EUROGRAPHICS 2002 / G. Drettakis and H.-P. Seidel
(Guest Editors)

Volume 21 (2002), Number 3

Dynamic Textures for Image-based Rendering of Fine-Scale 3D Structure and Animation of Non-rigid Motion

Dana Cobzaş, Keith Yerex and Martin Jägersand

Department of Computing Science, University of Alberta, Canada
{dana,keith,jag}@cs.ualberta.ca

Abstract

The problem of capturing real world scenes and then accurately rendering them is particularly difficult for fine-scale 3D structure. Similarly, it is difficult to capture, model and animate non-rigid motion. We present a method where small image changes are captured as a time varying (dynamic) texture. In particular, a coarse geometry is obtained from a sample set of images using structure from motion. This geometry is then used to subdivide the scene and to extract approximately stabilized texture patches. The residual statistical variability in the texture patches is captured using a PCA basis of spatial filters. The filters coefficients are parameterized in camera pose and object motion. To render new poses and motions, new texture patches are synthesized by modulating the texture basis. The texture is then warped back onto the coarse geometry. We demonstrate how the texture modulation and projective homography-based warps can be achieved in real-time using hardware accelerated OpenGL. Experiments comparing dynamic texture modulation to standard texturing are presented for objects with complex geometry (a flower) and non-rigid motion (human arm motion capturing the non-rigidities in the joints, and creasing of the shirt).

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Image Based Rendering

1. Introduction

Many graphics rendering and animation techniques generate images by texture mapping a 3D scene model. This approach works well if the geometric model is sufficiently detailed but photorealism is still hard to achieve. Computer vision structure-from-motion techniques can be used to reconstruct the model from a set of sample images. The model can be reconstructed based on points, planar patches[16] or lines[3] but the process is in general tedious and requires a lot of human intervention. For texture mapping the model is decomposed into small planar patches that are then textured from the sample view that is closest to the desired position. An advantage of metric reconstruction is that the model can be correctly reprojected under perspective projection. Non-euclidean models are more easy to construct from a set of input images[4, 19], but without additional metric information it is difficult to specify physically correct novel views.

Recent research in image-based rendering (IBR) offers an alternative to these techniques by creating a non-geometric model of the scene from a collection of images. The idea behind the IBR techniques is to sample the *plenoptic function*[12] for the desired scene. There are two main approaches to this problem. One is to sample the plenoptic function under some viewing constraints that limit the camera scene to bounding box[5, 10] or to only rotations [18]. Another approach is presented in [9], where a photoconsistent volumetric model is reconstructed from a set of input images by organizing the rays. These methods will theoretically generate correct renderings but are practically hard to apply for real scenes and require calibrated cameras.

Some work towards combining model-based and image-based has been performed. Debevec *et. al.*[3] renders novel views with view dependent textures, chosen either from the closest real sample image, or as a linear combination of the nearest images. We extend this work by functionally representing and parameterizing the view dependent texture by combining image-plane based synthesis [8, 7, 11] with tracking and scene geometry estimation. Buehler[2] proposed a generalization of the lumigraph by reconstructing an approximate geometric proxy for a scene from a collection of images. A novel view is obtained by combining rays from input images using a "camera blending field". Instead of focusing on the ray set here we represent the variability in the texture, which also allow us to extend applications to non-rigid and time-varying scenes.

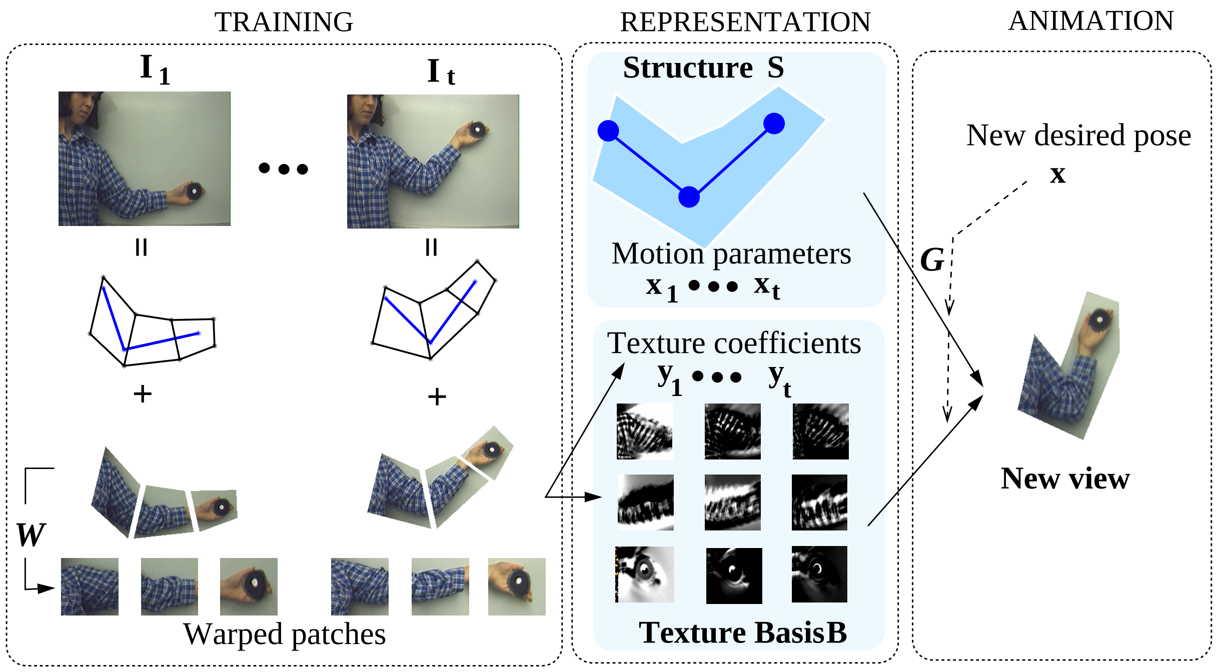

Figure 1: *A sequence of training images $I_1 \cdots I_t$ is decomposed into geometric shape information and dynamic texture for a set of quadrilateral patches. The scene structure S and motion parameters $\mathbf{x}_1 \cdots \mathbf{x}_t$ are determined from the projection of the structure. The dynamic texture for each quadrilateral is decomposed into its projection $\mathbf{y}_1 \cdots \mathbf{y}_t$ on an estimated basis B. For a given desired position $\mathbf{x}$, a novel image is generated by warping new texture synthesized from the basis B on the projected structure.*

There are several practical challenges with current techniques: (1) For the model-based approaches, generating a detailed model from images requires dense point correspondences. In practice, usually only a few points can be tracked reliably and accurately and object structure estimated at best coarsely. (2) In conventional texture-based rendering, the placement of triangles so that real edges on the object are aligned with edges in the model is critical. However, with a sparse model obtained from images of an otherwise unknown object this is difficult to ensure. (3) Without imposing a geometric structure on the scene, rendering arbitrary views requires a dense sampling of the plenoptic function which is practically hard to accomplish for real scenes. (4) IBR techniques are difficult to apply in dynamic scenes.

