Markerless 3D Face Tracking

Walder, Christian; Breidt, Martin; Bulthoff, Heinrich; Scholkopf, Bernhard; Curio, Cristobal

Published in:
Pattern Recognition

Link to article, DOI:
10.1007/978-3-642-03798-6

Publication date:
2009

Markerless 3D Face Tracking

Christian Walder[1,2], Martin Breidt[1], Heinrich Bülthoff[1], Bernhard Schölkopf[1], and Cristóbal Curio[1]

[1] Max Planck Institute for Biological Cybernetics, Tübingen, Germany.
[2] Informatics and Mathematical Modelling, Technical University of Denmark.

Abstract. We present a novel algorithm for the markerless tracking of deforming surfaces such as faces. We acquire a sequence of 3D scans along with color images at 40Hz. The data is then represented by implicit surface and color functions, using a novel partition-of-unity type method of efficiently combining local regressors using nearest neighbor searches. Both these functions act on the 4D space of 3D plus time, and use temporal information to handle the noise in individual scans. After interactive registration of a template mesh to the first frame, it is then automatically deformed to track the scanned surface, using the variation of both shape and color as features in a dynamic energy minimization problem. Our prototype system yields high-quality animated 3D models in correspondence, at a rate of approximately twenty seconds per timestep. Tracking results for faces and other objects are presented.

1 Introduction

Creating animated 3D models of faces is an important and difficult task in computer graphics due to the sensitivity of the human perception of face motion. People can detect slight peculiarities present in an artificially animated face model, which makes the animator's job rather difficult and has lead to *data-driven* animation techniques, which aim to capture live performance. Data-driven face animation has enjoyed increasing success in the movie industry, mainly using marker-based methods. Although steady progress has been made, there are certain limitations involved in placing physical markers on a subject's face. Summarizing the face by a sparse set of locations loses information, and necessitates motion re-targeting to map the marker motion onto that of a model suitable for animation. Markers also occlude the face, obscuring expression wrinkles and color changes. Practically, significant time and effort is required to accurately place markers, especially with brief scans of a numerous subjects — a scenario common in the computer game industry.

Tracking without markers is more difficult. To date, many markerless tracking methods have made extensive use of optical flow calculations between adjacent time-steps of the sequence. Since local flow calculations are noisy and inconsistent, it is necessary to introduce spatial coherency constraints. Although significant progress has been made in this direction [1], the sequential use of between-frame flow vectors can lead to continual accumulation of errors, which

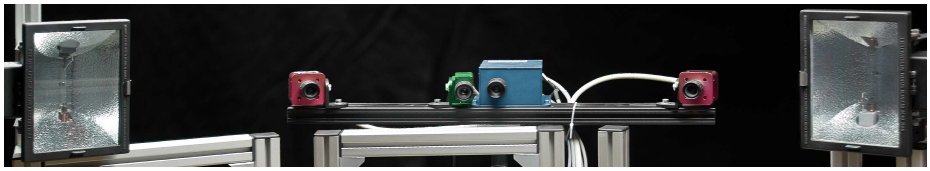

Fig. 1. Setup of the dynamic 3D scanner. Two 640 by 480 pixel *photon focus MV-D752-160* gray-scale cameras (red) compute depth images at 40 Hz from coded light projected by the synchronized *minirot H1* projector (blue). Two strobes (far left and right) are triggered by the 656 by 490 pixel *Basler A601fc* color camera (green), capturing color images at a rate of 40 Hz.

may eventually necessitate labor intensive manual corrections [2]. It is also noteworthy that facial cosmetics designed to remove skin blemishes strike directly at the key assumptions of optical flow-based methods. Non flow-based methods include [3]. There, local geometrical patches are modelled and stitched together. [4] introduced a multiresolution approach which iteratively solves between-frame correspondence problems using feature points and 3D implicit surface models. Neither of these works use color information. For face tracking purposes, there is significant redundancy between the geometry and color information. Our goal is to exploit this multitude of information sources, in order to obtain high quality tracking results in spite of possible ambiguities in any of the individual sources. In contrast to classical motion capture we aim to capture the surface densely rather than at a sparse set of locations.

