University of Calgary, Department of Computer Science Technical Report 2006-824-17

Interactive Decal Compositing with Discrete Exponential Maps

Ryan Schmidt[*] University of Calgary
Cindy Grimm[†] Washington University in St. Louis
Brian Wyvill[‡] University of Calgary

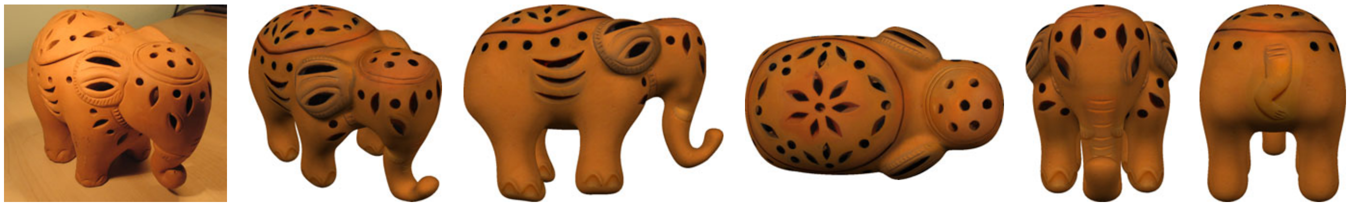


Figure 1: *A clay elephant statue (left) was modeled using sketch-based implicit-surface modeling software. Then, a lapped base texture and 25 feature textures were extracted from 22 images taken with a digital camera and composited on the surface. Photography, image creation, and texture positioning was completed in under an hour.*

Abstract

A method is described for texturing surfaces using *decals*, images placed on the surface using local parameterizations. Decal parameterizations are generated with a novel $O(N \log N)$ discrete approximation to the exponential map which requires only a single additional step in Dijkstra's graph-distance algorithm. Decals are dynamically composited in an interface that addresses many limitations of previous work. Tools for image processing, deformation/feature-matching, and vector graphics are implemented using direct surface interaction. Exponential map decals can contain holes and can also be combined with conformal parameterization to reduce distortion. The exponential map approximation can be computed on any point set, including meshes and sampled implicit surfaces, and is relatively stable under resampling. The decals stick to the surface as it is interactively deformed, allowing the texture to be preserved even if the surface changes topology. These properties make exponential map decals a suitable approach for texturing animated implicit surfaces.

1 Introduction

Texture mapping [Blinn and Newell 1976] is one of the major stages in the modeling and animation pipeline. Texture design is generally a manual process and consumes a significant amount of the effort in most animation projects. Constrained parameterization techniques [Lévy 2001] can provide some relief if suitable images are available, however constraint placement is tedious and may need to be repeated if the surface or texture is modified. Painting tools, particularly 3D painting systems [Hanrahan and Haeberli 1990], are the real workhorses of interactive texture design. Unfortunately these tools are relatively inflexible. Useful operations such as copy-and-paste are unavailable and there is no provision for re-using existing textures.

A third style of texturing interface, which combines aspects of both painting and constraint tools, is the *decaling* interface, introduced by Pedersen [1996]. In this approach the metaphor is that of decals, or "stickers", which are 2D images affixed to the surface. Decals are treated as independent scene elements which are simply constrained to lie on surfaces, but may otherwise be interactively manipulated. Because a simple mapping exists between the image and the surface, 2D image processing tools can be trivially implemented. Decals are composited in real-time, mimicking 2D image compositing [Porter and Duff 1984] and vector graphics interfaces. This approach allows artists to interact with surface texture directly, using familiar 2D methods and tools. One of the largest benefits of decaling is that it allows for easy re-use of 2D images in texture design. When combined with a digital camera or image database, realistic textures can be created very quickly (Figure 1).

Pedersen's [1996] pioneering work on decaling interfaces had many practical limitations. A global base parameterization was required, complicating use on implicit and point-set surfaces and preventing animation. Decals were limited to deformed rectangles by the iterative mass-spring mesh optimization approach taken to parameterization. The user was required to manually define the decal corners, and it was not possible to automatically create a decal around an arbitrary surface curve or update the decals if the underlying surface changed. Some recent systems have taken decal-like approaches, including lapped textures [Praun et al. 2000] and texture sprites [Lefebvre et al. 2005], however neither provides support for interactive editing tools such as cut-and-paste.

We present a decaling interface that addresses many of the problems encountered in Pedersen's work [1996]. First, our approach is entirely local - we do not require a base parameterization or any other pre-processing of the surface beyond initial sampling. Our decals are based on a local exponential map parameterization (Section 4) which is generated from a single point and geodesic radius, simplifying the user interface and supporting automatic creation of decals. To efficiently generate these parameterizations we introduce a novel discrete approximation to the exponential map which requires only a simple addition to Dijkstra's algorithm (Section 4.2). The approximation is computed on a point set, implying that any surface which can be sampled - triangle mesh, point set, or implicit surface - can be textured with exponential map decals.

Our decals compare favorably to decals created using local conformal parameterization, in particular we find that exponential map decals often preserve an intuitive sense of "squareness" that is lost with parameterizations based on global optimization. However, we achieve our most robust results by combining the two approaches (Section 5). Distortion is also reduced by allowing decals to have holes. We introduce techniques for interacting with decals, including a deformation tool and surface vector graphics (Section 6). We also address texturing animated implicit surfaces (Section 7),

[*]e-mail: rms@cpsc.ucalgary.ca
[†]e-mail: cmg@cs.wustl.edu
[‡]e-mail: blob@cpsc.ucalgary.ca

a problem which has eluded previous decaling systems. We show that our approach can preserve texturing even in the presence of topological changes.

2 Related Work

Interactive painting [Hanrahan and Haeberli 1990] has dominated texture mapping interfaces. These tools are very robust. The recent development of Octree textures [Benson and Davis 2002] [DeBry et al. 2002] supports 3D painting on most types of surfaces without requiring a surface parameterization. The drawback of this class of interfaces is that painting realistic textures requires manual dexterity and artistic skills which most people lack.

