A Direct Texture Placement and Editing Interface

Yotam I. Gingold, Philip L. Davidson, Jefferson Y. Han, Denis Zorin
New York University
719 Broadway
New York, NY 10003 USA
{gingold,philipd,jhan,dzorin}@mrl.nyu.edu

ABSTRACT
The creation of most models used in computer animation and computer games requires the assignment of texture coordinates, texture painting, and texture editing. We present a novel approach for texture placement and editing based on direct manipulation of textures on the surface. Compared to conventional tools for surface texturing, our system combines UV-coordinate specification and texture editing into one seamless process, reducing the need for careful initial design of parameterization and providing a natural interface for working with textures directly on 3D surfaces.

A combination of efficient techniques for interactive constrained parameterization and advanced input devices makes it possible to realize a set of natural interaction paradigms. The texture is regarded as a piece of stretchable material, which the user can position and deform on the surface, selecting arbitrary sets of constraints and mapping texture points to the surface; in addition, the multi-touch input makes it possible to specify natural handles for texture manipulation using point constraints associated with different fingers. Pressure can be used as a direct interface for texture combination operations. The 3D position of the object and its texture can be manipulated simultaneously using two-hand input.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Algorithms, Design, Human Factors.

Keywords: Texturing, surface parameterization, multi-touch interface.

INTRODUCTION
The goal of 3D texturing is to enhance object appearance by using images to modify surface material attributes, for example, colors, reflection coefficients, or normals.

In existing computer animation and computer-aided design systems, a typical pipeline for adding textures to surface meshes includes several stages, as shown in Figure 1, top row. These stages often correspond to separate tools or views.

First, the user selects a region of mesh to work with. The user then assigns texture coordinates to the mesh, using a combination of basic projection operations, more advanced automatic algorithms for mapping meshes to the plane, and manual adjustment of vertex positions. Once the texture coordinates are established, the polygons corresponding to this part of the mesh can be drawn in the image plane and used as a guide for constructing the texture map by adding, distorting, and blending preexisting images, as well as painting directly into the texture. The tools for establishing parameterization, editing the texture, and rendering a complete 3D model are often logically separate, and the user needs to switch between these three views multiple times during the process. The texture placement and painting process is indirect, as it happens not directly on the three-dimensional geometry but in texture space, where the shapes and sizes of surface triangles are often distorted. The user must mentally invert the map from the 3D mesh to texture space, adjusting for this distortion. Photographs and other images need to be warped in texture space to achieve good results in 3D; defining a suitable wrapping requires considerable experience and careful layout of texture coordinates.

We describe a system which eliminates the separation between these stages and enables direct manipulation of textures on surfaces (Figure 1, bottom row). All texture editing can be done directly on the 3D model, similar to 3D painting systems.

In our interaction metaphor, the texture can be thought of as a malleable thin sheet of material that clings to the surface and can be arbitrarily moved and deformed by constraining and moving points and areas of the texture. This directly corresponds to the intuitive idea behind texture mapping: the surface is decorated with textures. Traditional approaches often require the user to do the opposite: determine how to flatten parts of the surface to the plane of the texture. Our system allows multiple textures to be placed separately and then blended together by specifying the location and size of blend regions directly on the surface.

Our approach to texturing works particularly well with multi-touch and pressure-sensitive hardware. This type of input device allows the user to manipulate multiple points on the texture simultaneously, greatly enhancing his ability to specify complex deformations directly, rather than in multiple stages.

UIST'06, October 15–18, 2006, Montreux, Switzerland..

Sensitivity to pressure provides a natural framework for adjusting the size of texture areas affected by deformations and blending multiple textures.

We consulted with artists and found that the most cumbersome aspects of the texturing process are establishing a satisfactory parameterization and juggling the three views (3D model, parameterization, and texture editor). We interviewed, at length, an artist from Maxis/Electronic Arts. He characterized the problem of painting into a distorted, flattened region of mesh as surmountable, given training. However, he noted that he had no easy way to map or remap existing textures and photographs onto different models. He also described the iterative process of establishing a satisfactory parameterization: artists make minor adjustments to the parameterization in tandem with texture editing. Finally, he stated that he would prefer as much of the texturing process as possible to take place in the 3D view.

RELATED WORK

Our work builds on research in mesh parametrization, painting on 3D meshes, and multi-touch interfaces.

Parameterization. Mesh parametrization (i.e. computing a map or a collection of maps from a mesh to the plane) is fundamental to a broad range of applications, and we cannot do justice to the spectrum of work in the area. We focus primarily on the work most closely related to ours—specifically, on techniques which emphasize high performance and the ability to add constraints.

Most common techniques for parameterization of disk-like mesh areas rely on minimizing some measure of mesh distortion. A measure of distortion based on elasticity was introduced in Maillot et al. [28]; related stretching techniques are described in Piponi and Borshukov [30], Sander et al. [34], and Yoshizawa et al. [43]. Elasticity-based formulations are most commonly nonlinear, although Yoshizawa et al. [44] describe an approach which yields good results while solving only two linear systems. A different class of methods is based on solving a single linear system; these approaches are particularly suitable for interactive applications. A commonly-used general form with variable weights was described in Floater [9]; specific geometrically-motivated choices of weights were proposed in Desbrun et al. [7] and Lévy et al. [26]. An alternative approach (directly minimizing angle distortion) was proposed in Sheffer and de Sturler [38]; while the original method requires the relatively expensive solution of a nonlinear problem, a more efficient and robust version is described in Sheffer et al. [39]. Zayer et al. [45] introduce a boundary-free parameterization method that achieves good results by solving several linear systems. Parametrization based on tracing geodesics is described in Lee et al. [24].