We present a novel surface tracking algorithm which addresses these issues. The input to the algorithm consists of an unorganized set of four-dimensional (3D plus time) surface points, along with a corresponding set of surface normals and surface colors. From this data, we construct a 4D implicit surface model, as well as a regressed function which models the color at any given point in space and time. Our 4D implicit surface model is a partition of unity type method similar to [5], but uses a local weighting scheme which is particularly easy to implement efficiently using a nearest neighbor library. By requiring only an unorganized point cloud, we are not restricted to scanners which produce a sequence of 3D frames, and can handle samples at arbitrary points in time and space as produced by a laser scanner, for example.

2 Surface Tracking

In this section we present our novel method of deforming the initial template mesh to move in correspondence with the scanned surface. The dynamic 3D scanner we use is a commercial prototype (see Figure 1) developed by ABW GmbH (**http://www.abw-3d.de**) and uses a modified coded light approach with phase unwrapping. A typical frame of output consists of around 40K points with texture coordinates that index into the corresponding color texture image.

Input. The data produced by our scanner consists of a sequence of 3D meshes with texture images, sampled at a constant rate. As a first step we transform each

mesh into a set of points and normals, where the points are the mesh vertices and the corresponding normals are computed by a weighted average of the adjacent face normals, using the method described in [6]. Furthermore, we append to each 3D point the time at which it was sampled, yielding a 4D spatio-temporal point cloud. To simplify the subsequent notation, we also append to each 3D surface normal a fourth temporal component of value zero. To represent the color information, we assign to each surface point a 3D color vector representing the RGB color, which we obtain by projecting the mesh produced by the scanner into the texture image. Hence we summarize the data from the scanner as the set of m (point, normal, color) triplets $\{(\mathbf{x}_i, \mathbf{n}_i, \mathbf{c}_i)\}_{i<1<m} \subset \mathbb{R}^4 \times \mathbb{R}^4 \times \mathbb{R}^3$.

Template mesh. In addition to the above data, we also require a template mesh in correspondence with the first frame produced by the scanner, which we denote by $M_1 = (V_1, G)$, where $V_1 \in \mathbb{R}^{3\times n}$ are the n vertices and $G \subset J \times J$ the edges where $J = \{1, 2, \ldots, n\}$. The construction of the template mesh could be automated — for example we could *(a)* take the first frame itself (or some adaptive refinement of it, for example as produced by a marching cubes type of algorithm such as [7]), or *(b)* automatically register a custom mesh as was done in a similar context in *e.g.* [1]. Instead we opt for an interactive approach, using the *CySlice* software package — this semi-automated step requires approximately 15 minutes of user interaction and is guaranteed to lead to a high quality initial registration (see Figure 3, top left). We normally use a template mesh of 2100 vertices, but this is not an algorithmic restriction, higher res meshes are demonstrated in the accompanying video (see footnote 1, page 7).

Output. The aim is to compute the vertex locations of the template mesh for each frame $i = 2, \ldots s$, such that it moves in correspondence with the observed surface. We denote the vertex locations of the i-th frame by $V_i \in \mathbb{R}^{3\times n}$. Throughout the paper we refer to the j-th vertex of V_i as $\mathbf{v}_{i,j}$. We also use $\tilde{\mathbf{v}}_{i,j} \in \mathbb{R}^4$ to represent $\mathbf{v}_{i,j}$ concatenated with the relative time of the i-th frame. That is, $\tilde{\mathbf{v}}_{i,j} = (\mathbf{v}_{i,j}^\top, \Delta i)^\top$ where Δ is the interval between frames.