Another type of texture mapping interface is constrained parameterization [Lévy 2001; Kraevoy et al. 2003]. Here a set of constraints are manually specified between the desired texture image and the surface. Global optimization algorithms are then applied to map the image onto the surface such that the constraints are satisfied and some distortion metric is minimized. Recent advances support point sets [Zwicker et al. 2002], and atlas generation from multiple images [Zhou et al. 2005]. These systems do not address the general problem of texture design, the desired 2D image(s) are assumed to already exist. Constrained parameterization can also be used to apply decals, however the fluid interface of [Pedersen 1996] is difficult to implement because of the problem of simultaneously moving all of the constraints across the surface.

Procedural texturing [Ebert 2002] provides a semi-automated approach to texture generation in exchange for detailed artistic control. Only certain classes of texture can be generated, and many methods assume that the surface is already parameterized. Lapped textures [Praun et al. 2000] take a decal-like approach to texture synthesis by overlapping many small images on a mesh. Our decals can be used to apply lapped textures to any sampled surface.

Pedersen [1996] describes an interactive texture design tool which supports high-level operations such as copy-and-paste, where the copy region can be interactively dragged across the surface. Any type of surface can be used, although implicit surfaces require a time-consuming manual segmentation [Pedersen 1995]. Decals (called *patchinos* in this work) are created by manually connecting four points on the surface with geodesics, limiting the copy and paste regions to deformed rectangles. A rectangular or cylindrical decal shape is required by the iterative mass-spring optimization technique used to move decals. This method is noted to be unstable [Pedersen 1996], and when the mass-spring mesh collapses the decal is lost and the user must re-start the process. One of the main contributions of our work is to introduce a more robust method for representing and interacting with decals.

[Lefebvre et al. 2005] describe an octree-texture-based decaling system based on parallel projection, limiting the use of decals to relatively flat surfaces unless severe distortion is acceptable. Operations such as copy-and-paste are not available because the surface is not explicitly parameterized.

One approach to decaling is to simply drag square images around in texture space, where a surface parameterization has already been computed automatically using global mesh parameterization algorithms [Floater 1997; Desbrun et al. 2002; Sheffer et al. 2005; Gu and Yau 2003]. Related methods can be applied for point set parameterization [Floater and Reimers 2001; Alexa et al. 2003]. However, for most surfaces the global parameterization necessarily introduces significant distortion, resulting in decals that change size as they are dragged across the surface. These techniques can also be used to apply decals by interactively computing the parameterization only in the region of the decal. However, as we will show in Section 5.1, this can result in unexpected distortion. Parameterization atlases [Maillot et al. 1993; Sorkine et al. 2002; Grimm 2004; Zhang et al. 2005] can reduce global distortion, however discontinuities in distortion across the atlas boundaries again result in unexpected decal deformations.

Parameterizing implicit surfaces presents some additional challenges, particularly if the surface is animated. Existing approaches are not interactive and provide either very limited control over where textures are applied [Tigges and Wyvill 1999] or require topological knowledge a priori [Grimm 2004]. Octree texturing has limited applicability because there is no way to map back to the "rest pose" described by [Lefebvre et al. 2005] during animation. None of these methods address the issue of animated topology change. Our system produces consistent and predictable results in these situations. Additional control is available to the animator since decals can be keyframed.

3 Overview

The goal of our decaling interface is to make 3D surface texture design as fluid and straightforward as 2D vector graphics and image compositing is with software such as Adobe Illustrator. This is accomplished by hiding all aspects of surface parameterization in the texturing interface. Unlike [Pedersen 1996], we do not require a global base parameterization. Each decal is an independent object in the system, with a simple layer order to determine the decal compositing sequence. The artist interacts with decals, rather than the underlying parameterizations. All decals are dynamically composited each frame using the alpha blending and texturing mapping hardware found on commodity graphics hardware (Figure 2a). Other blending modes, such as dodge and burn, can be implemented with programmable GPUs.

Decals are dynamically generated based on a center point, orientation, and radius. A decal is dragged across the surface by repositioning the center point with ray-surface intersections. The orientation and radius are interactively controlled with simple 3D widgets (Figure 2b).

Decals can also be used as canvases for 2D operations. 2D vector graphics objects such as lines and curves can be manipulated by moving control points on the surface (Figure 2d). Local feature-alignment can be performed interactively by manipulating 2D image deformations, again via 3D control points (Figure 2c). Curves drawn in screen space can also be projected into decals and used to control image processing operations such as blurring (Figure 2a) and composited copy-and-paste (Figure 2b). Painting tools can be implemented by using decals as local canvases for traditional 2D or 3D painting interfaces. The advantage of the decal-as-canvas metaphor is that decals can be (re)moved.

Finally, decals maintain consistency as the underlying surface changes. Difficult cases, such as topological change in implicit surfaces (Figure 2e), are handled robustly. Decals also provide a mechanism for texturing animated implicit surfaces. In the following sections we will explain how our decals are created, as well as describe the implementation and use of our decaling interface.

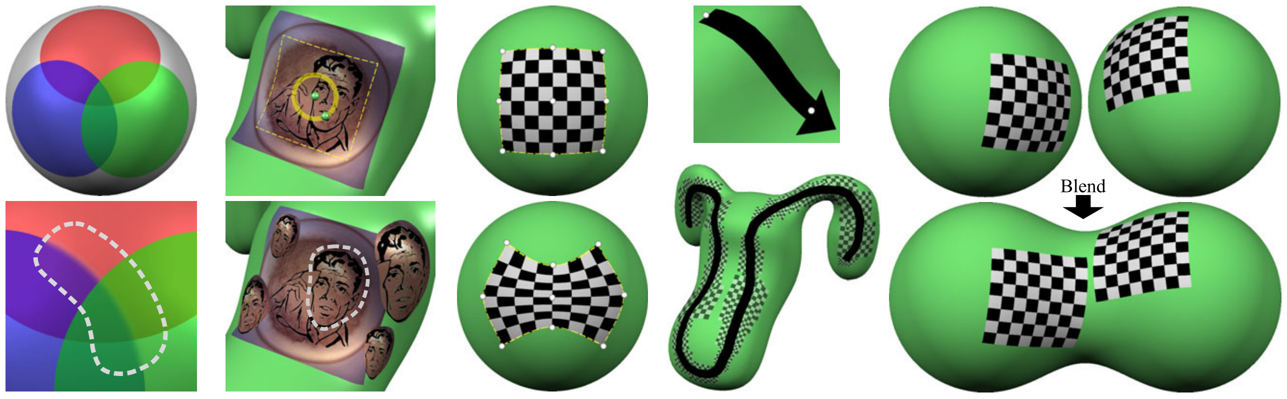


Figure 2: *Three overlapping semitransparent decals are composited in (a, top). A selection region is used to blur the blue decal (a, bottom) while leaving the red and green decals unchanged. In (b), the two decals inside the selection region are baked into a new decal which has been pasted back onto the surface several times. In (c), a lattice tool is used to deform the decal. A surface vector line with an endcap is created between the control points in (d,top) by rendering a 2D line into an automatically-generated decal. Surface curves can be rendered using a set of vector lines (d,bottom). The underlying decals are shown as checkerboards. In (e), the decals on two separate implicit surfaces are automatically updated when the surfaces are blended.*