A number of papers introduce various types of constraints into the parameterization. Lévy [25] uses penalty terms in the energy to approximate constraints. We propose a different energy that matches constraints exactly, which is important for usability, while Lévy weights smoothness of parameterization versus exactness of constraint-matching. We ensure smoothness with a higher-order energy and gradient-matching. Most importantly, our approach to incremental solution updating enables interactive texture manipulation. Desbrun and Alliez [7] suggest using the Lagrange multiplier formulation; Kraevoy et al. [22] point out that not all constraints for a given mesh can be satisfied by a one-to-one mapping and describe an algorithm for adding points to the triangulation so the constraints can be met. A number of papers consider the more difficult problem of consistently parameterizing several surfaces [32, 21, 36]; while these techniques can also be specialized to constrained parameterization, they are relatively expensive.

An important problem, not addressed in this paper, is partitioning the surface into patches that can be mapped to the plane, or finding the best way to cut the surface such that a single patch is formed. Many authors have developed different automatic approaches to this problem [31, 10, 20, 37, 35, 19, 47]. Yamauchi et al. [42] describe a complete automated texturing pipeline, largely following the traditional stages we have described above.

It is possible to avoid the parameterization problem entirely by storing texture information in a volume data structure (e.g. DeBry et al. [6]).

Finally, we note that our approach to texturing can be regarded as using image warping on a mesh [4], and that the incremental matrix inversion formula we use was applied in James and Pai [18] to interactive simulation.

3D painting. As introduced in Hanrahan and Haeberli [14], painting directly on the surface is a natural way to create textures from scratch. In Agrawala et al. [1], a tracker is used to "paint" on a real object; the paint appeared in the texture of the corresponding scanned computer model. Carr and Hart [5] show how texture resolution can be increased when painting on a surface. Igarashi and Cosgrove [16] discuss an adaptive technique for parametrization while 3D painting. In contrast, our emphasis is on the application and modification of preexisting textures, rather than creating textures from scratch.

Multi-touch interfaces. There is a large body of relevant work exploring two-handed input. Most of this work precedes the recent advent of practical multi-touch input devices and uses multiple mice/pucks or multimode Wacom tablets. Guiard [11] presents a theoretical framework for asymmetrically assigned roles in two-handed interaction, including the natural partitioning of a graphical task into view manipulation with the non-dominant hand and simultaneous editing or object manipulation with the dominant hand. 3D camera control and object manipulation with two mice was explored in Balakrishnan and Kurtenbach [3] and Zeleznik et al. [46], and with 6-DOF trackers in Hinckley et al. [15].

Graphical modeling operations using two hands in a symmetric manner have also been explored. This approach has been applied to simultaneously pan, zoom, or rotate the view context in a 2D map setting in Kurtenbach et al. [23], to alignment tasks in Balakrishnan and Hinckley [2], and for sophisticated 3D mesh modeling operations, as in Twister [27] which uses a pair of 6-DOF trackers.