2.1 Algorithm

We take the widespread approach of minimizing an energy functional, $E_{\text{obj.}}$, which in our case is defined in terms of the entire sequence of vertex locations, $V_1, V_2, \ldots, V_s$. Rather than using the (point, normal, color) triplets directly, we instead use summarized versions of the geometry and color, as represented by the implicit surface embedding function $f_{\text{imp.}}$ and color function $f_{\text{col.}}$, respectively. The construction of these functions is explained in detail in Appendix A. For now it is sufficient to know that the functions can be setup and evaluated rather efficiently, are differentiable almost everywhere, and

1. $f_{\text{imp.}} : \mathbb{R}^4 \to \mathbb{R}$ estimates the signed distance to the scanned surface given the spatio-temporal location (say, $\mathbf{x} = (x, y, z, t)^\top$). The signed distance to a surface $\mathcal{S}$ evaluated at $\mathbf{x}$ has absolute value $|\text{dist}(\mathcal{S}, \mathbf{x})|$, and a sign which differs on different sides of $\mathcal{S}$. At any fixed t, the 4D implicit surface can be thought of as a 3D implicit surface in (x, y, z) (see Figure 2, left).

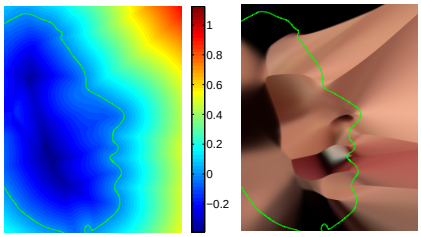

Fig. 2. The nearest neighbor implicit surface (left, intensity plot of $f_{\text{imp.}}$) and color (right, RGB plot of $f_{\text{col.}}$) models. Time and one space dimension are fixed, plotting over the two remaining space dimensions. Shown is a vertical slice through the data of a human face, revealing the profile contour with the nose pointing to the right. For reference, the zero level set of the implicit appears in both images as a green curve.

2. $f_{\text{col.}} : \mathbb{R}^4 \to \mathbb{R}^3$ similarly estimates a 3-vector of RGB values. Evaluated away from the surface, the function returns an estimate of the color of the surface nearest to the evaluation point (see Figure 2, right).

Modelling the geometry and color in this way has the practical advantage that as we construct $f_{\text{imp.}}$ and $f_{\text{col.}}$ we may separately adjust parameters which pertain to the noise level in the raw data, and then visually verify the result. Thereafter we approach the tracking problem under the assumption that $f_{\text{imp.}}$ and $f_{\text{col.}}$ contain little noise, while summarizing the relevant information in the raw data.

The energy we minimize depends on the vertex locations through time and connectivity (edge list) of the template mesh, the implicit surface model, and the color model, *i.e.*, $V_1, \ldots V_s, G, f_{\text{imp.}},$ and $f_{\text{col.}}$. With a slight abuse of notation, the functional is $E_{\text{obj.}} \equiv \sum_{l \in \text{terms}} \alpha_l E_l$, where the α_l are parameters which we fix as described in Section 2.2, and the E_l are the individual terms which we now introduce. Note that it is possible to interpret the minimizer of the above energy functional as the *maximum a posteriori* estimate of a posterior likelihood in which the individual terms $\alpha_l E_l$ are interpreted as negative log-probabilities.

Distance to the Surface. The first term is straightforward — in order to keep the mesh close to the surface, we approximate the integral over the template mesh of the squared distance to the scanned surface. As an approximation to this squared distance we take the squared value of the implicit surface embedding function $f_{\text{imp.}}$. We approximate the integral by taking an area weighted sum over the vertices. The quantity we minimize is given by $E_{\text{imp.}} \equiv \sum_i \sum_j a_j f_{\text{imp.}}(\tilde{\mathbf{v}}_{i,j})^2$. Here, as throughout the paper, a_j refers to the Voronoi area [6] of the j-th vertex of M_1, the template mesh at its starting position, but we state the simpler form here as it is easier to implement, and is more numerically stable.

