Image-based Rendering using Hardware Accelerated Dynamic Textures

Keith Yerex, Dana Cobzas and Martin Jagersand
Computing Science, University of Alberta, Canada
www.cs.ualberta.ca/~{keith,dana,jag}

ABSTRACT

With recent improvements in consumer graphics hardware, image-based rendering in real-time is possible by modulating (blending) a large basis of transparent textures. We make efficient use of this by developing a two stage model, where a high quality rendering is achieved by combining an approximate geometric model with a time varying *dynamic texture* blended from the basis. The *dynamic texture* compensates for the inaccuracies in the approximate geometry by encoding the resulting texture intensity errors in a way similar to in mpeg movie compression, but here parameterizing the variability in pose instead of time, hence allowing the interpolation of arbitrary views. Additionally, we show how this model can be captured from uncalibrated images using an ieee1394 digital web-cam and real-time tracking. We show experiments of capturing and rendering everyday objects such as flowers and houses.

1. INTRODUCTION

A long standing goal in image-based modeling and rendering is to take a sequence of images from a scene (e.g. using a web cam or household camcorder), and construct a sufficient model for re-rendering any view of that scene. Implemented on a consumer PC this could allow ordinary users to capture real objects and scenes and insert into e.g games, home architecture and planning software, or send 3D "photos". This is unlike current methods, which in the case of image-based rendering (e.g. lumigraph or plenoptic function based [5, 8, 9]) require precisely calibrated cameras to sample and parametrize the ray set. Methods based on capturing 3D Euclidean models and texture images also require calibrated cameras[12, 3], and in the case of using 3D range sensing it is difficult to precisely register (align) the camera texture images with the captured 3D model[2].

Here we present an two level approach in which first real-time visual tracking is used to capture a non-Euclidean model. The model represents an approximation of the true scene structure, and is used to approximately stabilize texture patches during tracking. Second, using image statistics, a spatial basis is constructed which captures the residual intensity variation in the stabilized texture. Both the geometry and texture basis are parameterized in pose to enable rendering of new poses. In the rendering stage, given a new desired view, the geometric structure is reprojected into the new image, and a new texture is generated by modulating the texture basis with the pose of the new view. Finally new texture is warped onto the geometry giving the image of the new view. The advantage of the approach is that it decomposes the difficult problem of exactly capturing geometry and aligning it with texture images into two simpler simpler tasks where the strengths and weaknesses of each of the two subtasks complement each other. In [1] we show that in image-based rendering, methods based on a weak perspective camera and linear factorization into affine geometry and pose are more robust and suitable for uncalibrated video, than those based on a full perspective camera and projective geometry. However, for a real camera the linear factorization yields only an approximate model. In [7] we show how image variations due to small motions can be captured by a linear basis, and parameterized in pose. In this paper we combine the two techniques, so that we can capture, model and render scenes also under large motions.

In our implementation both the tracking and rendering runs at frame rate on a 1.4GHz consumer PC. In order to achieve this we show how to implement the texture modulation and warp in hardware accelerated OpenGL using transparent textures and texture blending.

2. THEORY

From a geometric view, IBR techniques relate the pixel-wise correspondence between sample images I_t and a desired views I. This can be formulated using a warp function w to relate $I_t(w) = I$. If $I_t \approx I$ 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, 8].

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 P and a set of motion parameters

