scene illumination. If scene appearance from only a single viewpoint is to be varied for different illumination conditions, high-speed recording under time-multiplexed illumination allows relighting the scene [Wenger et al. 05].

Image-Based Object and Scene Manipulation

Image- and video-based rendering approaches concentrate on view interpolation. In recent times, image-based methods have been proposed also for modeling and rendering appearance variations. A set of parameters is used to span a domain to search and warp the images of an object. For articulated objects, for example, such a parameter set can be defined by a skeletal pose representation [Xu et al. 11, Hilsmann et al. 13] (Chapter 11), or by silhouette shape features [Hauswiesner et al. 13]. For facial expressions, facial feature locations in the image can be used [Zhang et al. 06]. During synthesis, a temporal coherent matching strategy is used to identify the *nearest* database image(s) to the given configuration of pose or facial expression in descriptor space. The retrieved images are then used to synthesize an image for this pose configuration. The best matching database image is mapped onto an animated 3D model of a human. As the retrieved image might not show exactly the same pose as required, fine-scale warping of the rendered image is necessary. Other approaches synthesize articulated human body poses by a convex combination of example poses [Casas et al. 14].

The interpolation domain for skeletal poses or facial expressions is much more complex and of higher dimensionality than the interpolation domain

necessary for view synthesis. Possible expressions/poses that can be synthesized from the database are restricted to the convex hull of examples, and sampling the space such that every possible expression/pose can be synthesized becomes intractable. This limitation has been addressed by splitting up the object, i.e., the face or human body, into subregions that are assumed to be more or less independent. Each of these regions then has its own descriptor space and is synthesized from different example images. To produce the final image, the subregion images are seamlessly blended.

Compositing, Augmentation, and Consolidation with Traditional Computer Graphics

The paramount goal of image- and video-based rendering techniques is to capture real environments and synthesize novel views. However, if further interaction with or editing of the scene is required several additional problems occur, many of them still unsolved. One challenge is realistic compositing of image- and video-based rendering results with those from traditional 3D rendering approaches. It is as apparent that there are many applications where real content is to be complemented with synthetic assets, or the acquired 3D data is (partly) used in otherwise synthetic scenes. For example, few blockbuster movies nowadays are not augmented with 3D renderings in the form of special effects (Chapter 20). Also live TV sports broadcasts and other application areas rely on convincingly augmenting real-world footage with synthetic 3D-rendered content (Chapters 21, 23).

Arguably the simplest solution to combining image- or video-based rendering with traditional 3D rendering is to 3D-reconstruct the scene from the recorded video footage (Part II), augment the scene in 3D world space, and 3D-render it again. However, image-based 3D reconstruction methods have numerous limitations. A reconstruction method may be applicable only to single objects and may require a controlled environment for capture, or recorded scene appearance is not factorized into lighting and reflectance preventing photo-realistic augmentation with other 3D models, or the scene is optically too complex for faithful reconstruction. These restrictions give rise to several interesting research challenges toward realistic augmentation of real-world scenes with synthetic 3D graphics. Among the different aspects that need to be considered are correctly matching the illumination of the virtual object to that of the real-world scene as well as color bleeding and shadows cast by the virtual object onto the real-world scene, and vice versa. Fortunately, the task of augmenting a complex, dynamic real-world scene with virtual objects can, in essence, be reduced to consistently augmenting each video frame separately.