Color. We assume that each vertex should remain on a region of stable color, and accordingly we minimize the sum over the vertices of the *sample variance* of the color components observed at the sampling times of the dynamic 3D scanner. We discuss the validity of this assumption in Section 4. The sample variance of a vector of observations $\mathbf{y} = (y_1, y_2, \ldots, y_s)^\top$ is $V(\mathbf{y}) \equiv \sum_{i=1}^{s} \left(y_i - \sum_{i'=1}^{s} y_{i'}/s\right)^2 / s$. To ensure a scaling which is compatible with that of $E_{\text{imp.}}$, we neglect the term $1/s$ in the above expression. Summing these variances over RGB channels, and taking the same approximate integral as before, we obtain $E_{\text{col.}} \equiv \sum_{i,j} a_j \| f_{\text{col.}}(\tilde{\mathbf{v}}_{i,j}) - \sum_{i'} f_{\text{col.}}(\tilde{\mathbf{v}}_{i',j})/s \|^2$.

Acceleration. To obtain smooth motion we also minimize a similar approximation to the surface integral of the squared acceleration of the mesh. For a

physical analogy, this is similar to minimizing a discretization in time and space of the integral of the squared accelerating forces acting on the mesh, assuming that it is perfectly flexible and has constant mass per area. The corresponding term is given by $E_{\text{acc.}} \equiv \sum_j a_j \sum_{i=2}^{s-1} \|\mathbf{v}_{i-1,j} - 2\mathbf{v}_{i,j} + \mathbf{v}_{i+1,j}\|^2$.

Mesh Regularisation. In addition to the previous terms, it is also necessary to *regularize* deformations of the template mesh, in order to prevent unwanted distortions during the tracking phase. Typically such regularization is done by minimizing measures of the amount of *bending* and *stretching* of the mesh. In our case however, since we are constraining the mesh to lie on the surface defined by $f_{\text{imp.}}$, which itself bends only as much as the scanned surface, we only need to control the stretching of the template mesh.

Due to space constraints we now only briefly motivate our choice of regulariser. It is possible to use variational measures of mesh deformations, but we found these energies inappropriate in our experiments as it was difficult to choose the correct amount by which to penalize the terms — either: *(1)* the penalization was insufficient to prevent undesirable stretching of the mesh in regions of low deformation, or *(2)* the penalization was too great to allow the correct deformation in regions of high deformation. It is more effective to penalize an adaptive measure of stretch, which measures the amount of local distortion of the mesh, while retaining invariance to the absolute amount of stretch. To this end, we compute the *ratio* of the area of adjacent triangles, and penalize the deviation of this ratio, from that of the initial template mesh M_1, *i.e.*

$$E_{\text{reg.}} \equiv \sum_{i=2}^{s} \sum_{\mathbf{e} \in G} a(\mathbf{e}) \left(\frac{\text{area}(\text{face}_1(\mathbf{e}_i))}{\text{area}(\text{face}_2(\mathbf{e}_i))} - \frac{\text{area}(\text{face}_1(\mathbf{e}_1))}{\text{area}(\text{face}_2(\mathbf{e}_1))} \right)^2 .$$

Here, $\text{face}_1(\mathbf{e})$ and $\text{face}_2(\mathbf{e})$ are the two triangles containing edge $\mathbf{e}$, $\text{area}(\cdot)$ is the area of the triangle, and $a(\mathbf{e}) = \text{area}(\text{face}_1(\mathbf{e}_1)) + \text{area}(\text{face}_2(\mathbf{e}_1))$. Note that the ordering of face_1 and face_2 affects the above term. In practice we restore invariance with respect to this ordering by augmenting the above energy with an identical term with reversed order.