In this paper we present a method inbetween image- and model-based rendering. Using a coarse geometric model, we can approximately stabilize image motions from large viewpoint changes. The residual image variability is accounted for using a varying texture image, *dynamic texture*, synthesized from a spatial texture basis, which is described in Section 2.1. Section 2.3 describes the geometric models, and how they are captured using structure-from-motion[19]. Both the structure and dynamic texture variation is parameterized in terms of object-camera pose, and object articulation, hence allowing rendering of new positions and animation of object motions. We show how to implement the *dynamic texture* rendering using hardware accelerated Open-GL in Section 3, and in Section 4 we verify practical performance in qualitative and quantitative experiments capturing and animating a house, a flower, and a human arm.

2. Theory

From a geometric view, IBR techniques relate the pixelwise correspondence between sample images I_t and a desired views I_w. This can be formulated using a warp function w to relate $I_t(w) = I_w$. If $I_t \approx I_w$ then w is close to the identity function. However, to relate arbitrary viewpoints, w can be quite complex, and current IBR methods generally require carefully calibrated cameras in order to have a precise geometric knowledge of the ray set [5, 10].

In our method the approximate texture image stabilization achieved using a coarse model reduces the difficulty of applying IBR techniques. The residual image (texture) variability can then be coded as a linear combination of a set of spatial filters. (Figure 1). More precisely, given a training sequence of images I_t and tracked points $[\mathbf{u}_t, \mathbf{v}_t]$, a simplified geometric structure of the scene S and a set of motion parameters $\mathbf{x}_t$ that uniquely characterize each frame is estimated from the tracked points (section 2.3). The reprojection of the structure given a set of motion parameters $\mathbf{x}$ is obtained by

$$[\mathbf{u}, \mathbf{v}] = \mathcal{G}(S, \mathbf{x}) \tag{1}$$

where $\mathcal{G}$ is a projection function for a particular geometry of the scene. Using the re-projected structure $[\mathbf{u}_t, \mathbf{v}_t]$ each sample image I_t is divided into Q quadrilateral regions I_{qt} (I_{qt}

having zero support outside the quad). Each quad is warped to a standard shape (rectangle) I_{wqt} (Section 2.1).

$$I_t = \sum_{q=1}^{Q} I_{qt} \tag{2}$$

$$I_{wqt} = I_{qt}(\mathcal{W}(\mathbf{u}_t, \mathbf{v}_t)) \tag{3}$$

Using the algorithm described in section 2.1 we then compute for each quadrilateral a set of basis images B_q that capture the image variability caused by geometric approximations and illumination changes and the set of corresponding blending coefficients $\mathbf{y}_{qt}$

$$I_{wqt} = B_q \mathbf{y}_{qt} + \bar{I_q} \tag{4}$$

where $\bar{I_q}$ represents the mean image. To generate a new view we first estimate the projection of the structure S in the desired view (specified by motion parameters $\mathbf{x}$ in Equation 1). Then, texture modulation coefficients $\mathbf{y}$ are computed and a texture corresponding to the new view is blended (Equation 4). Finally the texture is warped to the projected structure (Equations 3,2). The following sections describe this process in detail.

2.1. Dynamic textures

The purpose of the *dynamic texture* is to allow for a non-geometry based modeling and synthesis of intensity variation during animation of a captured model. To illustrate how motion can be generated using a spatial basis consider two simple examples: 1/ A drifting grating $I = \sin(u+at)$ can be synthesized by modulating only a sin and cos basis $I = \sin(u+at) = \sin(u)\cos(at) + \cos(u)\sin(at) = \sin(u)y_1 + cos(u)y_2$, where y_1 and y_2 are mixing coefficients. 2/ Small image translations can be generated by $I = I_0 + \frac{\partial I}{\partial u}\Delta u + \frac{\partial I}{\partial v}\Delta v$. Here the image derivatives $\frac{\partial I}{\partial u}$ and $\frac{\partial I}{\partial v}$ form a spatial basis. Below we first extend this to 6 and 8 parameter warps representing texture transforms, with depth, non-rigidity and lighting compensation. Having shown generatively based on one image that these kinds of intensity variation can indeed be represented using a linear basis we then describe how estimate this basis from actual image variability in a more stable way from the statistics over long sequences of images.

2.1.1. Parameterizing image variability

Formally, consider an image stabilization problem. Under an image constancy assumption the light intensity from an object point p is independent of the viewing angle[6]. Let $I(t)$ be an image (patch) at time t, and I_w a stabilized (canonical) representation for the same image. In general, then there exists some coordinate remapping w s.t.

$$I_w(\mathbf{p}) = I(w(\mathbf{p}), t) \tag{5}$$

Hence, I_w represents the image from some hypothetical viewing direction, and w is a function describing the rearrangement of the ray set from the current image $I(t)$ to I_w. In principle w could be found if accurate models are available for the scene, camera and their relative geometry.

In practice, at best an approximate function $\hat{w}$ can be found, which may be parameterized in time (e.g. in movie compression) or pose (e.g. in structure from motion and pose tracking). Below we develop mathematically the effects of this approximation. In particular we study the residual image variability introduced by the imperfect stabilization achieved by $\hat{w}$, $\Delta I = I(\hat{w}, t) - I_w$. Let $\hat{w} = w + \Delta f$ and rewrite as an approximate image variability to the first order (dropping t):

$$\Delta I = I(w+\Delta w) - I_w = I(w) + \frac{\partial}{\partial w} I(w)\Delta w - I_w = \frac{\partial}{\partial w} I(w)\Delta w \tag{6}$$

The above equation expresses an optic flow type constraint in an abstract formulation without committing to a particular form or parameterization of w. In practice, the function w is usually discretized using e.g. triangular or quadrilateral mesh elements. Next we give examples of how to concretely express image variability from these discrete representations under different camera geometries.