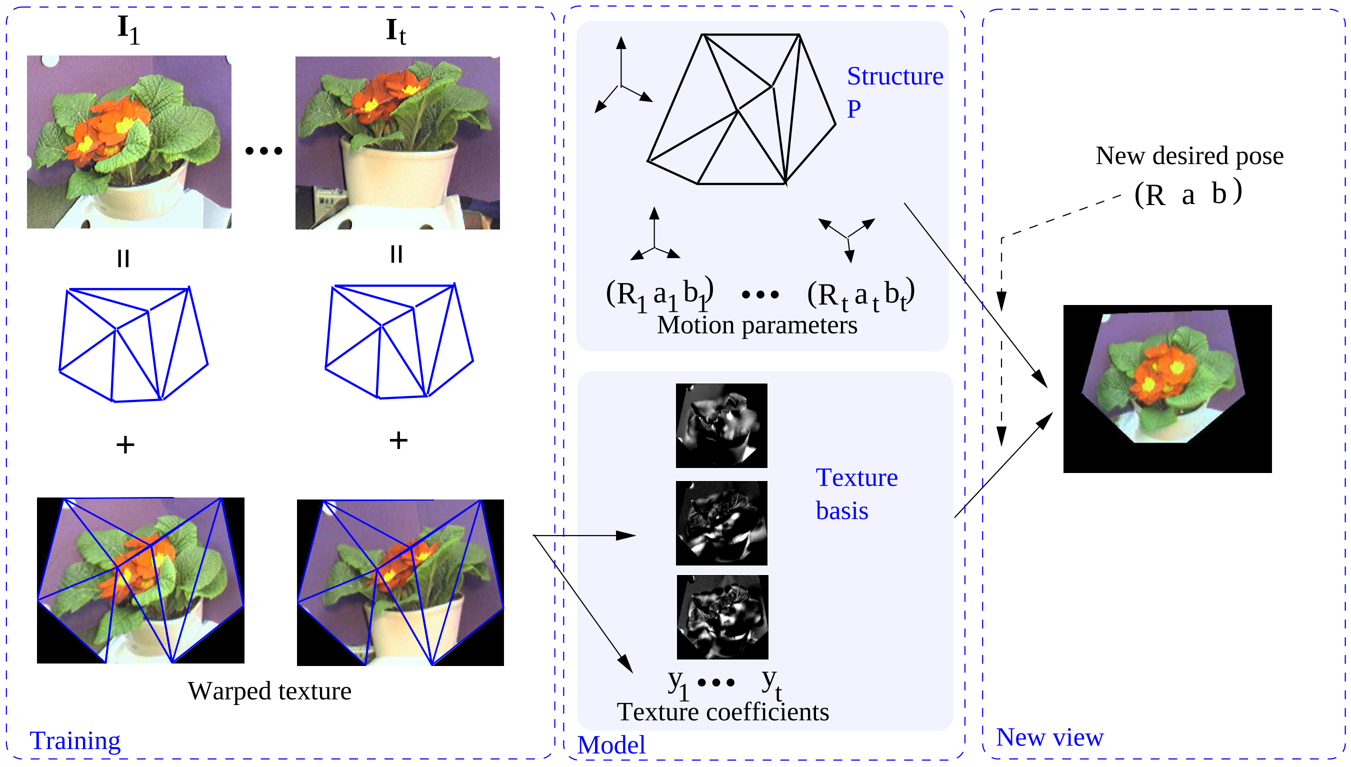

Figure 1: A sequence of training images $I_1 \cdots I_t$ is decomposed into geometric shape information and dynamic texture for a set of quadrilateral patches. The scene structure P and motion (r, s, a, b) is determined from the projection of the structure using a factorization algorithm. The dynamic texture is decomposed into its projection $\mathbf{y}$ on an estimated basis B. For a given desired position, a novel image is generated by warping new texture synthesized from the basis B on the projected structure.

$\mathbf{x}_t = (R_t, a_t, b_t)$ that uniquely characterize each frame is estimated from the tracked points using affine structure from motion (section 2.2). The reprojection of the structure given a set of motion parameters $\mathbf{x} = (R, a, b)$ is obtained by

$$\begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} = RP + \begin{bmatrix} a \\ b \end{bmatrix} \tag{1}$$

The projection of the estimated structure $[\mathbf{u}_t, \mathbf{v}_t]$ into the sample images is divided into Q triangular regions I_{qt} that are then warped to a standard shape I_{wqt} to generate a bigger texture I_{wt}.

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

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

$$I_{wt} = \sum_{q=1}^{Q} I_{wqt} \tag{4}$$

Using the algorithm described in section 2.1 we then compute a set of basis images B that capture the image variability caused by geometric approximations and illumination changes and the set of corresponding blending coefficients $\mathbf{y}_t$.

$$I_{wt} = B\mathbf{y}_t + \bar{I} \tag{5}$$

To generate a new view we first estimate the projection of the structure P 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 5). Finally the texture is warped to the projected structure (inverse of Equations 5,3,2). 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 parameter warps representing texture transforms, with depth, non-rigidity and lighting compensation, and then we show how to stably estimate this basis from actual image variability.