2.2 Implementation

Deformation Based Re-parameterization. Optimising with respect to the $3(s-1)n$ variables corresponding to the n 3D vertex locations of frames $2, 3, \ldots, s$ has the following critical shortcomings: **1)** It necessitates further regularisation terms to prevent folding and clustering of the mesh, for example. **2)** The number of variables is rather large. **3)** Compounding the previous shortcoming, convergence will be slow, as this direct parameterization is guaranteed to be ill-conditioned. This is because, for example, the regularisation term $E_{\text{reg.}}$ acts in a sparse manner between individual vertices. Hence, loosely speaking, gradients in the objective function due to local information (for example due to the color term $E_{\text{col.}}$) will be propagated by the regularisation term in a slow domino-like manner from one vertex to the next only after each subsequent step in the optimization. A simple way of overcoming these shortcomings is to optimize with

respect to a lower dimensional parameterization of plausible meshes. To do this we manually select a set of control vertices that are displaced in order to deform the template mesh. To this end, we take advantage of some ideas from interactive mesh deformation [8]. This leads to a linear parameterization of the vertex locations $V_2, V_3, \dots V_s$, namely $\overline{V}_i = V_1 + P_i B$, where $P_i \in \mathbb{R}^{3\times p}$ represent the free parameters and $B \in \mathbb{R}^{p\times n}$ represent the basis vectors derived from the deformation scheme [9]. We have written $\overline{V}_i$ instead of V_i, as we apply another parameterized transformation, namely the rigid body transformation. This is necessary since the surfaces we wish to track are not only deformed versions of the template, but also undergo rigid body motion. Our vertex parameterization hence takes the form $V_i = R(\theta_i)\overline{V}_i + \mathbf{r}_i = R(\theta_i)(V_1 + P_i B) + \mathbf{r}_i$, where $\mathbf{r} \in \mathbb{R}^3$ allows an arbitrary translation, $\theta_i = \left(\alpha_i, \beta_i, \gamma_i\right)^\top$ is a vector of angles, and $R(\theta) \in \mathbb{R}^{3\times 3}$ is a rotation matrix.

Remarks on the re-parameterization. The above scheme does not amount to tracking only the control vertices. Rather, the objective function covers all vertices, and the control vertices are optimized to minimize this global error. Alternatively one could optimize all vertex positions in an unconstrained manner. The main drawback of doing so however is not the greatly increased computation times, but the fact that allowing each vertex to move freely necessitates numerous additional regularisation terms in order to prevent undesirable mesh behaviors such as triangle flipping. While such regularisation terms may succeed in solving this problem, the above re-parameterization is a more elegant solution, as we found the problem of choosing various additional regularisation parameters to be more difficult in practice than the problem of choosing a set of control vertices that is sufficient to capture the motion of interest. Note that the precise placement of these control vertices is not critical, provided they afford sufficiently many degrees of freedom. Hence, the computational advantages of our scheme are a fortunate side effect of the regulariser induced by the re-parameterization.

Incremental Optimization. It turns out that even in this lower dimensional space of parameters, optimizing the entire sequence at once in this manner is computationally infeasible. Firstly, the number of variables is still rather large: $3(s-1)(p+2)$, corresponding to the parameters $\{(P_i, \theta_i, \mathbf{r}_i)\}_{i=2\dots s}$. Secondly, the objective function is rather expensive to compute, as we discuss in the next paragraph. It turns out however, that optimizing the entire sequence would be problematic even if it were computationally feasible, due to the difficulty of finding a good starting point for the optimization. Since the objective function is non-convex, it is essential to be able to find a starting point which is near to a good local minimum, but it is unclear how to initialize all frames $2, 3, \dots s$ given only the first frame and the raw scanner data. Fortunately, both the computational issue and that of the starting point are easily dealt with by incrementally optimizing within a moving temporal window. In particular, we first optimize frame 2, then frames 2-3, frames 2-4, frames 3-5, frames 4-6, *etc.* With the exception of the first two steps, we always optimize a window of three frames, with all previous frames held fixed. Importantly, it is now reasonable to

