Projective Registration with Difference Decomposition

Michael Gleicher
Apple Research Laboratories
1 Infinite Loop M/S 301-3J
Cupertino, CA 95014
gleicher@cs.cmu.edu

ABSTRACT

Current methods for registering image regions perform well for simple transformations or the large image regions. In this paper, we present a new method that is better able to handle small image regions as they deform with non-linear transformations. We introduce *difference decompositon*, a novel approach to solving the registration problem. The method is a generalization of previous methods and can better handle non-linear transforms. Although the methods are general, we focus on projective transformations and introduce *piecewise-projective* transformations for modeling the motions of non-planar objects. We conclude with examples from our prototype implementation.

1. Introduction

Whether under the guise of tracking or registration, determining the transformation between regions of two images is an important problem in computer vision. In addition to vision applications, such as recognition and surveillance, it has a number of applications for computer graphics, such as image mosaicing, model-based compression, and video effects. In such applications an accurate transformation is required; the use of simplified models or inaccurate parameters may lead to undesirable visual artifacts.

Research in computer vision has provided methods for region registration. State of the art methods apply numerical optimization to minimize the error between the transformed target and reference images. Such an approach is called a sum-of-squared-difference approach (SSD) because an L_2 norm is typically used to measure the error. Variants of this method have been shown to track simple (affine or translational) deformations quickly [5] and more complex projective [13] [8] or piecewise bilinear [14] transformations over large image regions. SSD methods replace the image registration problem with non-linear minimization, allowing the use of standard algorithms.

In this paper, we present a tracker that is capable of following small image regions as they deform with complex transformations. Because using a black box solver provides little opportunity to employ our problem specific insights, we derive an optimzation-based algorithm in the specific context of image registration. This leads to a more general derivation of the SSD algorithm. By exploiting the generality, we can create a method that better suits our demands. While our methods are general, we focus on projective image transformations since they have particularly useful applications. We also introduce the concept of a piecewise-projective deformation that can better approximate cases where a single projective patch cannot.

The rest of this paper is organized as follows. We first review the existing SSD approaches. We then introduce our "difference decomposition" method in Section 3. Section 4 discusses specific details for tracking projective deformations, and introduces the piecewise projective transform. We conclude by presenting experimental results.

2. SSD Trackers

Given a reference image I_0 and a target image I_1, the registration problem is to compute an image transformation Υ that transforms the target image such that it matches the base image in a specified region R. In practice, we pick a parameterized model $\Upsilon_{\mathbf{q}}$ and solve for the vector of parameters $\mathbf{q}$.

Often, our primary interest is determining the geometric transformation between the images. A geometric transformation is defined as a mapping $\mathcal{R}^2 \rightarrow \mathcal{R}^2$ between coordinates. We typically include a transformation to account for changes in intensity as well. The most commonly used transformations are translation and affine deformation with linear offset and scaling of intensity.

We typically do not search for an exact match. Rather, we try to find the best possible match, typically with an L_2 (sum-of-squares) norm. SSD minimizes

$$e(\mathbf{q}) = \sum_{x,y \in R} \left(I_0(x,y) - (\Upsilon_{\mathbf{q}}(I_1))(x,y) \right)^2 . \qquad (1)$$

Solving this optimization problem for various motion

models has been a mainstay of computer vision. Historically brute force search was used [1]. This was inefficient and impractical for higher dimensional parameter spaces. More recent methods treat Equation 1 as a non-linear optimization problem. This approach was introduced by [7] who used a Newton-type iteration. Many authors have subsequently improved on this work. For example, [11] tracked affine deformations and [5] reformulated the tracker for real-time performance and reduced sensitivity to changes in illumination. [12] connected images texture, numerical issues, and tracker performance. A more sophisticated (Levenberg-Marquardt) solver was used to track more complex transforms including multi-bilinear [14] and projective [13] patches.

Unfortunately, casting the problem as non-linear minimization trades one hard problem for another. There is no guaranteed, direct method. Although the basic methods are centuries old and the algorithms highly evolved, they still rely on heuristics. Most algorithms, like those used in SSD, iteratively refine an estimate of the solution. At each step, the function being minimized is approximated by a simpler form with a known solution. For a practical introduction to numerical optimization, see [10] or [3].