4 Exponential Maps

Our goal is to map a 2D image onto a 3D surface $\mathcal{S}$ such that the center of the image lies at some point $\mathbf{p}$ on $\mathcal{S}$. This mapping requires a local 2D coordinate system on the surface around $\mathbf{p}$. Differential geometry provides us with a means for defining this mapping - the *exponential map* [do Carmo 1976]. The exponential map exp_p takes points on $\mathcal{S}$ to the tangent plane $\mathcal{T}_p$ at $\mathbf{p}$. This is accomplished via geodesics. For any unit vector $\mathbf{v} \in \mathcal{T}_p$, there exists a geodesic $g_\mathbf{v}$ parameterized by arc length such that $g_\mathbf{v}(0) = \mathbf{p}$ and $g'_\mathbf{v}(0) = \mathbf{v}$.

Essentially, $g_\mathbf{v}$ are the geodesics that splay out radially from $\mathbf{p}$ (Figure 3). For any point $\mathbf{q}$ in the neighbourhood of $\mathbf{p}$, a unique radial geodesic $g_\mathbf{v}$ passes through it. Hence, $\mathbf{q}$ can be mapped to $\mathcal{T}_p$ with the polar coordinates (r_g, θ_g), where r_g is the geodesic distance from $\mathbf{p}$ to $\mathbf{q}$ and θ_g is the polar angle of $\mathbf{v}$ in $\mathcal{T}_p$. These are the *geodesic polar coordinates* [do Carmo 1976]. Geodesic polar coordinates can alternatively be expressed as *normal coordinates* (u, v) in any orthogonal basis $\{\mathbf{e}_1, \mathbf{e}_2\}$ of the tangent plane.

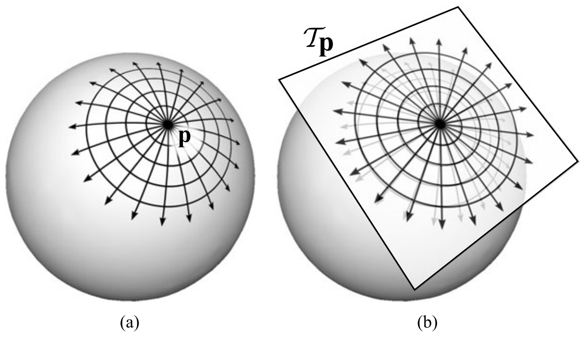


Figure 3: *The exponential map at a point* $\mathbf{p}$ *takes geodesic curves originating at* $\mathbf{p}$ *in (a) to straight vectors emanating from the origin of the tangent plane* $\mathcal{T}_p$ *in (b). Geodesic "circles", defined as isocontours of the geodesic distance function, are mapped to circles about the origin of* $\mathcal{T}_p$.

Normal coordinates are by definition a mapping from the plane to the surface and hence would seem to be an ideal solution for texture mapping. The inverse function theorem ensures that for any differentiable point on a surface, exp_p is defined and differentiable in some neighbourhood around $\mathbf{p}$ [do Carmo 1976]. If $\mathcal{S}$ is restricted to smooth manifolds, then the Hopf-Rinow theorem [Cheeger and Ebin 1975] states that exp_p is guaranteed to be *defined* on the entire surface. However, the map is still only diffeomorphic on a local neighbourhood of $\mathbf{p}$, implying that foldovers in the parameterization will occur if the neighbourhood grows too large.

An example of a sphere texture-mapped using an analytic exponential map parameterization is shown in Figure 4. In the neighbourhood of $\mathbf{p}$ (at the center of the leftmost image), the checkerboard exhibits low distortion. The parameterization is neither conformal nor area-preserving, as can be observed on the back of the sphere. However, the parameterization on the front of the sphere is very "square" and hence is ideal for applying a decal.

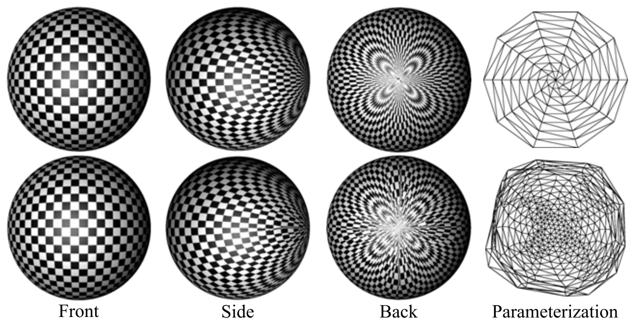


Figure 4: *Checkerboard texture applied to sphere using analytic parameterization (top) and discrete approximation (bottom). Front views show very high correspondence. Overall patterns in side and back views are similar, however the approximation exhibits errors accumulated during propagation. Rightmost images show 2D parameter space for lower-resolution triangulated spheres.*

4.1 Previous Approaches to Discrete Exponential Maps

Parameterizations based on exponential maps have seen some use in computer graphics. Projecting the neighbours of a triangle mesh vertex **p** onto the tangent plane at **p** is a crude approximation to exp_p. Welch and Witkin [1994] describe a more robust technique for approximating exp_p on the 1-ring of a triangle mesh vertex. This method has seen wide use as a component of global parameterization algorithms [Floater 1997], however the approximation does not extend beyond the 1-ring.

The key difficulty in computing exp_p is in finding the radial geodesic $g_{\mathbf{v}}$ that passes through a surface point **q**. Dijkstra's algorithm [Dijkstra 1959] is perhaps the best-known technique for approximating geodesic distances. Unfortunately the piecewise linear geodesics produced by Dijkstra's algorithm always lie on graph edges and hence provide a very poor estimation of θ_g.