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.

$$I_w(\mathbf{p}) = I(w(\mathbf{p}), t) \tag{6}$$

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):

$$\begin{aligned}\Delta I = I(w + \Delta w) - I_w = I(f) + \tfrac{\partial}{\partial w} I(f)\Delta w - I_w = \\ \tfrac{\partial}{\partial w} I(w)\Delta w\end{aligned} \tag{7}$$

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.

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

$$\begin{bmatrix} u_w \\ v_w \end{bmatrix} = \mathcal{W}(\mathbf{p}, \mathbf{a}) = \begin{bmatrix} a_3 & a_4 \\ a_5 & a_6 \end{bmatrix} \mathbf{p} + \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \tag{8}$$

This is also the standard image-to-image warp supported in OpenGL. Now we can rewrite the image variability Eq. 7 resulting from variations in the six affine warp parameters as:

$$\Delta I_a = \sum_{i=1}^{6} \tfrac{\partial}{\partial a_i} I_w \Delta a_i = \left[\tfrac{\partial I}{\partial u}, \tfrac{\partial I}{\partial v}\right] \begin{bmatrix} \frac{\partial u}{\partial a_1} \cdots \frac{\partial u}{\partial a_6} \\ \frac{\partial v}{\partial a_1} \cdots \frac{\partial v}{\partial a_6} \end{bmatrix} \Delta[a_1 \ldots a_6]^T \tag{9}$$

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 six parameter variability in terms of spatial image derivatives:

$$\begin{aligned}\Delta \mathbf{I}_a = \left[\tfrac{\partial \mathbf{I}}{\partial u}, \tfrac{\partial \mathbf{I}}{\partial v}\right] \begin{bmatrix} 1 & 0 & *\mathbf{u} & 0 & *\mathbf{v} & 0 \\ 0 & 1 & 0 & *\mathbf{u} & 0 & *\mathbf{v} \end{bmatrix} [y_1, \ldots, y_6]^T = \\ [\mathbf{B}_1 \ldots \mathbf{B}_6][y_1, \ldots, y_6]^T = B_a \mathbf{y}_a\end{aligned} \tag{10}$$

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

Depth compensation In the above we assumed a planar patch. Real world patches will often not be planar, and this planarity will introduce a parallax error. The intensity variation due to the depth parallax can be written as a linear basis:

$$\Delta \mathbf{I}_d = [\mathbf{B}_{d1}, \mathbf{B}_{d2}][y_{d1}, y_{d2}]^T \tag{11}$$

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[11, 6]. For a general object, the illumination component can be approximated with a low dimensional basis .

$$\Delta \mathbf{I}_l = [\mathbf{B}_1 \ldots \mathbf{B}_3][y_1 \ldots y_3]^T = B_l \mathbf{y}_l \tag{12}$$

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:

$$\begin{aligned}\Delta \mathbf{I} = \Delta \mathbf{I}_s + \Delta \mathbf{I}_d + \Delta \mathbf{I}_n + \Delta \mathbf{I}_l + \Delta \mathbf{I}_e = \\ B_s \mathbf{y}_a + B_d \mathbf{y}_d + B_l \mathbf{y}_l + \Delta \mathbf{I}_e = \mathbf{B}\mathbf{y} + \Delta \mathbf{I}_e\end{aligned} \tag{13}$$

Approximations to $\mathbf{B}_a$ can be computed from image derivatives, while $B_d \mathbf{y}_d$ and $B_l \mathbf{y}_l$ require detailed geometric knowledge of the scene, camera and lighting. Note in Eq. 13 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. 13)

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

$$\Delta \mathbf{I} = \hat{B}\hat{\mathbf{y}}, \tag{14}$$

the same format as Eq. 13, 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, so in addition we estimate a second transform J between our pose description and $\hat{\mathbf{y}}$. In our application we represent one object using several texture patches, and estimate J between texture mixing coefficients $\mathbf{x}$.