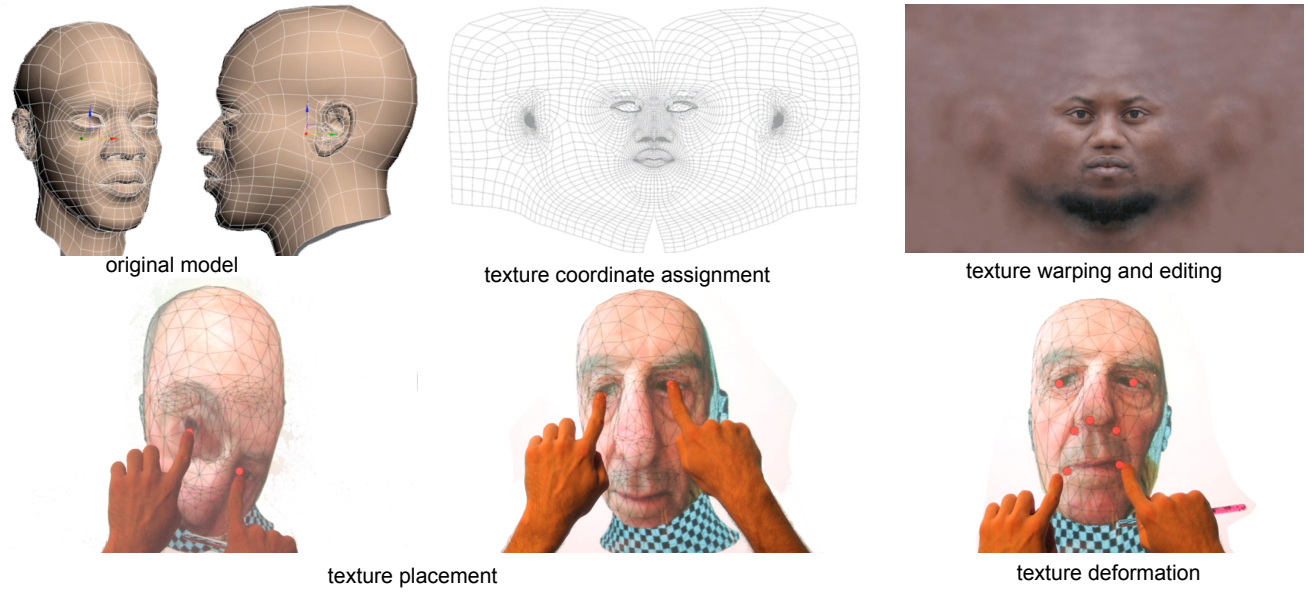

Figure 1: (top) The stages of traditional texturing, where the artist must switch contexts between 3D viewing and 2D texture painting and alignment [images courtesy of Jiri Adamec]. (bottom) Our proposed texturing allows the user to manipulate texture parameterization directly on the model.

While our texturing system can be used with a traditional input device and display, direct-display multi-touch interface devices are a natural match for its capabilities. These devices [8, 33, 40, 13] provide the advantage of direct graphical interaction, as well as benefits similar to those found in the literature on bi-manual input using multiple-mice: Wu and Balakrishnan [41] demonstrate two-finger rotation and scaling operations, while Igarashi et al. [17] describe a system for interactively applying complex free-form deformations of 2D models in the plane, where an arbitrary number of fingers each provide an additional 2-DOF control.

In this work, we use a multi-touch sensing technique recently introduced by Han [13], based on frustrated total internal reflection. Our system is implemented in a 36" drafting table form-factor, with a sensing resolution of approximately 20 ppi at 50 Hz. The system uses a compliant surface layer to provide reliable pressure sensitivity, which enables the use of passive styluses. As a result, users are free to use fingers or styluses as they see fit, for precision as well as comfort.

USER INTERFACE

Next, we describe the interaction operations supported by our system. Our system supports the following modes: texture placement, texture deformation, gluing, and blending.

Texture placement mode. The first step in the texturing process is to establish an initial mapping to the mesh which we adjust and refine. First, a texture is mapped to the mesh using an automatic distortion-minimizing parameterization with free boundaries, such that the texture patch is in the interior of the mesh. The user can then use a two-point texture placement scheme: two points are placed on the mesh to define the texture's translation, rotation, and scale, as shown in Figure 2. Using the multi-touch interface, users can adjust the points simultaneously with two fingers, continuously adjusting texture position on the surface.

This is the only operation in our system that requires multi-touch and can't be performed with a mouse. When using a mouse, the user must separately translate, rotate, and scale.

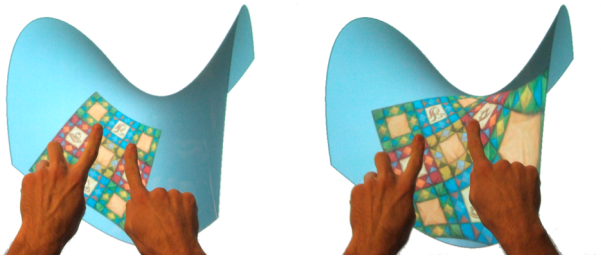

Figure 2: Two fingers completely determine the texture's translation, rotation, and scale.

Texture deformation mode. Once the initial location of the texture is defined, we fix the texture boundary and deform the texture on the surface in order to bring features of the texture into alignment with features on the mesh. This is achieved by defining a set of *live constraints*, which are points on the texture moved around by the user. The rest of the texture follows the live constraints; the influence of the constraints gradually decreases with distance. The influence of live constraints is further restricted by constraints defined in the glue tool, described below.

Once the user stops manipulating a live constraint, it converts into a *pushpin*, or dead, constraint. A pushpin constraint can be removed later; texture coordinates then elastically bounce back to their unconstrained positions. We find this behavior to be useful for undoing changes.

Alternatively, the user can choose to convert the deformation fixed by pushpins to *plastic*. As a result of this transformation, the current state of the texture does not change; however, pushpin constraints are no longer necessary to maintain the current mapping and are removed. All subsequent deformations are composed with the current deformation. This is useful for gradual, precise adjustment of textures without introducing excessive numbers of pushpin constraints, which may prevent smooth texture deformations. Figures 3 and 4 compare two types of deformations.

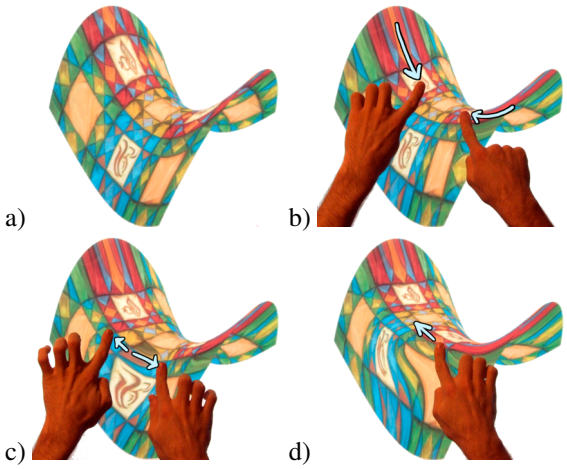

Figure 3: Plastic deformations: a) Undeformed parameterization. b) An initial plastic deformation by squeezing. c) A subsequent plastic stretching of the texture. d) A final deformation demonstrates the parameterization's plastic memory.