2.1.2. Structural image variability

Affine Under a weak perspective (or orthographic) camera geometry, plane-to-plane transforms are expressed using an affine transform of the form:

$$\begin{bmatrix} u_w \\ v_w \end{bmatrix} = \mathcal{W}_a(\mathbf{p}, \mathbf{a}) = \begin{bmatrix} a_3 & a_4 \\ a_5 & a_6 \end{bmatrix} \mathbf{p} + \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \tag{7}$$

This is also the standard image-to-image warp supported in OpenGL. Now we can rewrite the image variability Eq. 6 resulting from variations in the six affine warp parameters as:

$$\Delta I_a = \sum_{i=1}^{6} \frac{\partial}{\partial a_i} I_w \Delta a_i = \left[\frac{\partial I}{\partial u}, \frac{\partial I}{\partial v}\right] \begin{bmatrix} \frac{\partial u}{\partial a_1} \cdots \frac{\partial u}{\partial a_6} \\ \frac{\partial v}{\partial a_1} \cdots \frac{\partial v}{\partial a_6} \end{bmatrix} \Delta[a_1 \dots a_6]^T \tag{8}$$

Let $\{I\}_{discr} = \mathbf{I}$ be a discretized image flattened along the column into a vector, and let '$*\mathbf{u}$' and '$*\mathbf{v}$' indicate point-wise multiplication with column flattened camera coordinate u and v index vectors. Rewrite the inner derivatives to get an explicit expression of the 6 parameter variability in terms of spatial image derivatives:

$$\Delta \mathbf{I}_a = \left[\frac{\partial \mathbf{I}}{\partial u}, \frac{\partial \mathbf{I}}{\partial v}\right] \begin{bmatrix} 1 & 0 & *\mathbf{u} & 0 & *\mathbf{v} & 0 \\ 0 & 1 & 0 & *\mathbf{u} & 0 & *\mathbf{v} \end{bmatrix} [y_1, \dots, y_6]^T = [\mathbf{B}_1 \dots \mathbf{B}_6][y_1, \dots, y_6]^T = B_a \mathbf{y}_a \tag{9}$$

where $[\mathbf{B}_1 \dots \mathbf{B}_6]$ can be interpreted as a variability basis for the affine transform.

Projective For a perspective camera the physically correct plane-to-plane transform is represented by a projective homography,[4, 18]. Using homogeneous coordinates $\mathbf{p}_h = [u, v, 1]^T$ points in two planar scene views I_1 and I_2 are related by:

$$\begin{bmatrix} u \\ v \\ \lambda \end{bmatrix} = \mathcal{W}_h(\mathbf{p}_h, \mathbf{h}) = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{bmatrix} \mathbf{p}_h = H\mathbf{p}_h \tag{10}$$

This transformation is up to a scale, so in general H has eight independent parameters and it can be computed from four corresponding points in the two views. Analogously to the affine case we can rewrite the resulting first order image variability due to the homography parameters $h_1 \dots h_8$ in terms of a spatial derivatives:

$$\Delta \mathbf{I}_h = \left\{ I \sum_{i=1}^{8} \frac{\partial \mathbf{I}}{\partial h_i} \Delta h_i \right\}_{discr} = \ldots = \\ [\mathbf{B}_1 \ldots \mathbf{B}_8][y_1, \ldots, y_8]^T = B_h \mathbf{y}_h \quad (11)$$

Depth compensation While common mesh discretizations model objects as piece-wise planar, real world objects are seldom perfectly planar. This non-planarity will introduce a parallax error in the dependent on the relative depth difference between the true scene point and the appproximating plane. For small depth variations this results in small image plane translations. Hence, similar to the small coordinate translations caused by tracking and mesh geometry, we can write the resulting intensity variation due to the depth parallax by modulating a linear basis:

$$\Delta \mathbf{I}_d = [\mathbf{B}_{d1}, \mathbf{B}_{d2}][y_{d1}, y_{d2}]^T \quad (12)$$

Non-rigidity We consider only non-rigidities where the shape change is a function of some measurable quantity $\mathbf{x} \in \Re^n$. In this paper we choose $\mathbf{x}$ from pose and articulation parameters. Let $\mathbf{g}(\mathbf{x})$ (= $[u,v]^T$) represent the image plane projection of the non-rigid warp. We can then write the resulting first order image variation as:

$$\Delta \mathbf{I}_n = \left\{ \sum_{i=1}^{n} \frac{\partial \mathbf{I}}{\partial x_i} \Delta x_i \right\}_{discr} = \\ \left\{ \left[\frac{\partial \mathbf{I}}{\partial u}, \frac{\partial \mathbf{I}}{\partial v} \right] \left[\frac{\partial}{\partial x_1} g(x) \Delta x_1, \ldots, \frac{\partial}{\partial x_n} g(x) \Delta x_n \right] \right\}_{discr} = \\ [\mathbf{B}_1 \ldots \mathbf{B}_n][y_1, \ldots, y_n]^T = B_n \mathbf{y}_n \quad (13)$$

Illumination variation It has been shown that for a convex Lambertian object, the image variability due to different illumination can be expressed as a three dimensional linear basis[15, 6]. For a general object, the illumination component can be approximated with a low dimensional basis .

$$\Delta \mathbf{I}_l = [\mathbf{B}_1 \ldots \mathbf{B}_3][y_1 \ldots y_3]^T = B_l \mathbf{y}_l \quad (14)$$

2.1.3. Statistical image variability

In a real, imperfectly stabilized image sequence we can expect all of the above types of image variation, as well as unmodeled effects and noise $\Delta \mathbf{I}_e$. Hence, total residual image variability can be written as:

$$\Delta \mathbf{I} = \Delta \mathbf{I}_s + \Delta \mathbf{I}_d + \Delta \mathbf{I}_n + \Delta \mathbf{I}_l + \Delta \mathbf{I}_e = \\ B_s \mathbf{y}_s + B_d \mathbf{y}_d + B_n \mathbf{y}_n + B_l \mathbf{y}_l + \Delta \mathbf{I}_e = \mathbf{B}\mathbf{y} + \Delta \mathbf{I}_e \quad (15)$$

where $\Delta \mathbf{I}_s$ is either $\Delta \mathbf{I}_a$ or $\Delta \mathbf{I}_h$ depending on the geometry used. Approximations to $\mathbf{B}_a$ and $\mathbf{B}_h$ can be computed from image derivatives (this is used to bootstrap tracking), while $B_d \mathbf{y}_d + B_n \mathbf{y}_n + B_l \mathbf{y}_l +$ all require detailed geometric knowledge of the scene, camera and lighting. Note in Eq. 15 its linear form, where a set of basis textures, $B = [\mathbf{B}_1 \ldots \mathbf{B}_m]$, are mixed by a corresponding set of blending coefficients, $\mathbf{y} = [y_1 \ldots y_m]^T$. To avoid explicit modeling we instead observe actual image variability from an image sequence $\hat{I}(t)$, which has been stabilized using an approximate $\hat{f}$, so we now have a sequence of small variations $\Delta \hat{I}$. We compute a reference image as the pixel-wise mean image $\bar{\mathbf{I}} = \sum_{t=0}^{M} \frac{1}{M} \mathbf{I}_w(t)$, and form a zero mean distribution of the residual variability as $\hat{\mathbf{I}}_z(t) = \mathbf{I}_w(t) - \bar{\mathbf{I}}$. From these we can use standard PCA (principal component analysis) to compute a basis $\hat{B}$ which captures (up to a linear coordinate transform) the actually occurring image variation in the true B (Eq. 15)