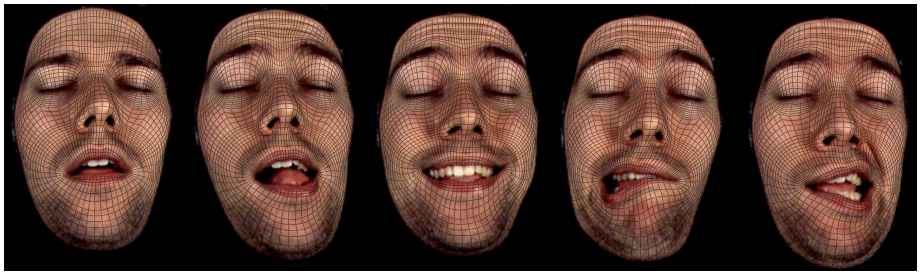

Fig. 3. A tracking example visualized by projecting the tracked mesh into the color camera image.

simply initialize the parameters of each newly included frame with those of the previous frame at the end of the previous optimization step.

Note that although we optimize on a temporal window with the other frames fixed, we include in the objective function all frames from the first to the current, eventually encompassing the entire sequence. Hence $E_{\text{col.}}$ forces each vertex inside the optimization window to stay within regions that have a color similar to that "seen" previously by the given vertex at previous time steps. One could also treat the final output of the incremental optimization as a starting point for optimizing the entire sequence with all parameters unfixed, but we found this leads to little change in practice. This is not surprising as, given the moving window of three frames, the optimizer essentially has three chances to get each frame right, with forward and backward look-ahead of up to two frames.

Parameter Selection. We first determined the parameters of the implicit surface/color models, and the deformation-based re-parameterization, can these can be visually verified independently of the tracking. Choosing the other parameters values was fairly straightforward, as *e.g.* tracking color and staying near the implicit surface are goals which typically compete very little — either can be satisfied without compromising the other. Hence the results are relatively insensitive to the $\alpha_{\text{imp.}}/\alpha_{\text{col.}}$. To determine suitable parameter setttings for $\alpha_{\text{imp.}}, \alpha_{\text{col.}}, \alpha_{\text{acc.}}$ and $\alpha_{\text{reg.}}$, we employed the following strategy. First, we removed a degree of freedom by fixing without loss of generality $\alpha_{\text{imp.}} = 1$. Next we assumed that the implicit surface was sufficiently reliable, and treated the distance to surface term almost like the hard constraint $E_{\text{imp.}} = 0$ by setting the next parameter $\alpha_{\text{col.}}$ to be $1/100$. We then took a sample dataset and ran the system over a 2D grid of values of $E_{\text{acc.}}$ and $E_{\text{reg.}}$, inspected the results visually, and fixed these two parameters accordingly for subsequent experiments.

3 Results

Tracking results are best visualised with animation, hence the majority of our results are presented in the accompanying video[1]. Here we discuss the perfor-

[1] `http://www.viddler.com/explore/anon42/videos/1/?secreturl=20055407`

mance of the system, and provide images of results of the tracking algorithm, which ran on a 64 bit, 2.4 GHz *AMD Opteron 850* processor with 4 GB of RAM, using a mixture of *Matlab* and *C++* code. We focus on timings for face data, and only report averages since the timings vary little over identity/performance.

The recording length is currently limited to 400 frames by operating system constraints. Note that this limitation is not due to our tracking algorithm, which has constant memory and linear time requirements in the length of the sequence. The dominating computation is evaluation of the objective function and its gradient during the optimization phase, and of this, around 80% of the time is on nearest neighbor searches into the scanner data using the algorithm of [10], in order to evaluate the implicit surface and color models. Including the 1-2 seconds required to build the data structure of the nearest neighbor search algorithm for each temporal window, the optimization phase of the tracking algorithm required around 20 seconds per frame. Note that only a small fraction of the recorded data needs to be stored in RAM at any given time. Note also that the computation times seem to scale roughly linearly with template mesh density. For example the four-fold upsampled template mesh in the video needed ≈ 3.5 times the computation time.