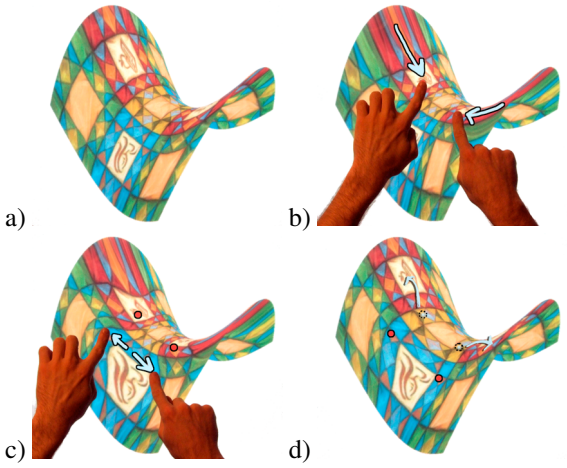

Figure 4: Elastic deformations: a) Undeformed parameterization. b) First, two live constraints squeeze the texture. c) Second, two additional live constraints stretch the texture. d) Finally, the pushpin constraints added in b) have been removed and the texture underneath bounces back to its unconstrained configuration.

Deformable regions: gluing mode and radius widget. We provide several tools for defining regions on the mesh.

We provide a *glue* brush to restrict deformations to particular areas of the mesh. In this mode, hidden pushpin constraints are applied to the vertices of triangles traversed by the brush. Glued areas can be used to separate regions of the mesh, acting as 'walls' across which changes cannot propagate. By completely encircling an area of the mesh, we limit the effect of edits inside to the interior region and, likewise, protect it from changes made outside. This is convenient for protecting local adjustments.

Alternatively, the user can specify a radius for the deformable area with a simple radius widget. Any constraint point automatically limits its influence approximately to the specified radius. (This is implemented by gluing the ring of triangles lying under a screen space circle of given radius.) Distant such constraint points have non-overlapping regions of influence that divide the mesh into multiple independent circles. These regions can be adjusted simultaneously, which is helpful in situations where multiple edits are taking place at once.

Figure 5: (left) Small and (right) large regions of influence.

Combining multiple textures, blending. Our texture adjustment naturally extends to multiple textures. For example, we may take multiple source photographs of an object to capture surface details from different directions. The user can adjust them separately to determine their relative alignment directly on the mesh geometry. This is convenient for accurately merging a number of source textures where the original alignment is not known.

Once relative positioning has been defined, we use an airbrush interface to blend out unnecessary or overlapping portions of the source textures, as illustrated in Figure 6. If the user wishes to have continuous, fine brush radius control with his non-dominant hand, the user can place two fingers on the radius widget to "pick it up" (demonstrated in the supplementary video).

Figure 6: (left) Small and (right) large areas of texture can be blended with the transparency airbrush.

Object positioning. In multi-touch settings, we can use a second hand to position the object while adjusting constraints with the other. This considerably simplifies texturing, as no mode switches are needed to change the orientation of the object, its scale, or its location on the screen.

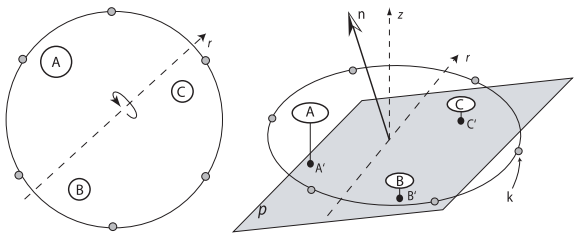

Figure 7: Tiltpad control: (left) On-screen interaction and (right) tiltpad interpretation.

The tiltpad is a disk-shaped isotonic pan-zoom-rotate control augmented by an isometric 'tilt' measurement, shown in Figure 7 (left). We first determine orientation in the plane, using the least-squares solution for translation, uniform scaling, and rotation. Then, we apply a 'tilt' rotation about an axis r in the view plane, calculated using terms indicated in Figure 7 (right). For the three contact points A, B, C (white) falling in the control area, we interpret pressure as 'depth' below the surface, and find the normal n of the best-fit plane p to the projected points A', B', C' (black). The 'tilt' transform is applied as an incremental rotation around the in-plane axis r that maps the z-axis to n. To ensure that n is well-defined, we add a set of weak constraint points k (grey) around the circular boundary of the control with depth equal to zero.

For consistency, we adjust solution parameters when points are added or removed in order to maintain a constant transformation. Pressure values are thresholded using a 'deadband' model, providing a transition from purely planar motion to tilt-sensitive manipulation.

ALGORITHMS

Next we describe essential algorithms used to implement the user interface described in the previous section.

Basic parameterization. At the heart of our system is an efficient algorithm for mapping a part of the mesh to the plane. It is similar to a widely used class of algorithms that solve a positive-definite linear system of equations.

However, we have observed that commonly used Floater-type algorithms, while performing well with either fixed or natural boundary constraints, do not perform as well in the setting with constrained interior points, especially for large distortion. Large deformations in this case result in mesh fold-overs in the texture domain, causing the same area of texture to be mapped onto multiple surface areas. Intuitively, this can be explained as follows. Qualitatively, the behavior of each coordinate obtained using a parametrization of this type is known to be similar to the behavior of the solution of the Laplace equation, which, in turn, is similar to the behavior of a soap film. For a point constraint in the interior of the domain, the solution will have a sharp point, thus likely to create a fold for small displacements.