A number of algorithms that produce more accurate geodesic distance approximations are available. See [Mitchell 2000] for a recent survey. The fast-marching mesh geodesic distance method presented by [Kimmel and Sethian 1998] runs in $O(N \log N)$ time. [Surazhsky et al. 2005] describe an algorithm which computes approximate mesh geodesic distances in a fraction of the time taken for the fast-marching approach. These algorithms are heavily dependent on the underlying mesh structure and hence unlikely to be adaptable to unstructured geometry such as point clouds.

4.2 Discrete Exponential Map Approximation

In the discrete setting, we have a set of points on some surface $\mathcal{S}$. To approximate the exponential map at **p**, we must find the geodesic distance and polar angle for each other point **q** on the surface. We approach the problem by computing these values directly in the tangent plane at **p**, rather than trying to find the surface geodesics. The resulting algorithm requires only a simple vector addition of the piecewise-linear geodesics produced by Dijkstra's algorithm.

First, consider the base case, with 3 points **p**, **r**, and **q**. The geodesics from **p** to **r** and **r** to **q** are known, however the geodesic from **p** to **q** is not (Figure 5a). Our task is not to find this geodesic but rather to compute $\mathbf{u}_{p,q}$, the 2D vector in normal coordinates of $\mathcal{T}_p$ produced by applying exp_p to **q**. Since $\mathbf{u}_{p,q}$ is a 2D vector, it can be rewritten as $\mathbf{u}_{p,q} = \mathbf{v} + (\mathbf{u}_{p,q} - \mathbf{v})$, where **v** is any 2D vector. Let $\mathbf{v} = exp_p(\mathbf{r}) = \mathbf{u}_{p,r}$. This is the known geodesic from **p** to **r**. We now have

$$\mathbf{u}_{p,q} = \mathbf{u}_{p,r} + (\mathbf{u}_{p,q} - \mathbf{u}_{p,r}) \tag{1}$$

where $(\mathbf{u}_{p,q} - \mathbf{u}_{p,r})$ is unknown. This 2D vector corresponds to some curve on the surface from **r** to **q**, although it is not in general a geodesic. We approximate this curve with the known geodesic from **r** to **q**. In the tangent plane at **r**, let $\mathbf{u}_{r,q} = exp_r(\mathbf{q})$. This is the necessary vector, however it is defined in normal coordinates of $\mathcal{T}_r$ and we need it in $\mathcal{T}_p$. To transform between the tangent planes, rotate $\mathcal{T}_r$ so that it is co-planar to $\mathcal{T}_p$, and then rotate around the normal to $\mathcal{T}_p$ to align the bases of the two planes. The second rotation can be treated as a 2D rotation in the tangent plane and applied directly to $\mathbf{u}_{r,q}$ (Figure 5b), so if the rotation angle is $\theta_{p,r}$, then $\mathbf{u}_{p,q}$ can be approximated by

$$\hat{\mathbf{u}}_{p,q} = \mathbf{u}_{p,r} + Rot2D(\theta_{p,r}) \cdot \mathbf{u}_{r,q} \tag{2}$$

as shown in Figure 5c (the red vector is $Rot2D(\theta_{p,r}) \cdot \mathbf{u}_{r,q}$).

We have made two approximations. First, assuming $(\mathbf{u}_{p,q} - \mathbf{u}_{p,r})$ is a geodesic is only correct on developable surfaces, where the Gaussian curvature $K = 0$. Second, we used a simple affine transformation to map between the tangent planes at **r** and **p**. Minding's theorem states that this map is isometric only between surfaces where K is a constant [do Carmo 1976]. Otherwise, some additional error is introduced. We have also assumed that the exponential map at any **p** can be computed in a small neighbourhood around **p**. In the discrete setting this local exponential map is approximated by transforming linear segments into the tangent plane. This local approximation, which we will denote $\widehat{exp}_p(\mathbf{q}) = \widehat{\mathbf{u}}_{p,q}$, is described in more detail in the next section.

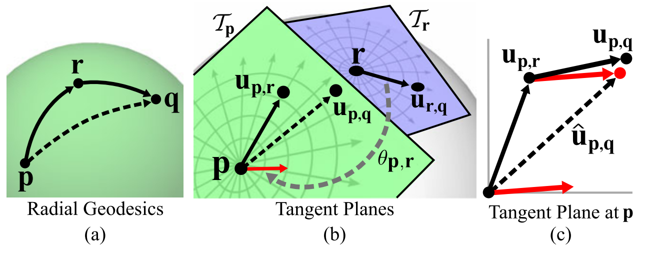


Figure 5: *The normal coordinates $\mathbf{u}_{p,q}$ of the unknown radial geodesic from **p** to **q** in (a) can be approximated using the known geodesics from **p** to **r** and **r** to **q**. The vector $\mathbf{u}_{p,r}$ (in normal coordinates at **r**) is transferred to the tangent plane at **p** using a 2D rotation with angle $\theta_{p,r}$, producing the red vector in (b). This vector is an approximation to $(\mathbf{u}_{p,q} - \mathbf{u}_{p,r})$ and can be added to $\mathbf{u}_{p,r}$ (c) to get the approximate result $\hat{\mathbf{u}}_{p,q}$.*

The final algorithm for approximating the exponential map can now be described. First, Dijkstra's algorithm is run from point **p** to generate, for each other surface point **q**, a piecewise-linear curve from **p** to **q** with vertices $\{\mathbf{p}_i\}$. These linear segments are then sequentially "lifted" into $\mathcal{T}_p$ by evaluating

$$\hat{\mathbf{u}}_{p,q} = \widehat{\mathbf{u}}_{p_0,p_1} + \sum_{i \geq 1} Rot2D(\theta_{p,p_i}) \cdot \widehat{\mathbf{u}}_{p_i,p_{i+1}} \tag{3}$$

which amounts to summing the 2D vectors produced by transforming each successive linear 3D segment into $\mathcal{T}_p$. Note that $\widehat{\mathbf{u}}_{p_i,p_{i+1}}$ is mapped directly into the tangent plane at **p** in Equation 3, rather than incrementally mapping to each previous tangent plane. The conditions described above on the tangent plane mapping hold regardless of the distance between **p** and $\mathbf{p}_i$. Transforming the vector through each previous tangent plane results in much higher total error.