The tracking results in the accompanying video is convincing, and exhibits very little accumulation of error, as can be seen by the consistent alignment of template mesh to the neutral expression in the first and last frames. As no markers were used, the color camera images provide photo realistic expression wrinkles. A challenging example is shown in Figure 3, where the algorithm convincingly captures complex deformations. Here we provide a few comments on the accompanying video, which contains far more results than this paper[1]. To test the reliance on color we applied face paint to the female subject. The deterioration in performance is graceful in spite of both the high specularity of the paint and the sparseness of the color information. To demonstrate that the system is not specific to faces we provide an example in which colored cloth is tracked using no change to the processing pipeline, except for a different template mesh topology. The cloth tracking exhibits only minor inaccuracies around the border of the mesh where there is less information to resolve the ambiguities due to plain colored and strongly shadowed regions. A final example in the video[1] shows a uniformly colored, deforming, and rotating piece of foam being tracked using shape cues alone.

4 Discussion and Future Work

By design, our algorithm does not use optical flow calculations as the basis for the surface tracking. Rather, we combine shape and color information on a coarser scale, under the assumption that the color does not change excessively on any part of the surface. This assumption did not cause major problems in the case of expression wrinkles, as such wrinkles tend to appear and disappear on a part of the face with little relative motion with respect to the skin. Hence, in terms of the color penalty in the objective function, wrinkles do not induce a strong

force in any specific direction. Although there are other lighting effects which are more systematic, such as specularities, and self shadowing, we believe these do not represent a serious practical concern for the following reasons. Firstly, we found that in practice the changes caused by shadows and highlights were largely accounted for by the redundancy in color and shape over time. Secondly, it would be easy to reduce the severity of these lighting effects using light polarisers, more strobes and lighting normalization based on a model of the fixed scene lighting. Due to the general lack of available data, we were unable to systematically compare the performance of our system with that of others. To make a first step towards establishing a benchmark, we intend to publish data from our system, in order to allow future comparisons. The tracking system we have presented is automated, however it is straightforward to modify the energy functional we minimize in order to allow the user to edit the result by adding vertex constraints for example. It would also be interesting to develop a system which can improve the mesh regularisation terms in a face specific manner, by learning from previous tracking results. Another interesting direction is intelligent occlusion handling, which could overcome some of the limitations of structured light methods, and also allow the tracking of more complex self occluding objects.

A KNN Implicit Surface and Color Models

In this appendix, we motivate and define our nearest neighbor based implicit surface and color models. Our approach falls into the category of *partition of unity* methods, in which locally approximating functions are mixed together to form a global one. Let Ω be our domain of interest, and assume that we have a set of non-negative (and typically compactly supported) functions $\{\varphi_i\}$ which partition unity, *i.e.* $\sum_i \varphi_i(\mathbf{x}) = 1, \forall \mathbf{x} \in \Omega$. Now let $\{f_i\}$ be a set of locally approximating functions for each $\sup(\varphi_i)$. The partition of unity approximating function on Ω is $f(\mathbf{x}) = \sum_i \varphi_i(\mathbf{x}) f_i(\mathbf{x})$. The φ_i are typically defined implicitly by way of a set of compactly supported auxiliary functions $\{w_i\}$. Provided the w_i are non-negative and satisfy $\sup(w_i) = \sup(\varphi_i)$, the following choice is guaranteed to be a partition of unity: $\varphi_i = \frac{w_i}{\sum_j w_j}$. Presently we take the extreme approach of associating a local approximating function f_i with each data point from the set $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_m \in \mathbb{R}^4$, produced by our scanner. In particular, for the implicit surface embedding function $f_{\text{imp.}} : \mathbb{R}^4 \to \mathbb{R}$, we associate with $\mathbf{x}_i$ the linear locally approximating function $f_i(\mathbf{x}) = (\mathbf{x} - \mathbf{x}_i)^\top \mathbf{n}_i$, where $\mathbf{n}_i$ is the surface normal at $\mathbf{x}_i$. For the color model $f_{\text{col.}} : \mathbb{R}^4 \to \mathbb{R}^3$, the local approximating functions are simply the constant vector-valued functions $f_i(\mathbf{x}) = \mathbf{c}_i$, where $\mathbf{c}_i \in \mathbb{R}^3$ represents the RGB color at $\mathbf{x}_i$. Note that the above description constitutes a slight abuse of notation due to our having redefined f_i twice.