2.1. Projective Transformations

Motion for tracking is rarely generated by a simple 2D image transformation. In certain special cases, however, real motions are be modeled by relatively simple image transforms. A projective transformation exactly models motions generated either by a camera rotating about its eyepoint, or if the imaged object is planar [13]. The projective transform in 2D is an 8 parameter transform:

$$x'(x,y) = \frac{q_1 x + q_2 y + q_3}{q_7 x + q_8 y + 1}, y'(x,y) = \frac{q_4 x + q_5 y + q_6}{q_7 x + q_8 y + 1}. \quad (2)$$

Recovering the projective transformation is particularly important for applications that resynthesize the images, such as image mosaicing [6][13][2]. Even in cases where the transformation doesn't exactly describe the motion, the projective transformation better approximates simpler transformations since it accounts for depth effects [8][6]. The projective group subsumes the more commonly used affine and translational cases, so a projective transformation always provides at least as good an approximation.

The obvious question is: why aren't projective transformations used more often? The answer seems to be that they are considerably harder to use in a tracker. Not only do they provide a larger, less constrained search space than the affine case, but the transformation is highly non-linear and the variables have widely varying sensitivities. Since affine or quadratic models are often close enough for small deformations, the additional effort is often not worthwhile.

Despite the difficulty, systems have been developed to track projective deformations. [13] described a system that enhances the standard SSD methods with the more sophisticated Levenberg-Marquardt optimization algorithm. [8] provided an algorithm that fit a succession of simplified transformations. Both demonstrate impressive results, but require large image regions because they rely on decimated images to sense larger interframe changes.

3. A Difference Decomposition Approach

Let us consider minimizing Equation 1 using an iterative approach. For a given step of the iterative optimization process, we have an estimate for the solution that we will call $\mathbf{q}$. When the transformation is applied to I_1 to give $I_1' = \Upsilon_{\mathbf{q}}(I_1)$, we are left with an image which still differs too much from I_0, the error $D = I_0 - I_1'$ has too great an L_2 norm. We must determine a transformation $\Delta\mathbf{q}$ that transforms I_1' into something that matches I_0. In a traditional numerical minimizer, our next estimate of the solution will be $\mathbf{q}_1 = \mathbf{q} + \Delta\mathbf{q}$. However, if we had really computed $\Delta\mathbf{q}$ as a transformation, we would expect the answer to be the composition of the two transforms. A general optimization procedure approximates this composition by addition. We can remove this approximation by using $\Upsilon_{\mathbf{q}_1} = \Upsilon_{\Delta\mathbf{q}} \circ \Upsilon_{\mathbf{q}}$, either by changing the update rule (for example, using matrix multiplication if the transform is an affine deformation matrix), or by picking a parameterization for which composition is addition (for example the parameterization for projective regions discussed in Section 4.1).

We now must tackle the problem of computing $\Delta\mathbf{q}$ that minimizes

$$e(\Delta\mathbf{q}) = \sum_{x,y \in R} \frac{1}{2}(I_0(x,y) - (\Upsilon_{\Delta\mathbf{q}}(I_1'))(x,y))^2. \quad (3)$$

For notational convenience, we define $E(\Delta\mathbf{q}) = I_0 - \Upsilon_{\Delta\mathbf{q}}(I_1')$, so $e(\Delta\mathbf{q}) = 1/2E \cdot E$.

A naive approach to minimize Equation 3 is to exhaustively try the possibilities, sampling $\mathcal{R}^n$.. When n is large, for example $n = 8$ for a projective deformation, any form of adequate sampling is impractical.

For now, let us ignore the impracticality and suppose there is some set of vectors $\{\mathbf{b}_1 \ldots \mathbf{b}_m\}$ that represent a sampling of the parameter space. It is sufficient to try all the sampling vectors to find $\Delta\mathbf{q}$. The obvious implementation would plug each $\mathbf{b}_i$ in for $\Delta\mathbf{q}$, compute the difference image for each one, and pick the one with the smallest norm. Rather than transform I_1' each time, we could equivalently transform I_0 by the inverse transform for each $\mathbf{b}_i$. What this does is create an image of what we would expect to see for I_1' if $\mathbf{b}_i$ was the correct answer for $\Delta\mathbf{q}$. The expectation images depend only on the base image and sample vectors and are independent of $\mathbf{q}$, I_1 or I_1', therefore they can be precomputed.