While there are relatively complex approaches allowing for the complete elimination of the problem [22], we have found that switching the energy type to the linearized analog of *bending* energy significantly improves the behavior. One can think of this energy as the energy of flattening a membrane which resists bending to the plane (ignoring stretching). Mathematically, the discretization of this energy can be expressed as

$$E = \sum_i \frac{1}{8\,area_i} \left(\sum_{j \in N(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(\mathbf{t}_i - \mathbf{t}_j) \right)^2$$

where $\mathbf{t}_i = [u_i, v_i]$ is the 2D position of vertex i in the texture domain; i varies over all vertices and $N(i)$ is the set of vertices adjacent to i; $area_i$ is computed as described in Meyer et al. [29]; and the angles α_{ij} and β_{ij} are those shown in Figure 8. This energy is based on the discretization for mean curvature presented by Meyer et al..

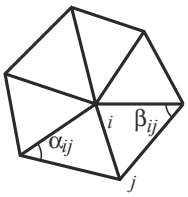

Figure 8: Angles referenced in the energy equation.

We observe that E is a quadratic function of the texture coordinates. To minimize this energy, we solve the system of linear equations with matrix A obtained by differentiating E twice with respect to all nonfixed points. For boundaries, we use two types of constraints: simple fixed constraints (in deformation mode) as well as natural constraints for free boundaries (see Desbrun et al. [7]) when the texture is initially placed on the surface.

Constraints. As described above, a constraint is a point correspondence between an arbitrary point in the texture chosen by the user and a point on the mesh. The point on the mesh need not be a vertex, so we express this constraint as a linear combination of the texture coordinates of the vertices of the triangle in the mesh containing the constrained point:

$$\alpha_1 \mathbf{t}_1 + \alpha_2 \mathbf{t}_2 + \alpha_3 \mathbf{t}_3 = \mathbf{t}^{fixed}$$

where $\alpha_1 + \alpha_2 + \alpha_3 = 1$ are the barycentric coordinates of the constrained point and $\mathbf{t}_i$, $i = 1, 2, 3$, are the texture coordinates of the three vertices of the triangle. In a more concise form we can write these constraints as two equations (one each for u and v) of the form $c^T t = d$, where t is the vector of texture coordinates for all points, c^T is the vector of coefficients with only three nonzero entries, and d is the fixed value.

To ensure the constraints are specified precisely, we use the standard Lagrange multiplier approach, i.e. we add extra variables and equations to our system. Instead of minimizing

the original quadratic functional E, for each constraint equation $c_i^T t = d_i$ we add a term $\lambda_i(c_i^T t - d_i)$. This leads to a linear system with the matrix

$$A^{ext} = \begin{pmatrix} A & C^T \\ C & 0 \end{pmatrix}$$

where C is a matrix composed out of vectors c_i. If we have M constraints, then the size of C is $M \times N$, where N is the number of vertices. In particular, this means the matrix changes whenever constraints are added and removed, and the linear system has to be solved from scratch. However, we minimize the amount of needed computation by using incremental inverse updates based on the Sherman-Woodbury-Morrison formula, discussed in more detail below.

Incremental matrix updates. All deformations resulting from live constraints are elastic, i.e. are computed using fixed cotangent coefficients in the energy E. We use the fact that only a relatively small part of the system matrix A^{ext} changes to incrementally compute the inverse whenever the constraints change. Here we briefly summarize the procedure, described in more detail in the appendix. We rely on the following facts. The crucial observation is that the matrix A^{ext} can be written as

$$\begin{aligned} A^{ext} &= \begin{pmatrix} A & 0 \\ 0 & I_{M\times M} \end{pmatrix} \\ &- \begin{pmatrix} 0 & -C^T \\ I_{M\times M} & \frac{1}{2} I_{M\times M} \end{pmatrix} \begin{pmatrix} -C & \frac{1}{2} I_{M\times M} \\ 0 & I_{M\times M} \end{pmatrix} \\ &= A^0 - UV \end{aligned}$$

where A^0 does not depend on constraints, U and V have dimensions $(N+M)\times 2M$ and $2M \times (N+M)$, respectively, and $I_{M\times M}$ is the identity matrix of size $M \times M$.

The classic Sherman-Morrison-Woodbury formula [12] makes it possible to solve the inverse system with matrix A^{ext} with M live constraints at the cost of solving $2M$ systems with matrix A^0 and different right-hand sides. This yields a substantial increase in performance if M is not very large and the matrix A^0 can be prefactorized. That is, the cost of solving $2M$ systems with a prefactorized matrix is substantially lower than the cost of assembling and solving a single system from scratch. We have observed this to be the case if we use an efficient direct solver and assume that M is no more than 10 (the maximum number of live constraints when using the multi-touch interface with two hands) and typically less.

On a 2.8 GHz P4, the seconds per update of a 2138 vertex mesh is, for 1 through 8 live constraints: .022, .026, .030, .033, .038, .043, .048, .056. The speedup over re-solving the system is between 7 and 9 times.

Plastic deformation. In plastic deformation mode, once a deformation is completed (in the case of the mouse-based interface, on the button-up event; for the multi-touch interface, when the last finger leaves the table), the entries of matrix A are updated using areas and cotangent weights computed from the texture coordinates. This mimics the plastic deformation of materials such as clay: the undeformed state is tracking the deformation, so when constraints are released the material does not spring back to the original position but stays in the deformed state.