Equation 3 approximates the exponential map by transforming 3D points into the 2D tangent plane at **p**. The magnitude of the resulting 2D vector approximates the geodesic distance with more accuracy than the value produced by Dijkstra's algorithm. The improvement comes from the use of vector addition, rather than the scalar addition used by Dijkstra's algorithm. This result is reminiscent of the improvement in Euclidean distance approximation observed when using *vector distance transforms* instead of *chamfer distance transforms* [Satherley and Jones 2001].

4.3 Implementation Details and Properties

The sampling rate of the underlying point set largely determines the accuracy in the discrete exponential map. The first restriction on sampling rate is that local geodesic neighbourhoods must be computable. We use a *k* nearest geodesic neighbour scheme, with *k* = 15. If mesh connectivity is unavailable, nearest Euclidean neighbours are assumed to be geodesic neighbours. This assumption is valid only if global sampling criteria hold [Dey and Goswami 2004]. In some cases, such as subdivision and implicit surfaces, additional samples can be generated automatically to resolve undersampling. Otherwise, algorithms that take a global approach to the neighbour-finding problem are necessary [Fleishman et al. 2005].

Related to the problem of computing geodesic neighbourhoods is that of computing the local discrete exponential map, $\widehat{exp}_p$. The angle-spreading technique of [Welch and Witkin 1994] introduces undesirable error. We assume that the length $|\mathbf{p}-\mathbf{q}|$ is a good estimate of geodesic distance and determine the normal coordinates by finding the angle θ_q between the vector $\vec{\mathbf{pq}}$ and $\mathcal{T}_p$, and then rotating $\vec{\mathbf{pq}}$ by θ_q around the axis $\vec{\mathbf{pq}} \times \mathbf{n}_p$. The rotation must always be in the direction of the normal, hence if $\mathbf{n}_p \cdot \mathbf{n}_q < 0$ the rotation angle is $\pi - \theta_q$.

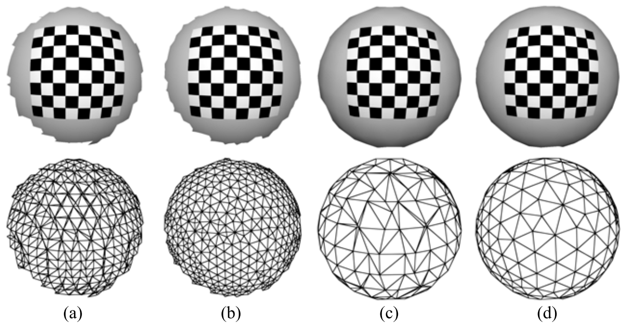


Figure 6: *The sampling of the underlying point sets (created by extracting the mesh vertices) has little effect on the decal shown in these images. The same number of points are used to sample the surface in (a) and (b). The ExpMap decal robustly handles the irregular sampling created by the sliver triangles in (a) and (c).*

Given adequate sampling, we have found the discrete exponential map to be very robust in practice. In particular, there is no requirement that the surface sampling be regular. Our interactive system is based on implicit surfaces, and we generate our point set by extracting the vertices of a marching cubes mesh. Marching cubes is known to produce very irregular vertex distributions, resulting in a worst-case sampling density much lower than the equivalent number of points evenly distributed. The resulting parameterization is visually indistinguishable from that computed with a more regular point distribution, even at moderate tessellation resolutions (Figure 6). The algorithm does assume that accurate surface normals are available. Noisy normals introduce local distortion but the algorithm remains stable. Noise in the point set, however, affects geodesic distances and causes the parameterization to quickly degenerate.

The discrete exponential map we describe is very efficient. Using a priority queue, the running time of Dijkstra's algorithm is $O(N \log N)$ in the number of surface samples. The discrete exponential map can be incrementally computed in-line with the Dijkstra propagation, hence the running time remains $O(N \log N)$.

Dijkstra's algorithm is known to be non-convergent. While we have not found any theoretical proof, we have some evidence that the discrete exponential map does converge. In numerical tests on a sphere, $\hat{\mathbf{u}}_{p,q}$ (Equation 2) converges quadratically to the analytic value of $\mathbf{u}_{p,q}$ as the distances between p,r and r,q go to zero. For longer piecewise-linear geodesic paths the numerical convergence is only linear, likely due to the accumulation of error at each approximation step. In general, the discrete exponential map appears to visually converge as the sampling rate is increased.

One property of exponential map parameterization is that developable surfaces (surfaces with zero Gaussian curvature) are parameterized with no distortion. Our discrete approximation reproduces this property, as seen in Figure 7.

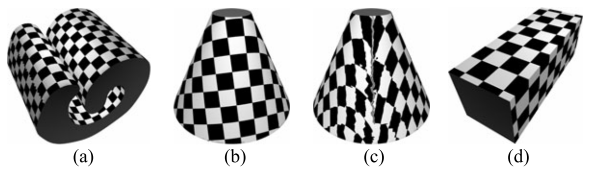


Figure 7: *Parameterizations generated for developable surface patches - (a) swiss roll, (b)/(c) cone front/back, and (d) box. Artifacts occur on the rear of the cone because the discrete approximation fails once the geodesic distance field collides with itself.*

5 Decal Parameterizations

Our interactive texturing system is based on local exponential map parameterizations which we call *ExpMap decals*. A basic ExpMap decal is defined by a *seed point* $\mathbf{p}$ on the surface and a geodesic radius r. The decal is generated by first running Dijkstra's algorithm to find an approximate geodesic disc with radius $r+\delta$. The δ value is necessary to ensure that the disc of radius r is contained within the decal, since the particular discretization may otherwise result in clipping. We use the largest neighbour distance as δ. This approximate geodesic disc is then parameterized using the discrete exponential map as described in Section 4.2. Finally the parameterization is scaled by $1/\sqrt{2}r$ and translated by $(0.5,0.5)$, so that the "geodesic square" inscribed in the disc lies at $[0,1] \times [0,1]$ in parameter space.