Briefly we perform the PCA as follows. Form a measurement matrix $A = [\mathbf{I}_z(1), \ldots, \mathbf{I}_z(M)]$. The principle components are the eigen vectors of the covariance matrix $C = AA^T$. A dimensionality reduction is achieved by keeping only the first k of the eigenvectors. For practical reasons, usually $k \ll M \ll l$, where l is the number of pixels in the texture patch, and the covariance matrix C will be rank deficient. We can then save computational effort by instead computing $L = A^T A$ and eigen vector factorization $L = VDV^T$, where V is an ortho-normal and D a diagonal matrix. From the k first eigenvectors $\hat{V} = [\mathbf{v}_1 \ldots \mathbf{v}_k]$ of L we form a k-dimensional eigenspace $\hat{B}$ of C by $\hat{B} = A\hat{V}$. Using the estimated $\hat{B}$ we can now write a least squares optimal estimate of any intensity variation in the patch as

$$\Delta \mathbf{I} = \hat{B}\hat{\mathbf{y}}, \quad (16)$$

the same format as Eq. 15, but without using any a-priori information to model B. While $\hat{\mathbf{y}}$ captures the same variation as $\mathbf{y}$, it is not parameterized in the same coordinates, but related by an (unknown) linear transform. We are not explicitly interested in $\hat{\mathbf{y}}$ for the rendering, but instead need to know how to relate the estimated global pose $\hat{\mathbf{x}}$ to $\hat{\mathbf{y}}$. This is in general a non-linear function f (e.g. due to the angular parameterization of camera pose in the view synthesis, Fig. 8, or the non-linear kinematics in the arm animation, Fig. 11). To relate these we note that tracking and pose factorization (Eq. 23) gives us poses $[\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_M]$ for all training images in the sample sequence, and from the orthogonality of $\hat{V}$ we have that the corresponding texture mixing are the columns of $[\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_M] = \hat{V}^T$. Hence, many standard function approximation techniques are applicable. We use multidimensional spline interpolation to estimate $\hat{f}$. Comparing cubic and linear splines we found that linear splines suffice. During animation these can be computed quickly and efficiently from only the nearest neighbor points.

2.2. Interpretation of the variability basis

In our application, the geometric model captures gross image variation caused by large movements. The remaining variation in the rectified patches is mainly due to:

1. Tracking errors as well as errors due to geometric approximations (e.g. weak perspective camera) cause the texture to be sourced from slightly inconsistent locations in the training images. These errors can be modeled as a small deviation $\Delta[h_1, \ldots, h_8]^T$ in the homography parameters from the true homography, and causes image differences according to Eq. 11. The weak perspective approximations, as well as many tracking errors are persistent, and a function of object pose. Hence they will be captured by $\hat{B}$ and indexed in pose $\mathbf{x}$ by f.
2. Depth variation is captured by Eq. 12. Note that projected depth variation along the camera optic axis changes as a function of object pose.

3. Assuming fixed light sources and a moving camera or object, the light variation is a function of relative camera-object pose as well.

From the form of Equations 11 and 12 we expect that pose variations in the image sequence will result in a texture variability described by combinations of spatial image derivatives. In Fig.2 we compare numerically calculated spatial image derivatives to the estimated variability basis $\hat{B}$.

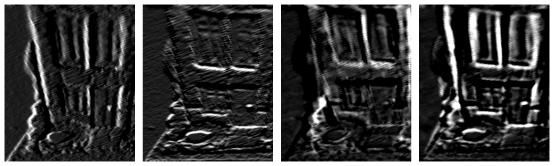

Figure 2: *Comparison between spatial derivatives* $\frac{\partial I_w}{\partial x}$ *and* $\frac{\partial I_w}{\partial y}$ *(left two texture patches) and two vectors of the estimated variability basis* $[\mathbf{B}_1, \mathbf{B}_2]$ *(right) for house pictures.*

In synthesizing texture to render a sequence of novel images the function f modulates the filter bank B so that the new texture dynamically changes with pose $\mathbf{x}$ according to $\mathbf{I}_w = B\hat{f}(\mathbf{x}) + \mathbf{I}$.

2.3. Geometric model

There are several techniques for estimating geometric structure from a set of images. Most of the methods assume a static scene and estimate the structure from a set of corresponding image points. Depending on the camera model and calibration data the estimated model can vary from a projective, affine to a metric or 3D Euclidean model. For a few images, under perspective projection assumption, the structure can be determined using epipolar geometry (two images) or trilinear tensor (three images) [4, 14]. In the case of video, *i.e.* long motion sequences, methods which utilize all image data in a uniform way are preferred. Such methods recover affine [19, 20, 13] or projective [17] structure using factorization approaches. Another approach to this problem is to start with an initial estimate of the model and refine it using sample images [16, 3].

For a dynamic scene or a nonrigid object the problem becomes much more complicated. A solution is to decompose the scene into rigid, static components and estimate each structure separately. Bregler [1] proposed a generalization of the Tomasi-Kanade factorization using linear combinations of a set of estimated basis shapes .

We propose here two structure-from-motion algorithms. The first one assumes rigid objects and estimates a metric geometry of the scene using a modified version of the Tomasi-Kanade factorization algorithm [19]. The second algorithm estimates the 2D geometry of an articulated arm assuming image plane motion.

2.3.1. Metric structure under weak perspective assumption

A general structure-from-motion algorithm starts with a set of corresponding features (point, lines, line segments) in a sequence of images of a scene or object and recovers the world coordinates of these features and the positions of the cameras (rotation and translation) relative to this representation under some viewing constraints (see Figure 3). The corresponding features can be established using tracking, correlation algorithms or by manual selection. In the most general case a 3D Euclidian structure is estimated assuming projective model of the camera. These algorithms require precise calibration of the camera, numerous corresponding features and are in general highly nonlinear and sensitive to tracking errors. Assuming a more simplified model of the camera (ortographic, weak perspective or paraperspective projection [4, 14, 13]), the problem is linearized and an affine structure of the scene is estimated using factorization. Using multiple images allows for stable solutions despite relatively few tracked points and typical tracking errors.

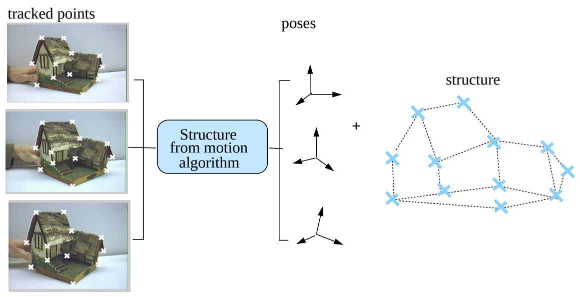

Figure 3: *A general structure from motion algorithm extracts the structure and camera poses from a set of tracked points.*

Here we have used an extension of the Tomasi-Kanade factorization algorithm[19] for weak perspective camera projection model similar to Weinshall and Kanade[20]. First, the algorithm recovers affine structure from a sequence of uncalibrated images. Then, a relation between the affine structure and camera coordinates is established. This is used to transform the estimated scene structure to an orthogonal coordinate frame. Finally, using similarity transforms expressed in metric rotations and translations, the structure can be reprojected into new, physically correct poses. Since we use only image information our metric unit of measure is pixel coordinates. We next describe a more detailed mathematical formulation of the problem.