Texture blending implementation. To implement our transparency airbrush tool, we start with a point C in screen space and pick the corresponding point on the mesh. We construct the matrix converting the picked triangle's texture coordinates to screen (pixel) coordinates. We use the inverse of this mapping to find a rough bounding box of the airbrush in texture coordinates. We then iterate over every texel in this rectangular bounds and, using our texture-to-screen mapping, project the texture coordinate onto the screen and compute its new α value based on its screen space distance from C (the place the user clicked/touched). We use the following α update formula

$$\alpha \leftarrow \alpha - flow\left(1 - 1/(2-s)^2\right)$$

where s is the normalized distance from C to the texel such that $s = 1$ when the distance equals the airbrush radius and $flow$ is the rate of airbrush "spray."

RESULTS

To demonstrate our system in action, we texture map an existing mesh geometry using casual photographs of a subject (Figure 9). We show a simplified workflow to quickly and easily produce a parameterized texture from snapshots taken from a family photo album. By aligning the different photographs directly on the mesh geometry, we avoid awkward operations in 2D space in which the user must map source images to the flattened representation of texture space.

Our workflow is illustrated in Figure 10 as well as the supplementary video. We load the mesh geometry, along with the first photograph from our set. Our first photograph is a portrait view, so we adjust the mesh orientation to roughly correspond to that viewing angle. In the two-point affine mapping stage, we align the two eyes to determine boundary conditions (a). From there, we use elastic deformation operations to move different features, such as the mouth, nose, and eyebrows, into alignment with the mesh geometry (b).

In (c), we move to a view of the left side and load in the second photograph. From this view, we can establish a base parameterization using the eye and ear. We move to a partial frontal view (d) to further refine the alignment of features around the lips and eye. We have now brought the separate textures into alignment for their appropriate regions. To eliminate the overlap between the two, we switch to texture blending mode (e), erasing areas of the textures which contain off-angle features or background imagery. Finally, we load the third image, taken from the right side, align it, and blend overlapping regions (f).

We have now obtained (g), a relatively uniform texture mapping on the model from three separate photographs. The relative mappings of the mesh to the separate texture UV spaces of the photographs are shown in Figure 11. Some artifacts exist from lighting difference in the original snapshots; these could be adjusted in a secondary image processing iteration. The major task of aligning feature data from the three im-

Figure 9: Geometric model and source photographs.

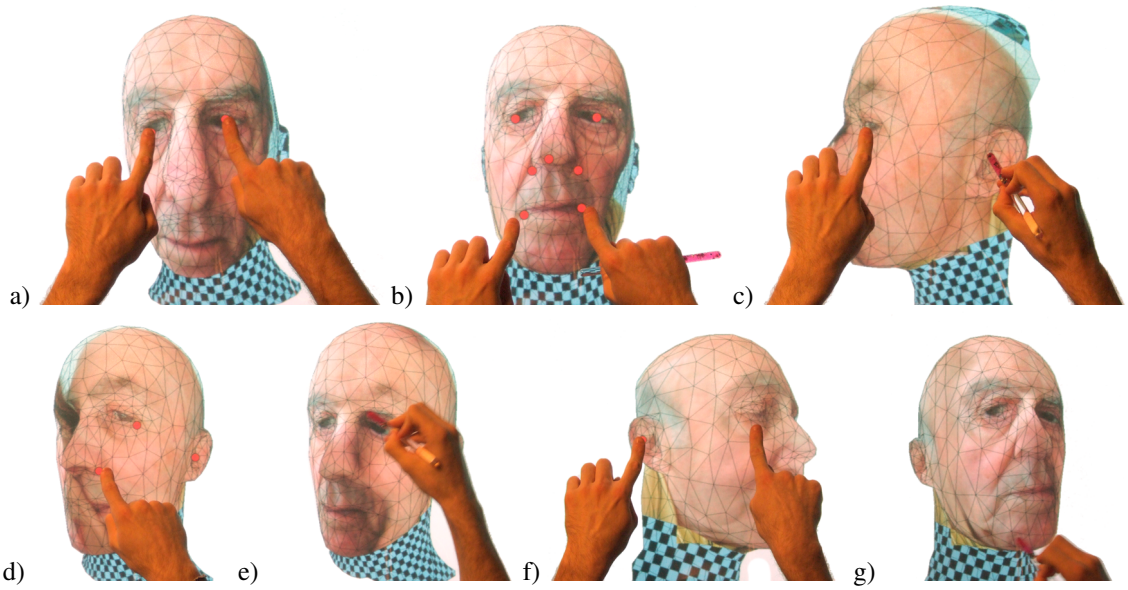

Figure 10: A sequence of steps showing the texturing of a model using our system. See the Results section for a description.