For every training image $\mathbf{I}_t$ we have from the orthogonality of $\hat{V}$ that the corresponding texture mixing coefficients are the columns of $[\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_M] = \hat{V}^T$. From the factorization of geometric structure we also have the corresponding $\mathbf{x}_t$.

To estimate the texture mixing coefficients for intermediate poses, we first apply n-dimensional Delaunay triangulation over the sampled poses $\mathbf{x}_t$. Then given any new pose $\mathbf{x}$ we determine which simplex the new pose is contained in, and estimate the new texture mixing coefficients $\hat{\mathbf{y}}$ by linearly interpolating the mixing coefficients of the corner points of the containing simplex.

2.2 Geometric model

A structure-from-motion algorithm starts with a set of corresponding features (point, lines) in a sequence of images of a scene and recovers the coordinates of these features and the cameras poses relative to this representation under some viewing constraints (see Figure 2). 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 feature tracking errors. Assuming a more simplified model of the camera (weak perspective[4, 10]), 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 2: A general structure from motion algorithm extracts the structure and camera poses from a set of tracked points.

Here we have developed an extension of the Tomasi-Kanade factorization algorithm[13] for weak perspective camera projection model inspired by [14]. 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:

$$\begin{aligned} u_{ti} &= s_t \mathbf{i}_t^T \mathbf{P}_i + a_t \\ v_{ti} &= s_t \mathbf{j}_t^T \mathbf{P}_i + b_t \end{aligned} \tag{15}$$

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 Eq. 15 for multiple points (N) tracked in several frames (M)

$$W = RP + \mathbf{t}1 \tag{16}$$

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 [14].

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:

$$\hat{W} = RP \;\; where \;\; \hat{W} = W - \mathbf{t}1 \tag{17}$$

Following [13], in the absence of noise we have 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

$$\begin{aligned} \hat{R} &= O_1' \\ \hat{P} &= \Sigma' O_2' \end{aligned} \tag{18}$$

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 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:

$$\begin{aligned} R &= \hat{R} Q \\ P &= Q^{-1} \hat{P} \end{aligned} \tag{19}$$

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

$$\begin{aligned} \hat{\mathbf{i}}_t^T Q Q^T \hat{\mathbf{i}}_t &= \hat{\mathbf{j}}_t^T Q Q^T \hat{\mathbf{j}}_t \qquad && (= s_t^2) \\ \hat{\mathbf{i}}_t^T Q Q^T \hat{\mathbf{j}}_t &= 0 && t \in \{1..M\} \end{aligned} \tag{20}$$

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. 16 are unit vectors scaled by the factor s_t and the second equation constrain them to orthogonal vectors. This generalizes [13] 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

$$\mathbf{x}_t = [\mathbf{r}_t, s_t, a_t, b_t] \tag{21}$$

The geometric structure is represented by P and its reprojection given a new pose $\mathbf{x} = [\mathbf{r}, s, a, b]$ is estimated by

$$[\mathbf{u}, \mathbf{v}] = sR(\mathbf{r})P + \begin{bmatrix} a \\ b \end{bmatrix} \tag{22}$$

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

3. IMPLEMENTATION

3.1 Hardware Rendering

To render the *dynamic texture* we use the texture blending features available on most consumer 3D graphics cards. These graphics accelerators can blend textures very efficiently, however they are very restrictive in terms of the types textures that can be used, making it somewhat complicated to hardware accelerate this type of rendering.

Unsigned Basis The rendering hardware used is designed for textures containing positive values only, while the spatial basis, Equation 14 is a signed quantity. We rewrite this as a combination of two textures with only positive components:

$$I_w(t) = B^+\mathbf{y}(t) - B^-\mathbf{y}(t) + \bar{I}$$

Where B^+ contains only the positive elements from B (and 0 in the place of negative elements) and B^- contains the absolute values of all negative elements from B. When blending, some textures will be added, and others subtracted.