Affine structure from motion Under weak perspective projection, a point $\mathbf{P}_i = (\mathbf{X}_i, \mathbf{Y}_i, \mathbf{Z}_i)^T$ is related to the corresponding point $p_{ti} = (u_{ti}, v_{ti})^T$ in image frame $I(t)$ by the following affine transformation:

$$\begin{aligned} u_{ti} &= s_t \mathbf{i}_t^T \mathbf{P}_i + a_t \\ v_{ti} &= s_t \mathbf{j}_t^T \mathbf{P}_i + b_t \end{aligned} \tag{17}$$

where $\mathbf{i}_t$ and $\mathbf{j}_t$ are the components along the camera rows and columns of the rotation R_t , s_t is a scale factor and (a_t, b_t) are the first components $\mathbf{t}1_t$ of the translation $\mathbf{t}_t$ (R_t and $\mathbf{t}_t$ aligns the camera coordinate system with the world reference system and represents the camera pose).

Rewriting 17 for multiple points (N) tracked in several frames (M)

$$W = RP + \mathbf{t}1 \tag{18}$$

where W is a $2M \times N$ matrix contains image measurements, R represents both scaling and rotation, P is the shape and $\mathbf{t}1$ is the translation in the image plane [20].

If the image points are registered with respect to their centroid in the image plane and the center of the world coordinate frame is the centroid of the shape points, the projection equation becomes:

$$\hat{W} = RP \;\; where \;\; \hat{W} = W - \mathbf{t}1 \tag{19}$$

Following [19], in the absence of noise we have $\text{rank}(\hat{W}) = 3$. Under most viewing conditions with a real camera the effective rank is 3. Considering the singular value decomposition of $\hat{W} = O_1 \Sigma O_2$ we form

$$\begin{array}{l} \hat{R} = O_1' \\ \hat{P} = \Sigma' O_2' \end{array} \tag{20}$$

where O_1', Σ', O_2' are respectively defined by the first three columns of O_1, the first 3×3 matrix of Σ and the first three rows of O_2 (assuming the singular values are ordered in decreasing order).

Metric constraints The structure and camera pose recovered using the factorization algorithm is in an affine frame reference system. To control the new positions of the rendered images in a metric reference system they need to be mapped to an orthogonal coordinate frame which in our case is aligned with the pixel coordinate system of the camera row and column.

The matrices $\hat{R}$ and $\hat{P}$ are a linear transformation of the metric scaled rotation matrix R and the metric shape matrix P. More specifically there exist a 3×3 matrix Q such that:

$$\begin{array}{l} R = \hat{R}Q \\ P = Q^{-1}\hat{P} \end{array} \tag{21}$$

Q can be determined by imposing constraints on the components of the scaled rotation matrix R:

$$\begin{array}{ll} \hat{\mathbf{i}}_t^T QQ^T \hat{\mathbf{i}}_t = \hat{\mathbf{j}}_t^T QQ^T \hat{\mathbf{j}}_t & (= s_t^2) \\ \hat{\mathbf{i}}_t^T QQ^T \hat{\mathbf{j}}_t = 0 & t \in \{1..M\} \end{array} \tag{22}$$

where $\hat{R} = [\hat{\mathbf{i}}_1 \cdots \hat{\mathbf{i}}_M \hat{\mathbf{j}}_1 \cdots \hat{\mathbf{j}}_M]^T$ The first constraint assures that the corresponding rows $s_t \mathbf{i}_t^T$, $s_t \mathbf{j}_t^T$ of the scaled rotation R in Eq. 18 are unit vectors scaled by the factor s_t and the second equation constrain them to orthogonal vectors. This generalizes [19] from an orthographic to a weak perspective case. The resulting transformation is up to a scale and a rotation of the world coordinate system. To eliminate the ambiguity we align the axis of the reference coordinate system with the first frame and estimate only eight parameters in Q (fixing a scale).

To extract pose information for each frame we first estimate the scale factor s_t and rotation components $\mathbf{i}_t$ and $\mathbf{j}_t$ by computing the norm of the rows in R that will represent the scale factors and then normalizing them. Considering that $\mathbf{i}_t$ and $\mathbf{j}_t$ can be interpreted as the orientation of the vertical and horizontal camera image axes in the object space, we compute the direction of the camera projection axis $\mathbf{k}_t = \mathbf{i}_t \times \mathbf{j}_t$. We now have a complete representation for the metric rotation that we parametrize with Euler angles $\mathbf{r}_t = [\psi_t, \theta_t, \varphi_t]$. Each camera pose is represented by the motion parameter vector

$$\mathbf{x}_t^{\mathbf{a}} = [\mathbf{r}_t, s_t, a_t, b_t] \tag{23}$$

The geometric structure is represented by

$$S^{\mathbf{a}} = P \tag{24}$$

and its reprojection given a new pose $\mathbf{x}^{\mathbf{a}} = [\mathbf{r}, s, a, b]$ is estimated by

$$[\mathbf{u}, \mathbf{v}] = \mathcal{G}^{\mathbf{a}}(S^{\mathbf{a}}, \mathbf{x}^{\mathbf{a}}) = sR(\mathbf{r})S^{\mathbf{a}} + \begin{bmatrix} a \\ b \end{bmatrix} \tag{25}$$

where $R(\mathbf{r})$ represents the rotation matrix given the Euler angles $\mathbf{r}$.

2.3.2. Planar motion of a kinematic arm

One way to estimate the shape of a nonrigid object is to decompose it into rigid parts and use standard structure from motion algorithms to estimate the geometry of each component. They have to be later unified into a common reference frame using additional constraints (for example kinematics constraints). An interesting solution, proposed by Bregler [1], approximates a non-rigid object by a linear combination of a set of estimated basic shapes.

As an illustration of this idea we estimated the geometry of an arm assuming the motion is parallel to the image plane, so the problem becomes planar. Knowing the position of the palm (h_u, h_v), shoulder (s_u, s_v) and the length of the two arm segments a_1, a_2, the position of the elbow (e_u, e_v) can be estimated using inverse kinematics equations for a two link planar manipulator (see Figure 5)

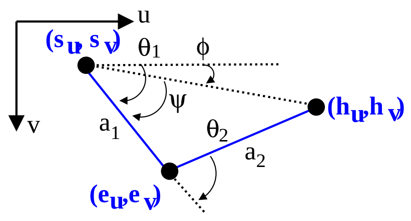

Figure 5: *Arm kinematics*

$$\begin{array}{l} e_u = a_1 cos\theta_1 + s_u \\ e_v = a_2 cos\theta_1 + s_v \end{array} \tag{26}$$

where

$$\begin{array}{l} \theta_1 = \phi - \psi \\ \phi = atan2(h_v - s_v, h_u - s_u) \\ \psi = atan2(a_2 sin\theta_2, a_1 + a_2 cos\theta_2) \\ \theta_2 = \pm 2tan^{-1}\sqrt{\frac{(a_1+a_2)^2 - (h_u^2+h_v^2)}{(h_u^2+h_v^2) - (a_1-a_2)^2}} \end{array} \tag{27}$$