Illumination Reconstruction The appearance of a synthetic object is determined by its 3D shape, surface reflectance characteristics, and illumination. If the goal is to augment some virtual object into an image, its 3D shape and reflectance properties are known. Only the lighting conditions of the scene in the image are unknown and must be reconstructed. One simplification that is often made is that only the far-field illumination of the scene is reconstructed, i.e., the scene is assumed to be illuminated by a hemisphere infinitely far away so the illuminating light distribution can be represented as an environment map. Scene illumination can be acquired directly by placing a so-called light probe, often a mirroring sphere, into the real-world scene and taking a photo of it [Debevec 98]. Alternatively, a fish-eye lens can be used [Sato et al. 99]. If the recorded scene is not accessible anymore, illumination may be interactively estimated using some (coarse) scene geometry proxy [Karsch et al. 11]. In many cases, illumination estimation is tightly coupled with the reconstruction of overall scene appearance in a joint optimization approach [Kholgade et al. 14, Rogge et al. 14, Hara et al. 08, Haber et al. 09]. Regularly, surface reflectance of the real-world scene is assumed to be Lambertian [Ramamoorthi and Hanrahan 01b]. Illumination and reflectance reconstruction from photos and videos remains to be an active research area in computer graphics and computer vision.

Realistic Rendering In order to achieve consistent overall appearance when compositing virtual objects into real-world footage, not only direct scene illumination must be known but also inter-object light transport between all objects in the scene has to be computed. Inter-object light transport gives rise not only to cast shadows but also to such visually important yet subtle effects as indirect illumination, color bleeding, and caustics. In essence, taking these effects into account amounts to computing the rendering equation for the entire augmented 3D scene (Chapter 6).

Although stunningly realistic images can be rendered interactively (e.g., in video games), the light transport in these scenes is often approximated or based on simplifying assumptions [Ritschel et al. 12]. Rendering photorealistic images offline (i.e., without tight computational budgets but with significantly higher quality demands) is nowadays almost exclusively done using (Markov Chain) Monte Carlo methods. These methods share the concept of stochastically constructing paths that connect the sensor of a virtual camera to a virtual light source and computing the energy reaching the sensor's pixels. This process can be done in many different ways: sampling only from the sensor or the light sources (path tracing [Kajiya 86] or light tracing [Arvo 86]), sampling from both sides with deterministic connections (bidirectional path tracing [Lafortune and Willems 93, Veach

and Guibas 94]), mutating paths with Metropolis light transport [Veach and Guibas 97], or density estimation of path vertices (photon mapping [Jensen 96]). Going beyond pure light transport, additional realism can be achieved by simulating the effects of actual cameras such as depth-of-field [Lee et al. 10b, Kán 12], lens flares [Hullin et al. 11], or accurate simulation of lens models [Hanika and Dachsbacher 14].

Global illumination has seen tremendous progress in the last decades [Pharr and Humphreys 10, Křivánek et al. 13, Dachsbacher et al. 13]. Nevertheless, not all techniques are equally well suited for all scene settings which can require specifically tailored solutions [Veach and Guibas 97, Jakob and Marschner 12, Kaplanyan et al. 14, Hachisuka et al. 08, Kaplanyan and Dachsbacher 13a, Kaplanyan and Dachsbacher 13b, Georgiev et al. 12, Hachisuka et al. 12]. The demands on the rendering methods increase with the richness of detail, accuracy, and the spectrum of materials.

Compositing The most successful algorithm to cope with the insertion of virtual objects into a given scene is presumably the differential rendering technique first proposed in [Fournier et al. 93] and made popular by [Debevec 98]. The idea behind differential rendering is to compute the light interaction between the scene and the virtual object, i.e., the near-field illumination (for example, shadows cast on the ground), by making use of a coarse (hand-made) representation of the scene surrounding the virtual object. The scene is rendered once with and once without the virtual object to be augmented, and the difference is applied to the original input image. Given an input image I_{bg}, the (potentially manually created) 3D scene geometry proxy is global illumination-rendered from the same viewpoint as I_{bg}, once without the virtual object and once with the object inserted, resulting in images I_{noobj} and I_{obj}, respectively. Additionally, an object matte α is computed to mark pixels depicting the virtual object as 1 and all remaining pixels as 0. The final composite is then computed via

$$I_{\mathrm{final}} = \alpha \cdot I_{\mathrm{obj}} + (\mathbf{1} - \alpha) \cdot (I_{\mathrm{bg}} + (I_{\mathrm{obj}} - I_{\mathrm{noobj}})) \ .$$