Quantization Graphics cards generally require textures to be represented as bytes in the range 0-255, and after each blending operation (addition or subtraction of a basis texture) values are clipped to this range. This can be problematic since neither our basis textures nor the intermediate values when combining basis textures are will have values within the required range. The only guarantee is that the final result will be within that range. We scale the basis textures and coefficients to fit within this range as follows:

$$\begin{aligned} \tilde{B}^+ &= 255B^+\zeta^{-1} \\ \tilde{B}^- &= 255B^-\zeta^{-1} \\ \tilde{\mathbf{y}} &= 255^{-1}\zeta\mathbf{y} \end{aligned}$$

Where ζ is a diagonal matrix of the maximum absolute values from the columns of B. ($\zeta = \mathrm{d}iag(\max|B|)$)

Now $\hat{B}^+$ and $\hat{B}^-$ are both in the range 0-255, and can be used as textures in hardware. The problem regarding overflow in intermediate blending stages cannot be completely solved, but by drawing the mean image first, and then alternately adding and subtracting scaled eigenvectors, overflow is avoided in most cases.

Rendering of each frame is performed as in the following pseudo-code.

```
// draw the mean
BindTexture($\bar{I}$);
DrawTriangles();

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

  BindTexture($\tilde{B}_i^+$);
  if($\tilde{\mathbf{y}}_i(t) > 0$) SetBlendEquation(ADD);
  else SetBlendEquation(SUBTRACT);
  DrawTriangles();

  BindTexture($\tilde{B}_i^-$);
  if($\tilde{\mathbf{y}}_i(t) > 0$) SetBlendEquation(SUBTRACT);
  else SetBlendEquation(ADD);
  DrawTriangles();
}
```

3.2 Algorithm

Training data We capture an image sequence, and tracked feature locations. We use XVision [6] to set up a video pipeline and use SSD trackers to track features on-line. The feature positions and 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 geometric model based on the tracked points sampled in the training step using the technique described in section 2.2.

Dynamic Texture

1. Warp each frame $I(t)$ to a standard shape $I_w(t)$ based on tracked positions. (Equations 2,3,4). The standard shape is chosen to be the average positions of the tracked points scaled to fit in a square region.

2. Form a zero mean sample, and perform the PCA as described in section 2.1. Keep the first k basis vectors B, and the corresponding coefficients for each frame in the training sequence $\mathbf{y}(t)$.

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. Estimate texture blending coefficients $\mathbf{y}$ by interpolating the coefficients of the nearest neighbors from the coefficients, and poses from the training data.

3. Compute the new textures in the standard shape using Equation 5 and rewarp the textures onto the calculated geometry. (These two operations are performed simultaneously in hardware as described in section 3.1)

Figure 3: House sequence animated at different viewpoints

4. EXPERIMENTAL RESULTS

We have tested our method both qualitatively and qualitatively 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 and a flower.

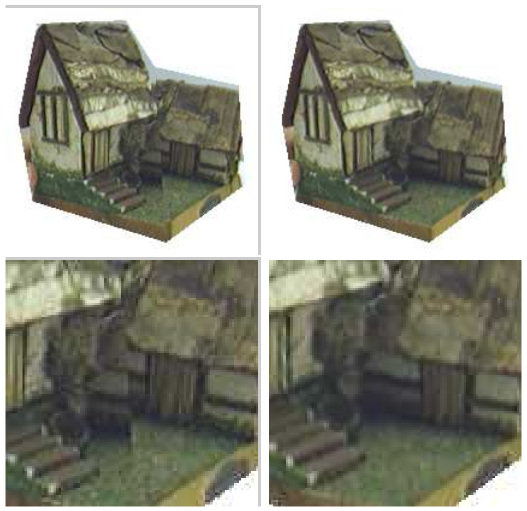

Figure 4: Geometric errors on the house sequence. Top: Rendered images using static (left) and dynamic (right) textures respectively. Bottom: Detail showing the geometric errors.

Many man-made environments are almost piece-wise planar. However, instead of making a detailed model of every 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 etc. Figure 3 shows the rendering of a toy house in different poses. 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. 3. 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. 4 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 8 triangles. This causes a significant residual variability in the texture images. A training sequence of 512 sample images from motions $\mathbf{x}$ 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 was used to render the example sequences seen in Fig. 5.