We can carry this idea a step further. For each basis vector, we can create a "difference template"

$$B_i = I_0 - \Upsilon_{\mathbf{b}_i}^{-1}(I_0). \tag{4}$$

This image shows the difference we would expect to see if $\mathbf{b}_i$ were the correct answer. To find the $\mathbf{b}_i$ that the best estimates $\Delta\mathbf{q}$, we find the B_i that best matches D.

3.1. Linearization

We can only try a (small) finite number of vectors. To get an exact answer for $\Delta\mathbf{q}$, we must be lucky in our choice for $\mathbf{b}_i$. In general, the choice that has the smallest error may not be the $\Delta\mathbf{q}$ that is closest to the exact right answer. To avoid exhaustive search we must make assuptions about the problem. As in most optimization problems, we make the assumption that the error is a "somewhat smooth" function of its parameters ($\Delta\mathbf{q}$). This means that the error is a measure of distance to the solution, so that decreasing the error is a way to move towards the solution. Each iteration can reduce the error, not necessarily elimate it.

The smoothness assumption implies that the function's higher order derivatives are small. Therefore, the error function is reasonably approximated by a low order polynomial. Many optimization algorithms (including those used in SSD) approximate the function with a quadratic. To obtain a quadratic approximation for e, we use a linear approximation for E, $E(\Delta\mathbf{q}) \approx D + \mathbf{K}\Delta\mathbf{q}$, where $\mathbf{K}$ is some matrix. The extremum of e can be computed by solving the linear system.

Linearity means that the errors scale and add with the inputs. So, with the linearity assumption, if $D = kB_i$, for some scalar k, we could estimate $\Delta\mathbf{q} = k\mathbf{b}_i$. Similarly, if $D = B_i + B_j$, then we could estimate $\Delta\mathbf{q} = \mathbf{b_i} + \mathbf{b_j}$. Or, combining scaling and adding, if $D = k_1B_i + k_2B_j$, then $\Delta\mathbf{q} = k_i\mathbf{b}_i + k_j\mathbf{b_j}$. This means that if we can find partial answers then we can add them up. It suggests that if we could decompose the difference image into a linear combination of the difference templates then the solution for $\Delta\mathbf{q}$ would be a linear combination of the corresponding basis vectors. Note that we never use the linearized error function (e.g. $\mathbf{K}$).

This "difference decomposition" is computed by considering the images as long vectors. The linear combination vector $\mathbf{k}$ is defined by $D = \mathbf{B}^T\mathbf{k}$, where $\mathbf{B}$ is the matrix with each row containing a difference template. Since there might not be an exact solution, we solve this by minimizing the squared error — squaring the equation, and solving for the vanishing of the gradient

$$\begin{aligned} \frac{\partial \frac{1}{2}(D - \mathbf{B}^T\mathbf{k})^2}{\partial \mathbf{k}} &= \frac{\partial(\frac{1}{2}D^2 - D\mathbf{B}^T\mathbf{k} + \frac{1}{2}\mathbf{k}\mathbf{B}\mathbf{B}^\mathbf{T}\mathbf{k})}{\partial \mathbf{k}} = 0 \\ -D\mathbf{B}^T + \mathbf{B}\mathbf{B}^\mathbf{T}\mathbf{k} &= \mathbf{0}. \end{aligned} \tag{5}$$

Equation 5 gives a method for estimating $\Delta\mathbf{q}$ by "decomposing the difference" between I_0 and I_1'. We first compute the how the difference D is decomposed into a linear combination of the difference templates by solving

$$D\mathbf{B}^T = \mathbf{B}\mathbf{B}^T\mathbf{k} \tag{6}$$

for $\mathbf{k}$, and then compute

$$\Delta\mathbf{q} = \mathbf{N}^\mathbf{T}\mathbf{k}, \tag{7}$$

where $\mathbf{N}$ is a matrix where each row has the corresponding basis vector.

Because of linearization, we do not need to be thorough with our basis set. The algorithm will find an estimate for $\Delta\mathbf{q}$ based on considering several nearby samples.

3.2. Generalizing the Traditional SSD

The derivation of the previous section basically provided a different, more general, derivation of standard SSD methods (in particular, the form described by [5]). A unit sample vector provides a first finite difference approximation for the derivative of the base image with respect to a variable. If we use the set of unit vectors for the samples, the difference templates B_i become the derivative images, and Equation 6 (Equation 7 isn't needed since $\mathbf{N}$ is the identity) is the update formula for SSD (see [5]).