In this form, it becomes clear that for each object the pixel value is simply copied from the rendered image I_{obj}. For the remaining pixels, if the virtual object does not affect its surrounding, I_{obj} and I_{noobj} are equal and the result is equal to I_{bg}. If I_{obj} is darker than I_{noobj} light is subtracted from the input photograph, introducing shadowed regions. On the other hand, if I_{obj} is brighter than I_{noobj} intensity is added signifying, for example, caustics. Several improvements of this technique have been proposed, e.g., for moving objects [Drettakis et al. 97], using final gathering [Loscos et al. 99], making use of differential photon mapping for refractions [Grosch 05] or taking near-field illumination into account [Grosch et al. 07].

17.5 Summary

Image- and video-based rendering can be categorized into purely image-based approaches, which directly synthesize new images by re-sampling and interpolating the captured data, and geometry-assisted approaches that exploit reconstructed depth information or higher-level models of the scene to guide the image synthesis process. Beyond pure view interpolation of static scenes, approaches for dynamic scenes and objects allow synthesizing new images in a space-time continuum. The categorization into purely image-based and geometry-assisted approaches is not to be understood as a fixed classification but rather aims at giving an overview on existing methods. Many approaches do not fit exactly into one single category but can be located somewhere in between. Combining image- and video-based rendering with traditional 3D rendering is an active field of research. The augmentation of real world-acquired scenes with virtually created content frequently requires specifically tailored methods and solutions.

18
Stereo 3D and Viewing Experience

Kai Ruhl

18.1 Introduction

In the last decade, technological advances in stereo 3D (S3D) have attracted renewed interest of both academia and industry, including the first commercially viable 3D content market for binocular video. Previous attempts at producing S3D content go back as far as the 19th century, using black and white photography providing one image per eye on static glasses. The 20th century saw first attempts toward S3D movies for a mass audience, but technological drawbacks had hindered its acceptance.

Today, S3D is well established particularly in the movie industry. Digital distribution and projection systems coupled with inexpensive polarization glasses that ameliorate user comfort issues have been crucial aspects. Equally important, content creators have gained more experience in producing high-quality S3D footage that enhances storytelling instead of relying on catchy effects. Toolmakers are able to effectively use the computational power of standard computer electronics to provide S3D content production and handling capabilities to a wider variety of movie production companies, while digital S3D camera systems provide footage that facilitates efficient post-production. In general, the entire production and delivery pipeline has matured to an unprecedented level, opening up many new research avenues (Figure 18.1).

S3D for gaming is another emerging mass market. However, the viewing experience has not yet reached sufficient acceptance levels as decreased

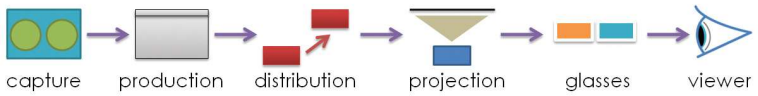


Figure 18.1: For 3D viewing, most of the traditional movie content production and presentation pipeline must be upgraded.

Table 18.1: Both monocular and binocular visual perception use 3D visual cues to construct a spatial model of a scene.

Monocular 3D Cues	Binocular 3D Cues
Size consistency	Convergence
Perspective consistency	Disparity
Motion parallax	
Focus	

willingness of users to wear glasses, a wider viewing angle distribution, and less isolated surroundings increase technological difficulties. This applies even more to 3D on mobile devices and to head-mounted displays, which only recently reached good levels of maturity.

This chapter gives an overview of contemporary S3D production and presentation, and goes into detail on 3D movies. Section 18.2 describes 3D cues that are used by the human visual system (HVS) to synthesize a 3D impression. Subsequently, current 3D viewing hardware and the way it produces these 3D cues are outlined in Section 18.3. Section 18.4 is devoted to the currently predominant stereoscopic S3D displays and their inherent challenges and limitations. Section 18.5 gives an overview of 3D video representation possibilities, both synthetic and from real-world data. Section 18.6 focuses on S3D post-production and depth estimation/correction from real-world data. Finally (Section 18.7) describes delivery of 3D content.