In practice we tracked the shoulder and the palm and computed the position of the elbow using the inverse kinematics formulas 26,27. The lengths of the two links are estimated from a subset of training images using manual selection of the elbow joint. Assuming the shoulder is not moving we estimate its position (s_u, s_v) by averaging the values of the

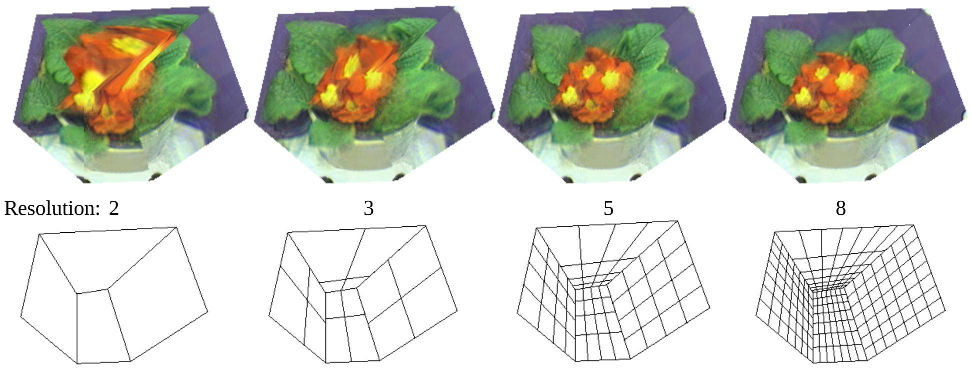

Figure 4: *Approximation to homographic warping in OpenGL with subdivision at different resolutions*

tracked shoulder points. There are in general two solution for the elbow joint but we kept only the one with positive θ_2 that produces a natural motion of the arm.

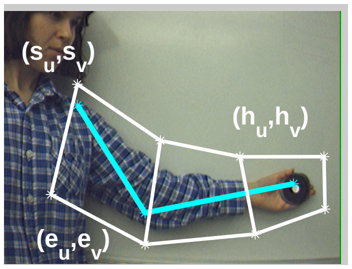

Figure 6: *Quadrilaterals defined on the wire frame arm to enclose the arm texture*

For texture mapping the arm we defined three quadrilaterals that follow the wire frame $(s_u,s_v),(e_u,e_v),(h_u,h_v)$ (see Figure 6). They have a unique representation $\mathcal{G}^{\mathbf{k}}$ given the geometry of the two link wire frame. We can express the geometry of the arm as

$$S^{\mathbf{k}} = [a_1, a_2, s_u, s_v, \gamma] \tag{28}$$

where γ represents the parameters of heuristic shape we impose on the wire frame arm. The motion of the arm is controlled by the position of the palm

$$\mathbf{x}^{\mathbf{k}} = (h_u, h_v) \tag{29}$$

3. Implementation

Our system runs entirely on consumer grade PCs using linux and other free software packages. For the tracking and model capture we use a Pyro ieee1394 webcam, interfaced to linux on a patched 2.4.4 kernel and using public ieee1394 digital video libraries from `www.sourceforge.net`. Video capture, tracking and geometry analysis are implemented as separate processes using PVM (Parallel Virtual Machine). The systems used for both capturing and rendering are 1.4 GHz machines with the hardware rendering running in OpenGL on an NVidia GEForce2 chip set.

3.1. Algorithm

Training data We use XVision[6] to set up a video pipeline and implement real time tracking. To initiate tracking, the user selects a number (about a dozen or more) high-contrast regions in the image with the mouse. These are grabbed, and tracked by estimating image variability $\mathbf{y}$ in Eq. 9 from the input video images. Most scenes are difficult to tile fully with trackable regions, while it is possible to succesfully track small subparts. From tracking we extract the first two parameters from $\mathbf{y}$, representing image plane position, and use these points to form large quadrilaterals over the whole scene. From these quadrilaterals textures are sampled at rates from 2 to 5 Hz, while the trackers run in the background at about 30Hz (for up to 20 trackers).

Geometric Model We estimate the geometry reprojection function $\mathcal{G}$ based on the samples from the training data using one of the techniques described in section 2.3.

Dynamic Texture

1. Warp each patch $I_q(t)$ to a standard shape $I_wq(t)$ using warping function $\mathcal{W}$ (Equations 7 or 10). The standard shape is chosen to be square with sides of length 2^n to accommodate hardware accelerated rendering (see 3.2).
2. Form a zero mean sample, and perform the PCA as described in section 2.1. Keep the first k basis vectors B_q, and the corresponding coefficients for each frame in the training sequence $\mathbf{y}_q(t)$.

New View Animation

1. For each frame in the animation compute the reprojection $[u,v]$ from the desired pose $\mathbf{x}$ as in Equation 1.
2. Calculate texture coefficients $\mathbf{y}_q$ for each texture by interpolating the coefficients of the nearest neighbors from the training data.
3. Compute the new textures in the standard shape using Equation 4 and rewarp the textures onto the calculated geometry. (These two operations are performed simultaneously in hardware as described in the next section)

3.2. Hardware Rendering

Texture Blending To render the *dynamic texture* we use the texture blending features available on most consumer 3D

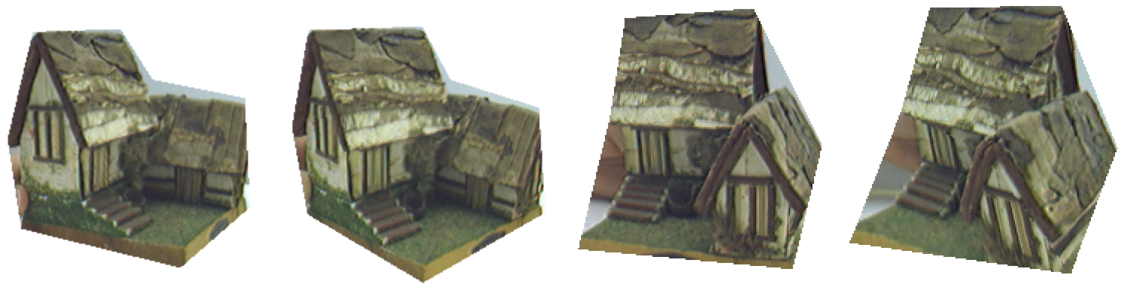

Figure 7: *House sequence animated at different viewpoints (deviation of about* 10° *from the original sample images)*

graphics cards. The rendering hardware used is designed for textures containing positive values only, while the spatial basis, Equation 16 is a signed quantity. We rewrite this as a combination of two textures with only positive components:

$$I_{wq}(t) = B_q^+ \mathbf{y}_q(t) - B_q^- \mathbf{y}_q(t) + \bar{I_q}$$

Where B_q^+ contains only the positive elements from B_q (and 0 in the place of negative elements) and B_q^- contains the absolute values of all negative elements from B_q. Then, before drawing each basis texture, the blending mode can be set to either scale by a coefficient and add to the frame buffer, or scale by a coefficient and subtract from the frame buffer (depending on the sign of the coefficient). A new view is rendered as in the following pseudocode:

```
for(each $q$)
{
  // draw the mean
  BindTexture($\bar{I_q}$);
  DrawQuad($q$);

  // add basis textures
  for(each $i$)
  {
    SetBlendCoefficient($|\mathbf{y}_{qi}(t)|$);

    BindTexture($B_{qi}^+$);
    if($\mathbf{y}_{qi}(t) > 0$) SetBlendEquation(ADD);
    else SetBlendEquation(SUBTRACT);
    DrawQuad($q$);

    BindTexture($B_{qi}^-$);
    if($\mathbf{y}_{qi}(t) > 0$) SetBlendEquation(SUBTRACT);
    else SetBlendEquation(ADD);
    DrawQuad($q$);
  }
}
```

Implementing texture blending in this way, the arm example (with three quadrilaterals using 100 basis textures each) each can be rendered at 18Hz, whereas another implementation which blends textures in software but still uses OpenGL to perform texture mapping can render the same animation at only 2.8Hz on the same machine.

Warping The warping used in hardware texture mapping is designed for 3d models and (for correct perspective warping) requires a depth component for each vertex. The way we currently form our normalized quads there is no consistent 3D interpretation of the texture patches, so we use 2D-2D planar warps, and for these, the hardware performs only affine warping, which is sufficient in the case of the arm example. However, in the structure from motion case, since we do not have an exact model, our method uses planar homographies (Eq. 10) to preserve perspective texture projection . These are approximated by applying the affine warp to subdivided regions. Each quadrilateral is first subdivided in a uniform grid in texture space. These vertices are then projected to image space using the homography warping function, and each of the smaller quadrilaterals is rendered in hardware without depth variation. By increasing the resolution of the subdividing mesh, the accuracy of the approximation can be increased (see Figure 4).

4. Experimental results

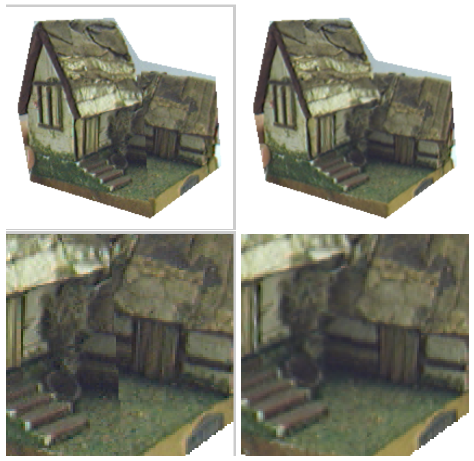

Figure 9: *Geometric errors on the house sequence. Top: Rendered images using static (left) and dynamic (right) textures respectively. Bottom: Detail showing the geometric errors.*

We have tested our method both qualitatively and quantitatively by capturing various scenes and objects and then reanimating new scenes and motions using dynamic texture rendering. Here we present the renderings of a toy house, a flower and the animation of non-rigid articulated motion from a human arm.

Many man-made environments are almost piece-wise planar. However, instead of making a detailed model of every

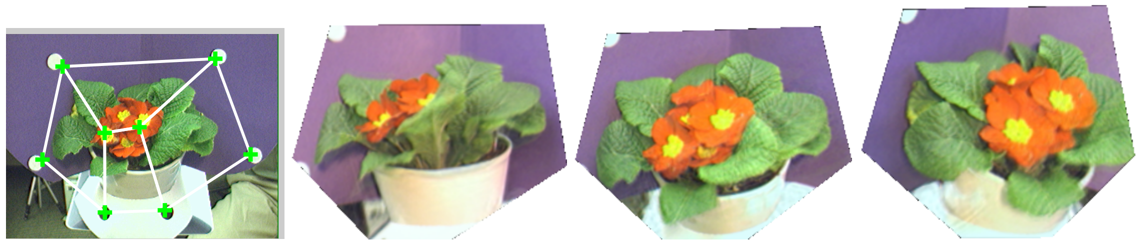

Figure 8: *Flower sequence: (left most) One of the original images and the outline of the quadrilateral patches*

surface, it is more convenient to model only the large geometric structure, e.g. the walls and roofs of houses, and avoid the complexity of the details, e.g. windows, doors, entry ways, trim, eaves and other trim. Figure 7 shows the rendering of a toy house in different poses. To capture the model a real time video feed was displayed and the user initialized SSD trackers by clicking on several house features. The house was turned to obtain several sample views. The motion range in one captured sequence is limited in our implementation since each marked tracking target has to remain visible. To obtain a large motion range two separate sequences were pieced together to generate Fig. 7. A total of 270 example frames were used to compute a texture basis of size 50.

The geometry used in the house sequence only coarsely captures the scene geometry. In conventional rendering these errors will be visually most evident as a shear at the junction of mesh elements, see Fig. 9 left. Compare to the one rendered using the dynamic texture (right), where the dynamic texture compensates for the depth inaccuracy of the mesh and aligns the texture across the seam.

Unlike man-made scenes, most natural environments cannot easily be decomposed into planar regions. To put our method to test, we captured a flower using a very simple geometry of only four quadrilaterals. This causes a significant residual variability in the texture images. A training sequence of 512 sample images from motions **x** with angular variation of $\mathbf{r} = [40, 40, 10]$ degrees around the camera *u*- *v*- and *z*-axis respectively. A texture basis of size 100 was estimated, and used to render the example sequences seen in Fig. 8. We also captured a movie (`flower.mpg`) that shows the reanimation of the flower.

As an example of a non-rigid object animation we captured the geometry of an arm from the tracked positions of the shoulder and hand using image-plane motions (see Section 2.3.2). We recorded a 512 sample images and estimate a texture basis of size 100. The arm motion was coded using the image position of the hand. Figure 11 shows some intermediate position reanimated using the dynamic texture algorithm. To test the performance of our algorithm we generate parallel renderings using static texture. To be consistent with the dynamic texturing algorithm we sampled texture from an equally spaced 100 images subset of the original sequence. Figure 10 illustrates the performance of the two algorithms for re-generating one of the initial positions and for a new position. We also created a movie (`arm.mpg`) that reanimates the arm in the two cases. Notice the geometric errors in the case of static texture algorithm that are corrected by the dynamic texture algorithm.