If only the unit square in texture space is required, then parameterizing the geodesic disc results in a significant amount of "wasted" space, particularly for large decals. This can be avoided by computing the discrete exponential map in-line with Dijkstra's algorithm, and truncating the propagation when $max(u,v) > r/\sqrt{2}+\delta$.

In our interactive system, the surface is rendered as a texture-mapped triangle mesh. Decals are stored as independent local parameterizations of portions of the surface, similar to [Praun et al. 2000]. Compositing is performed dynamically using alpha-blending. This may involve storage and rendering of hundreds of decal parameterizations, which is acceptable for interactive editing but less so if many decaled objects are part of a complex scene. For this case the decals can be baked into a global parameterization or octree texture.

5.1 Comparison to Other Mesh Parameterizations

We have compared our ExpMap decals with those generated using a variety of automatic mesh parameterization techniques. The decal creation procedure is the same as for ExpMap decals - we segment a region of the mesh that approximates a geodesic disc, and then interactively parameterize this disc.

We first compared our results with several mesh parameterization algorithms that require a boundary. In our case it is relatively simple to map the geodesic disc to a circular boundary. We tried the shape-preserving weights [Floater 1997], geodesic weights [Lee et al. 2005], and intrinsic weights [Desbrun et al. 2002]. However, in all of these cases we observed significantly higher distortion, likely introduced by the boundary mapping. In addition, the distortion is not frame-coherent, so while the user drags the decal across the surface it randomly changes size and orientation.

Better results were found with the natural conformal map [Desbrun et al. 2002], which does not require a boundary mapping and hence has significantly less distortion. However, if only two vertices are constrained (the minimum necessary to define a unique conformal solution), the orientation and scaling of the decal are not frame coherent as it is dragged across the surface or the surface is resampled. We found that if we constrained all of the vertices in the neighbourhood of the decal seed point, frame coherency was improved significantly, although some artifacts can still be seen.

We have observed that conformal decals are identical to ExpMap decals only on developable surfaces. On a sphere the "edges" of the square are similar, but the internal distortion is different (conformal decals are less uniform). However, in many relatively simple cases the square edges are not as well-preserved (Figure 8). Because ExpMap decals are based on geodesics, the decal distorts when the geodesics distort. Conformal decals use global optimization to spread the distortion out across the decal, producing decals with edges that we feel are much less "square" than ExpMap decals.

The problem is magnified during interaction. For example, on the cylinder in Figure 8b, the conformal decal begins to distort before the edge of the texture crosses the edge of the cylinder. This behavior is very unintuitive. Another problem is that the conformal parameterization is not necessarily contained within the geodesic disc. This can result in the decal being clipped (Figure 8c).

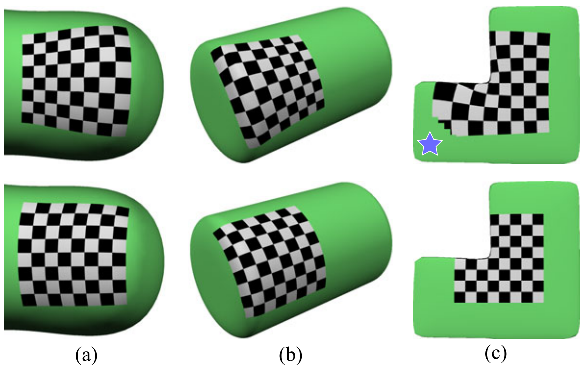


Figure 8: *Comparison between conformal (top) and ExpMap (bottom) decals. In these cases the global distortion minimization of the intrinsic conformal parameterization results in decals which are less "square" than the ExpMap decals. In (c), the conformal parameterization is clipped because it extends beyond the boundary of the decal (near the blue star).*

5.2 Hybrid Decals

Distortion in ExpMap decals is governed by the behavior of the set of radial geodesics originating at the seed point, which are in turn largely dependent on surface curvature. Variations in surface curvature cause the radial geodesics to spread apart or come together, which in turn distorts the decal. In theory this distortion happens smoothly, as in the analytic sphere example (Figure 4). However, the "greedy" nature of the discrete exponential map can produce artifacts, and even local foldovers, if the decal has high distortion. Generally this occurs with decals that cover undersampled regions of the surface with significantly varying curvature. Removing these artifacts is challenging - the algorithm has only local information, however global knowledge is required to spread out the distortion.

A global approach can be constructed by applying constrained conformal parameterization to ExpMap decals.

First, we identify regions of the decal that are distorted by defining a distortion metric

$$\epsilon_i = \max_j \left| \frac{|\mathbf{u}_i - \mathbf{u}_j|^2}{|\mathbf{p}_i - \mathbf{p}_j|^2} - 1 \right| \tag{4}$$

where $\mathbf{p}_j$ are the neighbours of point $\mathbf{p}_i$, and $\mathbf{u}_j$ are the associated texture coordinates. If ϵ_i is larger than a user-controllable threshold, the $\mathbf{u}$ of $\mathbf{p}_i$ and all of it's neighbours are discarded. Then a conformal parameterization is computed for these points, with all the "good" points constrained to their exponential map parameter values. The result is a hybrid ExpMap / Conformal decal (Figure 9).

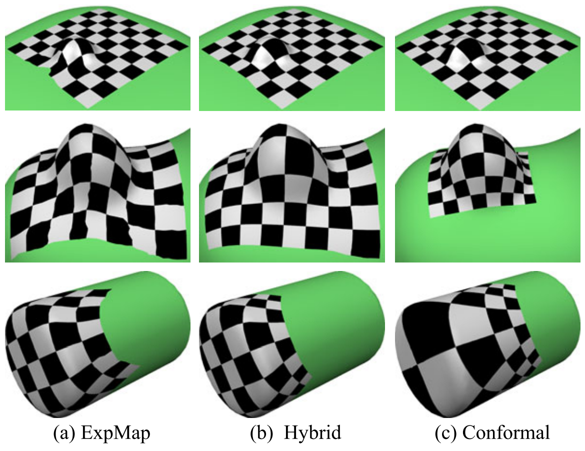


Figure 9: *Comparison between ExpMap (a), Hybrid (b), and Conformal (c) decals. The extreme variation in decal size (middle row) and distortion (bottom row) found in some cases with Conformal decals is mitigated with Hybrid decals.*

Hybrid decals also can repair the global foldovers that occur in the analytic exponential map at at singularities in the geodesic distance field, where there is no unique geodesic curve from the seed point. In some cases, the foldover is very small and may not appear depending on the discretization. Hybrid decals easily repair these foldovers. Global foldovers inevitably occur if the decal covers a large region with many changes in the sign of Gaussian curvature (Figure 10). Hybrid decals are not always useful in these cases because even though the global foldovers are avoided, there can still be very high distortion (Figure 10d) and even the conformal parameterization can have foldovers [Desbrun et al. 2002]. However, generally these regions are large enough that the metaphor of sticking a square image onto the surface is no longer applicable, so this issue is essentially beyond the scope of the decaling interface.