18.2 Stereo Perception and the Human Visual System

The human visual system (HVS) uses both monocular and binocular cues to create a natural 3D impression. Since the limited inter-ocular baseline makes stereo perception viable only for short to medium distances, monocular cues are at least as important as binocular ones and have been used by film and other media producers since the beginning of photography. Binocular cues are unavailable for a minority of the population (<10%) due to disorders known as phoria and tropia [Read and Bohr 14].

Monocular cues of a static view start with perspective and size consistency and the relative ordering of shapes. Adding observer or object movement, motion parallax is a strong cue, especially for objects at longer distances. With the eyes being finite-sized aperture instead of pinhole cameras, accommodation of the eye for sharpness is the another cue: If one eye focuses on an object of attention, only objects on that focal plane remain

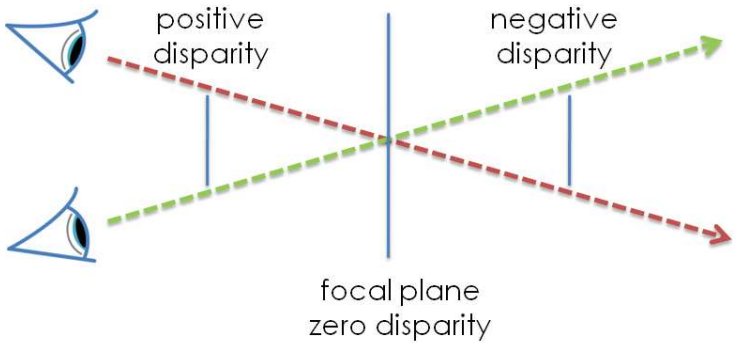


Figure 18.2: Positive, zero, and negative disparity form the stereoscopic cue of binocular vision that is based on object distance.

sharp, while objects in front or behind that plane are blurred. 2D film makers and photographers have long used this effect (called "depth-of-field") to guide the viewers' attention and produce an impression of depth, to the point that viewers have come to expect some amount of depth-of-field from professional footage.

Binocular cues use triangulation from both eyes to combine visual information and synthesize 3D structure from object relations, using only the fovea centralis, a small area on the retina with the highest receptor density and color sensitivity. Convergence is the cue that evaluates the angle between the eyes necessary to project the point of interest onto the fovea of each eye, and is the dominant cue for shorter distances. Retinal disparity is the second binocular cue that analyzes the differences between images projected into the left and right eye. Objects on the focal plane will produce the same image in both eyes; objects in front of the plane have positive disparity, and objects behind the plane have negative disparity, as shown in Figure 18.2. However, retinal disparity only works in a relatively limited depth range around the focal plane, known as Panum's area [Fender and Julesz 67]. Outside of that limited range of disparity fusion, the HVS is receiving double images (diplopia).

In a real-world setting, monocular and binocular cues are always consistent, providing a concerted viewing experience since there are no contradictions between cues extracted from the scene. The brain's cognitive processes weight and combine all this information in a meaningful way. For example, the influence of convergence is low for far-away objects, when the eye rays are almost parallel; while relative shifts from motion parallax would be weighted higher in this case.

Only a tiny portion of a natural scene is actually perceived clearly (i.e., on the fovea and in focus) at each time instance. As viewers look around any given scene, they combine different views of the same scene to assemble a mental model, all the while checking against their experience of the real world. Display systems which are not true 3D (unlike, e.g., holographic displays) have difficulties providing this experience.