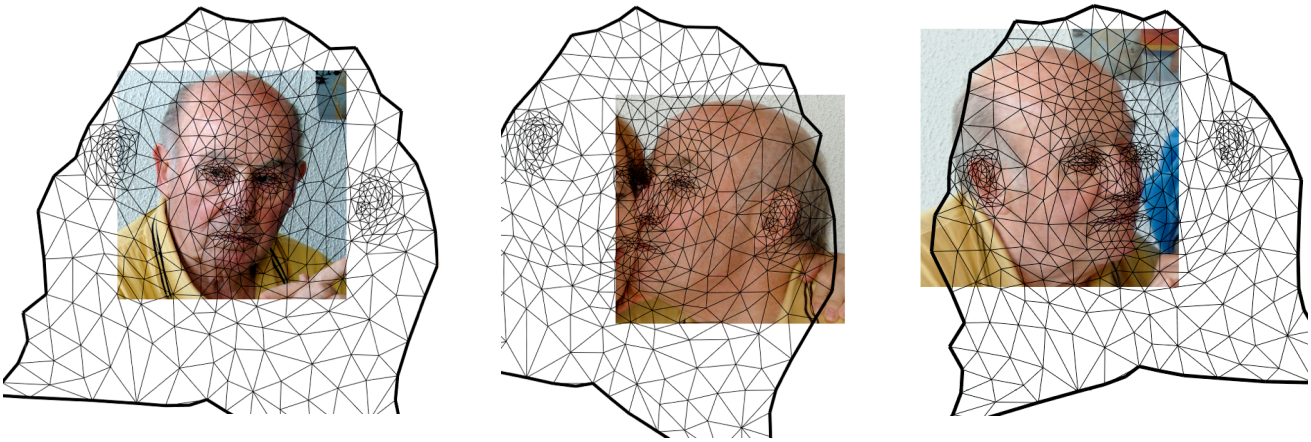

Figure 11: Texture maps generated by the system for the model shown in Figure 10.

ages to the mesh has been accomplished straightforwardly with our direct-manipulation interface.

CONCLUSIONS

Our experiments suggest that the interface for texturing described in this paper makes it possible to create textured models faster and requires less skill from the user than conventional techniques.

Artists typically use professional image editing software to edit textures. A limitation of our approach is that the large variety of tools already available in such systems cannot be leveraged directly. Thus, one needs to have a far more extensive collection of image-editing tools in the application itself. Our blending tool is just a first step in this direction.

In the current implementation, we use multitexturing to combine several textures on a model. For many applications this can be regarded as a limitation; it is often desirable to combine all color textures into a single texture for each part of the mesh. To do this, our interface needs to be combined with global parametrization tools. Texture resolution also may require adjustment, and the approach proposed in Carr and Hart [5] is likely to be a useful addition to the system. Increasing the robustness of the parametrization, e.g. by refining the mesh when necessary as in Kraevoy et al. [22], is another important direction for improving the back-end of our system.

Acknowledgments. We would like to thank Mike Khoury for his insights into the texturing process; NYU Computer Science colleagues for useful discussions; and the anonymous reviewers for their helpful suggestions.

Appendix: Sherman-Woodbury-Morrison formula

We follow Hager [12] to update the inverse of an $n \times n$ matrix A with a low-rank modification of the form UV, where U is $n \times m$, V is $m \times n$, and m is much smaller than n. The Sherman-Morrison-Woodbury formula

$$B^{-1} = [A - UV]^{-1} = A^{-1} + A^{-1}U(I - VA^{-1}U)^{-1}VA^{-1}$$

leads to the following algorithm for solving a system of equations $Bx = b$:

1. Solve $Ay = b$ for y.
2. Compute the $n \times m$ matrix $W = A^{-1}U$ by solving m linear systems $Aw_i = u_i$, where w_i and u_i are columns of W and U.
3. Form a small $m \times m$ matrix $C = I - VW$ and solve $Cz = Vy$ for z.
4. Set $x = y + Wz$.

This algorithm yields a considerable speedup if A can be factorized to a form which makes it possible to solve the system $Ay = b$ for different right-hand sides as needed in step 2. This is typical for direct solvers; e.g. we use the PARDISO solver, a part of the Intel MKL, which prefactors the matrix A as LDL^T, where L is a lower triangular and D is a diagonal matrix. Solving a system with A represented in this form is an order of magnitude faster than the cost of factorization.

REFERENCES

1. M. Agrawala, A. C. Beers, and M. Levoy. 3D painting on scanned surfaces. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 145–150, 1995.
2. R. Balakrishnan and K. Hinckley. Symmetric bimanual interaction. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 33–40, 2000.
3. R. Balakrishnan and G. Kurtenbach. Exploring bimanual camera control and object manipulation in 3D graphics interfaces. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 56–63, 1999.
4. T. Beier and S. Neely. Feature-based image metamorphosis. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 35–42, 1992.
5. N. A. Carr and J. C. Hart. Painting detail. *ACM Transactions on Graphics*, 23(3):845–852, 2004.
6. D. G. DeBry, J. Gibbs, D. D. Petty, and N. Robins. Painting and rendering textures on unparameterized models. *ACM Transactions on Graphics*, 21(3):763–768, 2002.
7. M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum*, 21(3):209–218, 2002.
8. P. Dietz and D. Leigh. DiamondTouch: a multi-user touch technology. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 219–226, 2001.
9. M. S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997.
10. X. Gu, S. J. Gortler, and H. Hoppe. Geometry images. *ACM Transactions on Graphics*, 21(3):355–361, 2002.
11. Y. Guiard. Asymmetric division of labor in human skilled bimanual action: The kinetic chain as a model. *The Journal of Motor Behavior*, 19(4):486–517, 1987.
12. W. W. Hager. Updating the inverse of a matrix. *SIAM Review*, 31(2):221–239, 1989.
13. Jefferson Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 115–118, 2005.
14. P. Hanrahan and P. Haeberli. Direct WYSIWYG painting and texturing on 3D shapes. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 215–223, 1990.
15. K. Hinckley, R. Pausch, J. C. Goble, and N. F. Kassell. Passive real-world interface props for neurosurgical visualization. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 452–458, 1994.
16. T. Igarashi and D. Cosgrove. Adaptive unwrapping for interactive texture painting. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 209–216, 2001.
17. T. Igarashi, T. Moscovich, and J. F. Hughes. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics*, 24(3):1134–1141, 2005.
18. D. L. James and D. K. Pai. ArtDefo: accurate real time deformable objects. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 65–72, 1999.
19. M. Jin, Y. Wang, S.-T. Yau, and X. Gu. Optimal global conformal surface parameterization. In *15th IEEE Visualization 2004 (VIS'04)*, pages 267–274, 2004.