Because the error function is non-linear, the first finite difference is unlikely to be a good approximation for the derivatives. To get a better value for the derivatives, implementation can use filtering to better estimate the spatial derivatives in the image[15], and via the chain rule use these to compute the derivative with respect to the transform variables. The difference decomposition approach compensates for this advantage by allowing the use of an arbitrary set of sampling vectors. By using vectors of varying lengths, we can model the non-linearities of the error further away from the origin.

3.3. When Linearization Breaks Down

If the error function were linear the unit sampling vectors would be sufficient. In fact, any set of vectors that spanned parameter space would be sufficient, although we might want to change coordinates to have better conditioning. Additional sampling vectors would be redundant.

Consider the one dimensional case. If $e(\Delta q)$ is quadratic, we find the minimum by solving the linear equation for its derivative equal to zero. Assuming that we know $e'(0)$, either the slope of the line, or the value of any sample along the line will tell us the solution. The SSD approach is the former approach while our approach is the latter. Any sample will give the same answer. If e' is not linear, however, using different samples will provide different results.

We could use multiple samples by fitting a higher-order polynomial than a line. This approach is difficult to gener-

alize in multiple dimensions and is generally not used because of stability problems. A different approach would be to find the solution suggested by each sample and average among them. A third variant would use the sample whose value was closest to zero, assuming that it best predicts the function is zero.

Equation 6 is effectively a combination of the second and third approaches. Solving the linear system provides us with a weighted average of the samples, with the weights dynamically chosen based on the proximity of the sample value to zero. If the sample were an exact match, then it is used completely.

3.4. Multiplier Damping

We now consider a potential problem: what if some sample vectors are redundant (or nearly so)? This means that there is insufficient texture to distinguish their effects on the image, and leads to a singular (or ill-conditioned) matrix in Equation 6[12]. Without any additional information, an algorithm has no good reason to pick one choice over the other. In practice, the algorithm will choose a bad one imposed by the realities of computational linear algebra. We therefore add some additional information.

Our addition is the observation that large values for the multipliers ($\mathbf{k}$) are bad, so that minimizing the magnitude of the multiplier vector is good. This approach is called a multiplier penalty or damped multiplier approach, and is discussed in [4] [9]. The same method appears in a different guise in the Levenberg-Marquardt algorithm [10]. We add an additional term into Equation 5, giving $m = \frac{1}{2}(D - \mathbf{B}^T\mathbf{k})^2 + \frac{1}{2}\epsilon\mathbf{k}\cdot\mathbf{k}$, where ϵ is a small constant. Solving for the gradient equal to zero gives a variant of Equation 6,

$$D\mathbf{B}^T = (\mathbf{B}\mathbf{B}^T + \epsilon\mathbf{I})\mathbf{k}. \tag{8}$$

Multiplier damping means that if two templates give equally good matches, their contributions are equally weighted. It also adds stability in cases where there is no good match as it discourages the algorithm from explaining things as a large positive plus a large negative weight.

3.5. Hierarchical Solution

The size of the estimate that we can reliably make for $\Delta\mathbf{q}$ is limited by how well E is approximated by a linear function. If the image is smoother, then this approximation will be valid over a larger range. To exploit this, registration algorithms usually work hierarchically: first operating on a decimated version of the image, then successively solving on less blurred versions. Such a strategy allows larger changes to be sensed in the coarse image, and precision to be added with the finer images.

We employ a coarse-to-fine strategy in our approach as well. At each level, we can use a different set of sampling vectors. At coarser levels, the sampling vectors are chosen that require less image texture, albeit at the cost of finding less accurate transformations. The accuracy can be resolved at the finer levels.

While the hierarchical process extends the range of SSD trackers, the amount of this extension is limited. First, there is a limit to how many levels can be used for tracking because small regions quickly become too small to use. Even in larger images, the texture required for unique matches is quickly blurred away. Also, the smoothing operation may remove non-linearities from the image, but not the non-linear transform. With our approach, we can use both longer sampling vectors and hierarchical solution to extend the range.

3.6. Algorithmic Control

Until this point, we have only focussed on the problem of determining one step of the iterative process. We now examine how this fits into the actual problem. For a tracking problem, we must find the transformation between the base image and each target frame. With our approach, we always compare the new image with the reference frame, although we could reset the reference frame after each match.