To define the φ_i, we first assume w.l.o.g. that $d_1 \leq d_2 \leq \ldots \leq d_k \leq d_i, \ \forall i > k$, where $\mathbf{x}$ is our evaluation point and $d_i = \|\mathbf{x} - \mathbf{x}_i\|$. In practice, we obtain such an ordering by way of a k nearest neighbor search using the *TSTOOL* software library [10]. By now letting $r_i \equiv d_i/d_k$ and choosing $w_i = (1 - r_i)_+$, it is easy to see that the corresponding φ_i are continuous, differentiable almost

everywhere, and that we only need to examine the k nearest neighbors of $\mathbf{x}$ in order to compute them. Note that the nearest neighbor search costs are easily amortized between the evaluation of $f_{\text{imp.}}$ and $f_{\text{col.}}$.

Larger values of k average over more local estimates and hence lead to smoother functions — for our experiments we fixed $k = 50$. Note that the nearest neighbor search requires Euclidean distances in 4D, so we must decide, say, what spatial distance is equivalent to the temporal distance between frames. Too small a spatial distance will treat each frame separately, too large will smear the frames temporally. The heuristic we used was to adjust the time scale such that on average approximately half of the k nearest neighbors of each data point come from the same time (that is, the same 3D frame from the scanner) as that data point, so that the other half come from the surrounding frames. In this way we obtain functions which vary smoothly through space and time. Note that it is easy to visually verify the effect of this choice by rendering the implicit surface and color models, as demonstrated in the accompanying video. This method is particularly efficient when we optimize on a moving window as discussed in Section 2.2. In this case, reasonable assumptions imply that the implicit surface and color models enjoy setup and evaluation costs of $O(q \log(q))$ and $O(k \log(q))$ respectively, where q is the number of vertices in a single 3D frame.

References

1. Zhang, L., Snavely, N., Curless, B., Seitz, S.M.: Spacetime faces: High-resolution capture for modeling and animation. In: ACM SIGGRAPH. (August 2004)
2. Borshukov, G., Lewis, J.P.: Realistic human face rendering for the matrix reloaded. In: SIGGRAPH 2003 Sketches, New York, ACM Press (2003)
3. Wand, M., Jenke, P., Huang, Q., Bokeloh, M., Guibas, L., Schilling, A.: Reconstruction of deforming geometry from time-varying point clouds. In: SGP '07: Proc. fifth Eurographics symp. on Geometry processing, Aire-la-Ville, Switzerland, ACM, Eurographics Association (2007) 49–58
4. Huang, X., Zhang, S., Wang, Y., Metaxas, D., Samaras, D.: A hierarchical framework for high resolution facial expression tracking. In: Articulated and non-rigid motion. Volume 1., Washington, DC, USA, IEEE Computer Society (2004)
5. Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., Seidel, H.P.: Multi-level partition of unity implicits. ACM Trans. on Graphics **22**(3) (July 2003) 463–470
6. Desbrun, M., Meyer, M., Schröder, P., Barr, A.H.: Discrete differential-geometry operators for triangulated 2-manifolds. VisMath **2** (2002) 35–57
7. Kazhdan, M., Bolitho, M., Hoppe, H.: Poisson surface reconstruction. In: SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing, Aire-la-Ville, Switzerland, Switzerland, ACM, Eurographics Association (2006) 61–70
8. Botsch, M., Kobbelt, L.: An intuitive framework for real-time freeform modeling. In: SIGGRAPH, New York, NY, USA, ACM, ACM (2004) 630–634
9. Botsch, M., Sorkine, O.: On linear variational surface deformation methods. IEEE Trans. Visualization and Computer Graphics **14**(1) (2008) 213–230
10. Merkwirth, C., Parlitz, U., Lauterborn, W.: Fast nearest neighbor searching for nonlinear signal processing. Phys. Rev. E **62**(2) (2000) 2089–2097