Given a person's propensity to look around, content providers for display systems that cannot provide all cues in a consistent manner must steer viewer attention to keep it within focus and convergence (see also Section 18.4). An environment like a movie theater, where viewers are willing to remain seated in a dark room with mostly undivided attention toward a screen filling most of their field-of-view, is very conducive for content creators, while living-room (TV, gaming) or on-the-road scenarios (mobile devices) present additional challenges.

18.3 3D Displays

An ideal 3D display would replicate all monocular and binocular visual cues to provide the exact same viewing experience as if watching the original natural scene—essentially a holodeck whose extent covers line-of-sight. Given that molecular replication is infeasible with today's technology, other substitutes must be found.

Currently, holographic and volumetric displays come closest to providing "true" 3D, in the sense that the pixel or voxel location is physically where the natural scene would be [Yaraş et al. 10, Grossman and Balakrishnan 06]. However they are still in an early technological stage, featuring both relatively low resolution, low frame-rate and limited alpha definition (i.e., the ability to adjust solidity/translucency). Content creation for this type of display has also not yet been explored thoroughly, nor has data delivery where each frame must transport the information necessary for an entire volume.

A more easily exploitable and hence more developed technique is the use of two separate 2D displays, one for each eye. Head mounted displays (HMD), shutter glasses, polarized glasses, and autostereoscopic displays fall into this category.

HMDs are the most stable of these solutions in the respect that they eliminate crosstalk (i.e., the "bleeding" of content destined for one eye into the other) completely. However the displayed image must align to the (micro-) head movements of the wearer to avoid nausea caused by visual/inner ear balance inconsistencies, an effect prevalent in previous generation devices. As the displays are located in close proximity to the eyes, they must feature high resolution in a small panel. Recently, display panels have

reached a stage where the resolution comes close to being practically applicable, and advances in real-time tracking have achieved a sufficient amount of visual/balance consistency, as demonstrated e.g., by the Oculus Rift.[1]

In scenarios where wearing a comparatively heavy HMD is impractical, shutter glasses coupled with a 2D display synchronized to them is a feasible solution that has had some commercial products, e.g., Nvidia 3D Vision,[2] mostly for small-audience applications since the glasses are still comparatively expensive. For mass audiences, passive systems such as polarized glasses are an inexpensive, practical possibility. The latest generation used in today's cinemas feature circular polarization, which allows viewers to tilt their heads to some degree without causing adverse effects. Compared to shutter glasses, today's polarized glasses still cause some crosstalk (typically less than 10% [Wang et al. 11b]).

Another emerging technology is autostereoscopic displays which provide multiple viewpoints (based on viewing angle) without requiring glasses [Urey et al. 11]. Moving around the center of a scene is possible to some degree, enabling motion parallax cues. On the other side, objects at infinity are not possible from all viewing angles since the scene needs to be centered somewhere. This requires major capture style adjustments from producers. Furthermore, the alignment of viewing angle slices to the viewer's eye is currently not entirely satisfactory and viewer position is crucial. Smaller autostereoscopic displays have been used in commercial mobile devices like phones (e.g., LG Optimus 3D, HTC Evo 3D) and gaming consoles (e.g., Nintendo 3DS), but again alignment issues make use cumbersome, i.e., the viewer has to hold the device in a certain way for best visual results.

Viewer acceptance of a certain display technology also depends on the setting. Within a movie theater, acceptance of glasses is higher than in an often more social scenario involving a TV in the living room. At the other end of the spectrum, acceptance of glasses is very low for mobile devices that are frequently operated in public.

18.4 3D Perception on 2D Displays

Currently, one of the predominant forms of 3D production and presentation is centered around S3D movies inside a theater, using polarized glasses. Creatives have understood how to design a compelling viewing experience while minimizing eye fatigue, focusing on expressing depth artistically to support the narrative and storytelling. This requires an understanding of the limitations of the 3D experience from a 2D screen plane.

[1] Oculus Rift – http://www.oculusvr.com/

[2] Nvidia 3D Vision – http://www.nvidia.com/object/3d-vision-main.html