With a fixed reference frame enables significant preprocessing[5]. The difference templates can be generated for all levels of the pyramid and the matrices for solving can be formed and factored. Unfortunately, this preculudes dynamic adjustment of ϵ. Instead, we pick a few fixed values for ϵ, compute the matrices for each during the precomputation phase, and use the one that give us the best result for each iteration of the algorithm. Another downside of the preprocessing approach is that we cannot dynamically change the region size, for example, when the target region crosses an edge of the frame.

For each new frame, we must first determine an initial guess for the solution. To date, we have used the result from the previous frame, however some form of prediction would enhance performance when a motion model applies. The optimization process is then executed at each level of the pyramid, iterating the steps until no progress is made. The step process computes $\Delta\mathbf{q}$ via Equation 8 for several values of ϵ, computes the residual error for each of these guesses, and chooses the best one.

3.7. Tuneable Parameters

We would like to use any knowledge we have about the motion being tracked to tune the tracker for a particular task. Like the traditional SSD, our algorithm allows tuning through the choice of the motion model and the process by which initial estimates for parameters are generated. Our algorithm adds additional flexibility by permitting the design of the sets of sampling vectors (plural because a different set is used for each level of the hierarchy) and the non-uniform damping parameters associated with them.

The difference decomposition approach requires the design of the sample vector sets. In practice this is not a problem. It is easy to choose a standard basis that gives at least as good performance as an SSD approach by including the same samples that would be used by a good implementation of the image gradient computations. Including the unit vectors and copies scaled by small integers (we usually include $\pm 1, \pm 2, \pm 3$) insures that the sample vectors span the parameter space. Additional vectors can be added to this basic set to improve tracker performance, albeit at the cost of increased computation per step and potential conditioning problems.

If we have some knowledge about the expected motions, we can tune our sample vectors accordingly. For example, if we know that horizontal motions are most common, we can add additional vectors to better sample these likely regions of parameter space.

4. Tracking Projective Regions

The formulation of the tracker in the previous section is intentionally non-specific about the type of transformation. There are very few restrictions on the types of transformations that we can use, for example we can use a non-differentiable model. We have implemented and tested translational, affine, bilinear, multi-bilinear, projective and piecewise-projective transformations. Since projective transformations are the most useful for our applications, and where previous methods have not provided sufficient performance, we focus on them here.

The non-linearity of projective transforms is one reason that standard methods do not handle them well, typically functioning only over very small changes between images. Our approach offers a number of enhancements to improve this. First, we explicitly handle the non-linearity in composing steps. Secondly, the use of larger sample vectors can extend the search range in a way that takes the non-linearity of the transform into account, not just the non-linearity of the image. Third, we can design our sample vector sets to better to focus our search in the areas of $\mathcal{R}^8$ in which realistic motions are more likely to appear. Fourth, because we do not need to compute analytic derivatives of the transformation, we can use an alternative parameterization that addresses a number of issues.

4.1. Parameterizing Projective Transforms

The most obvious implementation of a projective tracker would be to perform the minimization on the 8 parameters of Equation 2. Unfortunately, this leads to equations that are difficult to solve. Not only are the equations non-linear, but the sensitivities of the answer to the parameters vary widely. For example, a change in $\mathbf{q}_3$ will make less of a change in the answer than an equivalent change in $\mathbf{q}_7$. One way to address these differences is parameter scaling, as discussed by [4].

Rather than represent the projective transformation by the eight coefficients of Equation 2, we instead use an alternate representation and perform search over these new parameters. The parameterization that we use stores the positional offsets for the four corners of the bounding rectangle of the initial region. Transformation between this representation and the canonical parameters is computed by solving a system of linear equations [6]. This parameterization has a number of advantages:

- The parameters have uniform sensitivity and are less interrelated than the canonical parameters;
- The parameters have intuitive meanings (which facilitates debugging);
- The composition is addition;
- Common simple transforms can be represented by fixed parameter vectors (for example, a one pixel translation is always the same vector);
- Regions can be made to have abutting edges simply by sharing parameters;
- Inversion is negation;
- Users can give hints by pulling on patch corners;

Although the representation is similar to that of bilinear patches, we do compute the proper projective mapping.