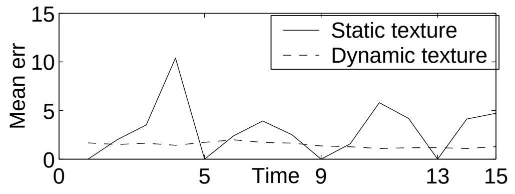

Figure 6: Intensity pixel error in the rendered images (compared to original image)

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 6). 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.

	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 mo-

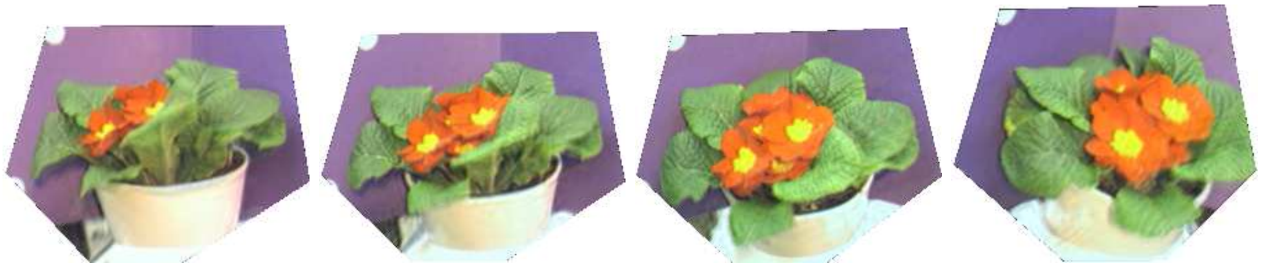

Figure 5: Flower sequence animated at different viewpoints.

tion 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 showed how to capture object models from video and render new views using a new type of image-based modeling where a coarse geometric model is captured from images, and a time-varying *dynamic texture* is overlaid to compensate for errors in the coarse geometry approximation. Our technique obliviates the need for expensive range sensors and calibrated setups, and instead lets users capture and model objects and scenes using inexpensive consumer web or video cameras with a standard PC.

To enable real-time rendering we take advantage of recent advances in consumer grade graphics cards which allow blending of several transparent textures. Two current HW limitations we face are 1) the unsigned byte arithmetic approximation in the texture blending occasionally causes overflows in intermediate values. 2) The graphics card memory sizes have not yet grown significantly enough to enable extensive use of several transparent texture layers, so currently we are limited to relatively small models in low resolution mode.

To more efficiently make use of current texture memory sizes we plan to investigate adaptive texture resolutions, so that the large spatial basis needed for high resolution rendering is used for frontal surfaces, and the number of basis vectors is reduced for triangles at oblique angles to the camera. A more trivial improvement is to switch from RGB color space to a more compact color representation such as YUV. Currently the tracking is purely bottom up. To improve stability we plan to integrate it with the model building to provide top-down feedback to maintain global model consistency and with the variability estimation to adapt the tracking templates over time.

6. REFERENCES

[1] D. Cobzas and M. Jagersand. A comparison of non-euclidean image-based rendering. In *Proceedings of Graphics Interface*, 2001.

[2] D. Cobzas, H. Zhang, and M. Jagersand. A comparative analysis of geometric and image-based 3d-2d registration algorithms. In *ICRA*, 2002.

[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] M. Jagersand. Image based view synthesis of articulated agents. In *Computer Vision and Pattern Recognition*, 1997.

[8] M. Levoy and P. Hanrahan. Light field rendering. In *Computer Graphics (SIGGRAPH'96)*, pages 31–42, 1996.

[9] L. McMillan and G. Bishop. Plenoptic modeling: Am image-based rendering system. In *Computer Graphics (SIGGRAPH'95)*, pages 39–46, 1995.

[10] M. Pollyfeys. *Tutorial on 3D Modeling from Images*. Lecture Nores, Dublin, Ireland (in conjunction with ECCV 2000), 2000.

[11] A. Shashua. *Geometry and Photometry in 3D Visual Recognition*. PhD thesis, MIT, 1993.

[12] I. Stamos and P. K. Allen. Integration of range and image sensing for photorealistic 3d modeling. In *ICRA*, 2000.

[13] 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.

[14] 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.