20. A. Khodakovsky, N. Litke, and P. Schröder. Globally smooth parameterizations with low distortion. *ACM Transactions on Graphics*, 22(3):350–357, 2003.

21. V. Kraevoy and A. Sheffer. Cross-parameterization and compatible remeshing of 3D models. *ACM Transactions on Graphics*, 23(3):861–869, 2004.

22. V. Kraevoy, A. Sheffer, and C. Gotsman. Matchmaker: constructing constrained texture maps. *ACM Transactions on Graphics*, 22(3):326–333, 2003.

23. G. Kurtenbach, G. Fitzmaurice, T. Baudel, and B. Buxton. The design of a GUI paradigm based on tablets, two-hands, and transparency. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 35–42, 1997.

24. H. Lee, Y. Tong, and M. Desbrun. Geodesics-based one-to-one parameterization of 3D triangle meshes. *IEEE Multimedia*, 12(1):27–33, 2005.

25. B. Lévy. Constrained texture mapping for polygonal meshes. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 417–424, 2001.

26. B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics*, 21(3):362–371, 2002.

27. I. Llamas, B. Kim, J. Gargus, J. Rossignac, and C. D. Shaw. Twister: a space-warp operator for the two-handed editing of 3D shapes. *ACM Transactions on Graphics*, 22(3):663–668, 2003.

28. J. Maillot, H. Yahia, and A. Verroust. Interactive texture mapping. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 27–34, 1993.

29. M. Meyer, M Desbrun, P. Schröder, and A. H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, pages 35–57. Springer-Verlag, Heidelberg, 2003.

30. D. Piponi and G. Borshukov. Seamless texture mapping of subdivision surfaces by model pelting and texture blending. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 471–478, 2000.

31. E. Praun, A. Finkelstein, and H. Hoppe. Lapped textures. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 465–470, 2000.

32. E. Praun, W. Sweldens, and P. Schröder. Consistent mesh parameterizations. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 179–184, 2001.

33. J. Rekimoto. SmartSkin: an infrastructure for freehand manipulation on interactive surfaces. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 113–120, 2002.

34. P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe. Texture mapping progressive meshes. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, pages 409–416, 2001.

35. P. V. Sander, Z. J. Wood, S. J. Gortler, and H. Hoppe. Multi-chart geometry images. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 146–155, 2003.

36. J. Schreiner, A. Asirvatham, E. Praun, and H. Hoppe. Inter-surface mapping. *ACM Transactions on Graphics*, 23(3):870–877, 2004.

37. A. Sheffer. Skinning 3D meshes. *Graphical Models*, 65(5):274–285, 2003.

38. A. Sheffer and A. S. E. de Sturler. Parameterization of faceted surfaces for meshing using angle-based flattening. *Engineering with Computers*, 17(3):326–337, 2001.

39. A. Sheffer, B. Lévy, M. Mogilnitsky, and A. Bogomyakov. ABF++: fast and robust angle based flattening. *ACM Transactions on Graphics*, 24(2):311–330, 2005.

40. A. D. Wilson. TouchLight: an imaging touch screen and display for gesture-based interaction. In *ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces*, pages 69–76, 2004.

41. M. Wu and R. Balakrishnan. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 193–202, 2003.

42. H. Yamauchi, H. P. A. Lensch, J. Haber, and H. P. Seidel. Textures revisited. *Visual Computer*, 21(4):217–241, 2005.

43. S. Yoshizawa, A. Belyaev, and H. P. Seidel. A fast and simple stretch-minimizing mesh parameterization. In *SMI '04: Proceedings of the Shape Modeling International 2004 (SMI'04)*, pages 200–208, 2004.

44. S. Yoshizawa, A. Belyaev, and H. P. Seidel. A moving mesh approach to stretch-minimizing mesh parameterization. *International Journal of Shape Modeling*, 11(1):25–42, 2005.

45. R. Zayer, C. Rossl, and H.-P. Seidel. Setting the boundary free: A composite approach to surface parameterization. In *Symposium on Geometry Processing*, pages 91–100, 2005.

46. R. Zeleznik, A. Forsberg, and P. Strauss. Two pointer input for 3D interaction. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 115–120, 1997.

47. E. Zhang, K. Mischaikow, and G. Turk. Feature-based surface parameterization and texture mapping. *ACM Transactions on Graphics*, 24(1):1–27, 2005.