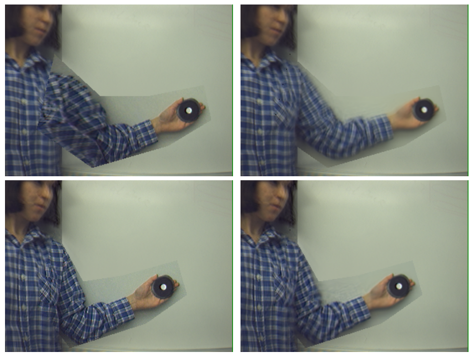

Figure 10: *Geometric errors on arm sequence. (top) Renderings of a new position using static and dynamic textures respectively. (bottom) Rerenderings for one frame from the training sequence using static and dynamic textures.*

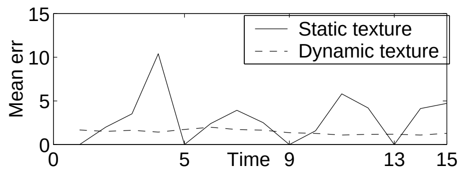

Figure 12: *Intensity pixel error in the rendered images (compared to original image)*

To quantify the geometric errors for our dynamic texturing algorithm compared to a classic texturing algorithm we took images of a pattern and then regenerate some of the original positions. We recorded differences in pixel intensities between the rendered images and original ones (Figure 12). Notice that the error was almost constant in the case of dynamic texture and very uneven in the case of static texture. For the static texture case we used frame 0,5,9,13 for sourcing the texture (consistent with using three texture basis vectors in the dynamic case) so is expected that the error drops to zero when reproducing these frames. The mean relative intensity error was 1.17% in the case of static texture and 0.56% in the case of dynamic texture. There are still some artifacts present in the reanimation mainly caused by the imprecise estimated geometric model. An improved solution would be to automatically estimate a non-rigid model from tracked points (similar to Bregler [1]).

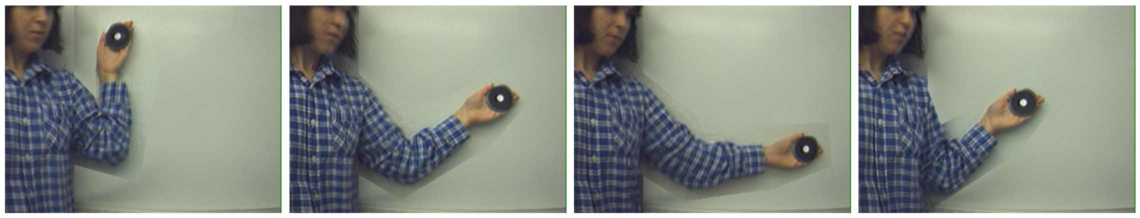

Figure 11: *Arm animation*

	Vertical jitter	Horizontal jitter
Static texture	1.15	0.98
Dynamic texture	0.52	0.71

Table 1: *Average pixel jitter*

For an animation there are global errors through the whole movie that are not visible in one frame but only in the motion impression from the succession of the frames. One important dynamic measurement is motion smoothness. When using static texture we source the texture from a subset of the original images ($k+1$ if k is the number of texture basis) so there is significant jumping when changing the texture source image. We tracked a point through a generated sequence for the pattern in the two cases and measure the smoothness of motion. Table 1 shows the average pixel jitter.

5. Discussion

We defined a *dynamic texture*, and showed how to use this to compensate for inaccuracies in a coarse geometric model. With a relaxed accuracy requirement, we show how to obtain a coarse scene model using a structure-from-motion algorithm on real-time video. Using the model, rectified texture patches are extracted, and the residual intensity variation in these is analyzed statistically to compute a spatial texture basis. In rendering and animation new images is generated by modulating the texture basis and warping the resulting, time varying texture onto the image projection of the geometric model under the new desired pose.

An advantage of our method is that all steps in modeling and re-rendering can be done with just an uncalibrated camera (web cam) and standard PC, instead of needing expensive 3D scanners. This also avoids the cumbersome registration needed to align a 3D model with camera images.

Current graphics cards have texture memory sizes designed for conventional static texturing. In our application we use a texture basis of 10-100 elements, which allows only one object of the type we shown to be rendered at a time. We plan to investigate ways of partitioning the texture basis so that only a subset is needed for each view, and adding a swapping strategy to move textures between graphic and main memory as predicted by the current motion being rendered. This we hope will extend our capability to capture and render for instance not only an arm, but a whole upper body of a human.

References

1. C. Bregler, A. Hertzmann, and H. Biermann. Recovering non-rigid 3D shape from image streams. In *IEEE Conference Computer Vision and Pattern Recognition (CVPR00)*, 2000.
2. C. Buehler, M. Bosse, S. Gortler, M. Cohen, and L. McMillan. Unstructured lumigraph rendering. In *Computer Graphics (SIGGRAPH'01)*, 2001.
3. P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from phtographs. In *Computer Graphics (SIGGRAPH'96)*, 1996.
4. O. D. Faugeras. *Three Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, Boston, 1993.
5. S. J. Gortler, R. Grzeszczuk, and R. Szeliski. The lumigraph. In *Computer Graphics (SIGGRAPH'96)*, pages 43–54, 1996.
6. G. D. Hager and P. N. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10):1025–1039, 1998.
7. Z. Hakura, J. Lengyel, and J. Snyder. Parameterized animation compression. In *Eurographics Workshop on Rendering*, 2000.
8. M. Jagersand. Image based view synthesis of articulated agents. In *Computer Vision and Pattern Recognition*, 1997.
9. K. Kutulakos and S. Seitz. A theory of shape by shape carving. *International Journal of Computer Vision*, 38:197–216, 2000.
10. M. Levoy and P. Hanrahan. Light field rendering. In *Computer Graphics (SIGGRAPH'96)*, pages 31–42, 1996.
11. T. Malzbender, D. Gelb, and H. Wolters. Polynomial texture maps. In *Computer Graphics (SIGGRAPH'01)*, 2001.
12. L. McMillan and G. Bishop. Plenoptic modeling: Am image-based rendering system. In *Computer Graphics (SIGGRAPH'95)*, pages 39–46, 1995.
13. C. J. Poelman and T. Kanade. A paraperspective factorization method for shape and motion recovery. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3):206–218, 1997.
14. M. Pollyfeys. *Tutorial on 3D Modeling from Images*. Lecture Notes, Dublin, Ireland (in conjunction with ECCV 2000), 2000.
15. A. Shashua. *Geometry and Photometry in 3D Visual Recognition*. PhD thesis, MIT, 1993.
16. I. Stamos and P. K. Allen. Integration of range and image sensing for photorealistic 3d modeling. In *ICRA*, 2000.
17. P. Sturm and B. Triggs. A factorization based algorithm for multi-image projective structure and motion. In *ECCV (2)*, pages 709–720, 1996.
18. R. Szeliski. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications*, pages 22–30, March 1996.
19. C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: A factorization method. *International Journal of Computer Vision*, 9:137–154, 1992.
20. D. Weinshall and C. Tomasi. Linear and incremental aquisition of invariant shape models from image sequences. In *Proc. of 4th Int. Conf. on Compute Vision*, pages 675–682, 1993.