4.2. Piecewise Projective Transforms

The projective transform is useful because it exactly models an important special case. The more general case (for the motion of curved surfaces) is unlikely to have a compact solution, and must be approximated. Computer graphics traditionally approximates curved surfaces with sets of small polygons. For tracking, a similar approximation would use projective patches that are small enough that the surface in each region is sufficiently flat to be well approximated by the transformation. This poses two additional challenges: the patches connect properly; and smaller patches are less likely to have sufficient texture for the projective tracking. The answer to these is to not treat the patches independently, creating a connected mesh. Enforcing connection between patch corners is sufficient to create connectivity between projective patch edges, and the constraints between the patches effectively allow them to share information. These constraints can be implemented by having patches share corners.

By parameterizing the projective patches as in the previous section, creating connected grids is simple. They are akin to the bilinear patches of [14], with the exception that within each patch, a proper projective transformation is computed and used. The piecewise projective patch offers a number of advantages. Foremost, it better approximates real surfaces because it accounts for depth effects, allowing larger piece sizes to be used (see the examples). Secondly, because a projective transformation is defined by any

pair of quadralaterals, any tiling of the patch with quadrilaterals can be used, not just grids. Piece edges could be placed along edges or other discontinuities. Third, texture-map methods (including hardware support) can be used for rendering. Fourth, the projective transformation is easily invertable.

5. Experiments

Our prototype implementation uses Quicktime movies, Quicktime VR object movies, and sets of still images as content. The video examples were captured using a consumer camcorder played into the built-in digitizer of a Power Macintosh 8500 and recorded using standard compression. Our implementation provides the options of linear or bicubic interpolation for the image warps (albeit with point sampling) and uses a linear intensity adjustment (offset and scale) for pixel values. We emphasize that we have not done any tuning to the algorithm for these examples. All examples use a 3 level pyramid with our default basis vector set consisiting of scaled unit basis vectors and translations.

Our implementation of the previous method [13] does not succeed on these examples. This is more a statement about our implementation than of the methods.

5.1. Projective Distortions

Figure 1 shows a projective test case: a 40 pixel square region is tracked as the camera moves. The tracker stays on target despite the extreme perspective and lighting changes. In the 30fps sequence, we see corners move more than 7 pixels along one axis between pairs of frames. At 15fps, the largest change (approximately 18 pixels) loses the tracker, which is not surprising since this is at the limit of the range of the basis set (4 pixels, 3 levels up the pyramid $= 4 * 2^{(3-1)} = 16$ pixels).

Figure 2 is taken from a Quicktime VR object movie, a sequence of still frames. The steps between frames are larger than in most motion examples. Known rotations are an excellent candidate for a tuned basis vector set.

5.2. Non-Projective Cases

To demonstrate the piecewise projective patch, we use 2 types of examples where a projective transformation does not model the motion: tracking rigid curved objects (Figure 3), and flexible objects (Figure 4).

In a reconstruction scenario (Figure 5 and 6), we see the advantages of smaller patch sizes. In our examples, sampling and illumination artifacts make the boundaries of the patch obvious. Numerically, the L^2 error decreases with patch size. Empirically, the piecewise projective patches have less error than comparably sized bilinear patches.

6. Conclusions

In this paper, we have presented difference decomposition, a method for finding the transformation between regions of two images. By examining the numerical optimization process, we have created an algorithm that is better able to handle non-linearities in the image transformation than conventional approaches. The algorithm can track smaller image regions over larger interframe differences for non-linear transformations. It also provides more opportunities for tuning for specific classes of motion. We pay particular attention to projective transformations, and introduced a piecewise projective transformation that is able to better approximate the motion of curved surfaces. We have demonstrated our system on a variety of examples.

In our view, this paper makes two contributions. First, it presents the difference decomposition approach that potentially provides better performance than previous tracking algorithms. Second, it introduces the piecewise projective patch, offering advantages over the previous piecewise bilinear patches.

References

[1] P. Burt, C. Yen, and X. Xu. Local correlation measures for motion analysis, a comparitive study. In *IEEE Conf. of Pattern Recognition and Image Proc.*, pages 269–274, 1982.

[2] S. E. Chen. Quicktime VR – an image-based approach to virtual environment navigation. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 29–38, August 1995.

[3] P. Gill, W. Murray, and M. Wright. *Practical Optimization*. Academic Press, New York, NY, 1981.

[4] M. Gleicher. *A Differential Approach to Graphical Interaction*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1994.

[5] G. Hager and P. Belhumeur. Real-time tracking of image regions with changes in geometry and illumination. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 403–410, 1996.

[6] P. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, U. C. Berkeley, June 1989.