We also note that while Hybrid decals do generally reduce distortion, minor distortion is often unnoticeable when more realistic textures with high-frequency details are used. Hybrid decals were not required to create any of the examples in this paper.

5.3 Partial Decals

Pedersen's patchinos were limited to "regions that are relatively flat except for a hole or branch extending outwards" [Pedersen 1996], because when passing over these features the patchino would become highly distorted. ExpMap decals suffer from the same problem, as seen in Figure 11a. Pedersen also noted that this situation could be avoided if the patchino contained a hole which passed

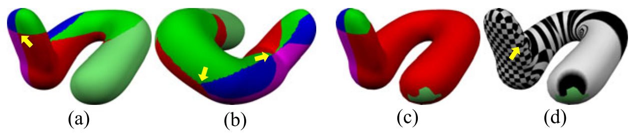


Figure 10: *The ExpMap decal in (a) is textured with an image that has a different color assigned to each of the 4 quadrants. The seed point is marked with an arrow. The varying curvature of the shape produces several foldovers, marked with arrows in (b). The hybrid decal appears more uniform in (c), however the distortion is very high and there are some small foldovers at the edge of the constraint region (d).*

around the feature. His system could not support this because the patchino optimization procedure required a complete mass-spring mesh, however it is trivial to implement with ExpMap decals.

To create partial decals, the unwanted points are removed from the Dijkstra computation and left unparameterized. In our system we implement this by simply halting the Dijkstra propagation at points whose absolute Gaussian curvature is larger than a user-defined threshold (Figure 11b). Other alternatives could include halting at creases, or based on boundaries painted by a user. Partial decals can also be combined with the Hybrid decal technique described in the previous section to parameterize the features without introducing distortion into the flatter portion of the decal. Note that the seed point for the decal cannot reside in the hole region.

Partial decals provide an example of a very interesting property of the discrete exponential map approximation described in Section 4.2. If the above procedure were performed using actual geodesics (such as those described by [Surazhsky et al. 2005]), the result would not be as shown in Figure 11b. The actual geodesic wave-fronts would pass around the “hole” and then collide behind it, creating a texture discontinuity. However, because the incremental vector addition is done in the tangent space of the exponential map, the “geodesic curves” on the surface actually pass *through* the feature - as if it were replaced by a smooth surface. This property may have applications for hole-filling in point sets. Note, however, that because the path taken from the seed point to the points behind the hole is less direct, the approximation error is larger.

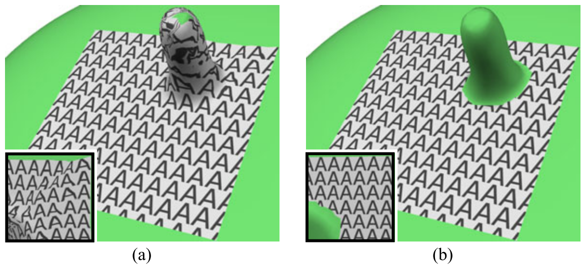


Figure 11: *In the case of a relatively smooth surface with a high-frequency feature, the decal is necessarily distorted (a), both on the feature as well as behind it. By simply halting the vector propagation based on curvature magnitude, a hole is created as the decal passes around the feature (b). The distortion introduced behind the feature is also avoided (insets).*

6 Interactive Techniques

Our decaling interface is based on the user interacting with the decal via a simple 3D control (Figure 2b,top). The decal is moved by dragging the center point across the surface, rotated with the circle, and scaled with the outer point. The parameterization is automatically recomputed as needed. To fit a decal around a set of points $\{\mathbf{p}_i\}$ on the surface, a seed point is chosen and then Dijkstra's algorithm is run until all $\mathbf{p}_i$ have been reached. If the points $\{\mathbf{p}_i\}$ have been created by projecting a 2D screen-space curve onto the surface, we use the projection of the 2D bounding box centroid as the seed point. For other sets of 3D points, a reasonable guess is the point closest to the center of the bounding box of $\{\mathbf{p}_i\}$.

To implement user-interface tools, it is useful to be able to map between the 3D surface and 2D parameter space for points not in the initial point set. A 2D parameter **u** can be mapped to 3D **q** on the surface by first finding the Delauney triangulation of the 2D points, then using barycentric interpolation in the triangle containing **u**. Mapping from 3D **q** to 2D **u** would ideally be done by adding the new point to the point set and recomputing the decal parameterization, however this is too expensive for interactive use. A quick approximation is to find the nearest neighbour to **q** and locally propagate the parameterization from this neighbour using Equation 2.

6.1 Decal Deformation

It is often useful to be able to deform a decal, either to match image features to surface features, or to correct for some unwanted distortion. One approach is to use 2D image deformations, modifying the texture image to better fit the parameterization. However, these deformations are not interactive if the image has high resolution. Image deformation also prevents re-use of the same texture image for multiple decals with different deformations. Instead, we deform the decal parameterization.

If the image deformation ω is desired, then the inverse deformation ω^{-1} must be applied to the parameterization. Unfortunately many common deformation functions are difficult to invert. In addition, we must deform the entire decal parameterization, not only the region inside the unit square $[0,1] \times [0,1]$, to minimize foldovers. To handle this issue we use a ω^{-1} function that has global support

Our deformation tool is based on C^2 variational scattered-data interpolation, also known as *thin-plate splines* or *radial basis functions*. We have a set of point constraints between sources (u,v) and destinations (u',v'). Two 2D variational warps are fitted, one which maps each (u',v') to the correspond u value, and another for the v value. This warp is evaluated for each point in the parameterization. The process is similar to the mesh deformation described by [Botsch and Kobbelt 2005]. Since the variational solution is based on point constraints, both lattice deformations (Figure 2) and feature alignment (Figure 12) are supported.