[7] B. Lucas and T. Kanade. An iterative image registration technique with application to stereo vision. In *Proceedings IJCAI-81*, pages 674–679, 1981.

[8] S. Mann and R. Picard. Virtual bellows: Constructing high quality stills from video. In *IEEE Conference on Image Processing (ICIP)*, November 1994.

[9] Y. Nakamura. *Advanced Robotics: Redundancy and Optimization*. Addison-Wesley, 1991.

[10] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1994.

[11] J. Rehg and A. Witkin. Visual tracking with deformation models. In *Proceedings of the IEEE International Conf. on Robotics and Automation*, April 1991.

[12] J. Shi and C. Tomasi. Good features to track. In *IEEE Conf. on Comp, Vision and Pattern Recognition*, 1994.

[13] R. Szeliski. Image moscaicing for tele-reality applcations. In *Workshop on Applications of Computer Vision*, 1994.

[14] R. Szeliski and J. Coughlan. Spline-based image registration. Technical Report 94/1, DEC CRL, 1994.

[15] Y. Xiong. Content-based expansion for image matching. Technical Report CMU-RI-96-26, June 1996.

Figure 1: Tracking a 40x40 projective region across a 74 frame, 320x240 video sequence.

Figure 2: Projective tracking in a QuicktimeVR object movie (a sequence of still images). The bottom row shows the warped region (I_1').

Figure 3: Frames from tracking a curved object using a 48x48 piecewise projective patch with a 3x3 grid.

Figure 4: Frames from tracking a flexing object using a 48x48 piecewise projective patch with a 3x3 grid.

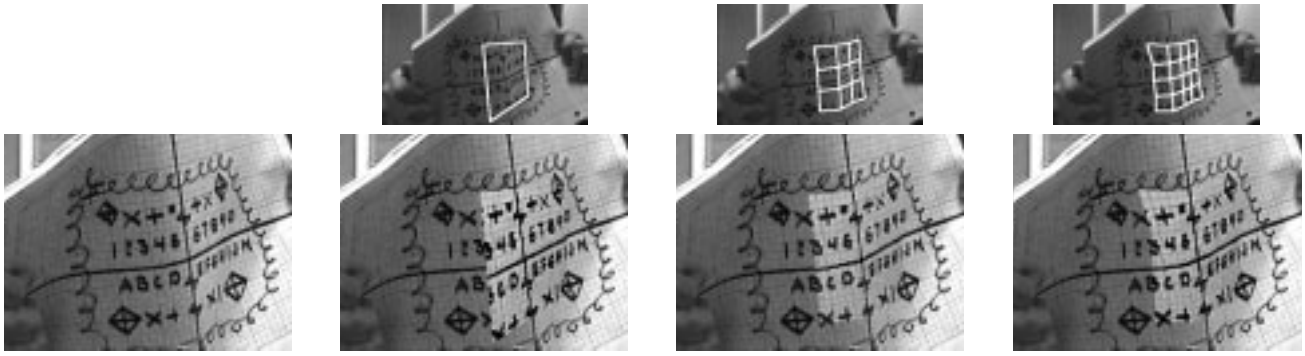

Figure 5: The sequence from Figure 4 is tracked with various 96x96 piecewise-projective patches. The upper row of images shows the trackers. Beneath each, the image is "reconstructed" by drawing the transformation of the patch from the first frame. We do not correct for illumination. The lower left image shows the portion of the original frame shown in the other images.

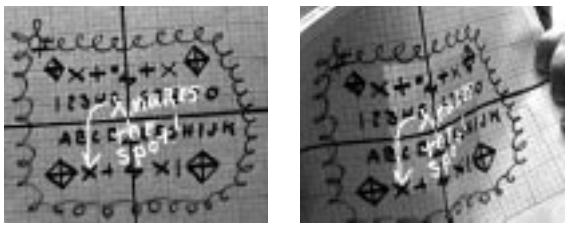

Figure 6: "Virtual Graffiti" - an application of tracking/reconstruction. After performing tracking (Figure 5), we paint on frame 1 of the video and the effects can be propagated to the rest.

Acknowledgements

This research was conducted in the computer graphics group of Apple Computers' Research Laboratories. I would like to thank Yalin Xiong for his assistance with this project including providing me with the QTVR example of Figure 2. Mark Wheeler and Roberto Manduchi, also of the QuicktimeVR group, helped with the preparation of the paper. Gavin Miller, my manager, encouraged this work as well as provided proofreading assistance.