6.2 Surface Vector Graphics

A decal can be used as a canvas for 2D vector graphics. A simple line element can be generated between two points on the surface by creating a decal originating at one point and with a decal radius equal to the geodesic distance to the second point. The line width is kept uniform by scaling linearly based on the decal radius. Defining a line element this way ensures that the line lies on a geodesic, intuitively the shortest curve between the two points. If the geodesic crosses regions of the surface with widely varying curvature,

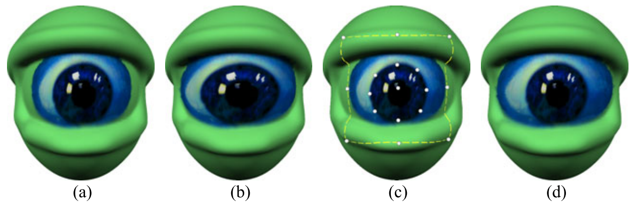


Figure 12: *Decal deformation. In (a), the decal image is too small to cover the eyeball. Using the lattice deformation shown in Figure 2 results in a stretched pupil (b). Placing extra constraint points around the pupil (c) produces the desired result (d).*

distortion in the decal may produce a non-uniform line width. In this case distortion can be reduced by subdividing the geodesic and placing each segment into a separate decal.

Geodesic line elements can be used as a building block for other surface vector elements. We create surface curves by projecting the vertices of screen-space curves onto the surface and then connecting them with line elements (Figure 13). One advantage of this approach is that portions of the surface which are occluded can still be painted, unlike the projective texturing found in 3D painting tools.

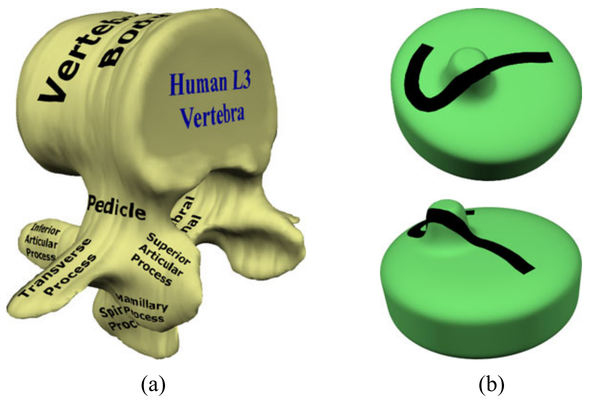


Figure 13: *In (a), a vertebra model extracted from a volume dataset is annotated using vector text. In (b), a screen space stroke (top) is mapped to a surface by connecting projected points with surface vector lines. The vector lines are geodesics and hence are continuous across the occluded area (bottom).*

7 Dynamic Surfaces and Animation

Most texture-mapping schemes assume that the underlying surface is static. If the surface changes, global parameterizations must be recomputed, frequently leading to significant changes in the texture space distortion that make the previous textures useless. Even simple remeshing can be problematic. Decal parameterizations fare much better when the underlying surface changes. Since the local parameterizations are dynamically generated, the decal seed points are simply moved to the new surface. As shown in Figure 6, decals are quite stable under remeshing, so in areas where the surface does not change the decal textures will be preserved. In the deformed regions, the decals will flow onto the new surface in a consistent and predictable manner.

Decals can also be used to texture animated surfaces. As long as the visible areas of the decals have low distortion, the results will be relatively frame coherent because (assuming adequate sampling) the decal parameterization is only influenced locally by changes in the surface. Essentially, if a portion of the surface under a decal is modified, only the geodesics passing across that area will change. The rest of the decal remains unaffected. This local influence helps greatly with frame coherence and also is desirable for interaction, since it is very disorienting if the decal unexpectedly changes as it is dragged across the surface, or the surface is deformed. Note that local influence does not hold if the modified area contains the decal seed point - in that situation, all geodesics are changed, hence so is the entire decal.

Because decals preserve texture across remeshing and surface deformation, they are an alternative for texturing animated implicit surfaces. Our modeling system is based on hierarchical skeletal implicit surfaces [Wyvill et al. 1999], where the surface is a composition of simpler shapes. Each shape has a local reference frame, so each decal is first associated with the nearest shape and stored at local coordinates in the reference frame. The decal position for any frame is then found by performing a gradient walk from these local coordinates to the surface. This approach has the benefit of allowing the user to jump to any frame. Animation with mass-spring decals would require the user to wait for simulation of all the in-between steps.

In practice, frame-coherence is largely determined by the amount of distortion in the decal. With high distortion the parameterization is likely to “pop” from frame to frame in the distorted regions. The popping occurs because the sampling rate is too low in the distorted regions, so denser and more regular sampling can help to some extent. However, we have found that replacing a highly-distorted decal with several smaller, less-distorted decals provides better results. Another option is to compute a mesh and use a more robust but slower geodesic computation [Surazhsky et al. 2005].

One benefit of animating with decals is that difficult situations such as topology change can be handled. Figure 14 shows several frames from an animation where two objects are blended. Decals can also be used as a canvas for 2D animation techniques. Video textures and sprite animations [Lefebvre et al. 2005] are easily applied, and vector animations can be created by keyframing the control points of vector elements (Section 6.2).

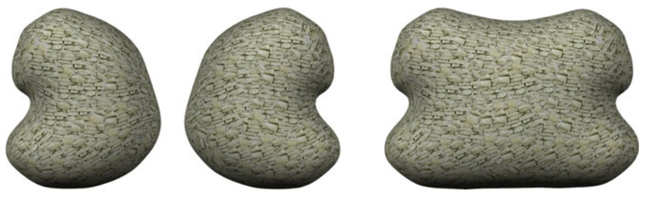


Figure 14: *Decals can preserve texture during topological change. The decals on these the two implicit surfaces smoothly slide together in the blending area. The rest of the texture is unaffected.*

8 Results

Combined with a digital camera, our decaling system provides a quick and easy-to-use interface for texturing 3D models. Using an existing model and a few snapshots, the dog in Figure 15 was composited in a few minutes. The base fur texture was generated by uniformly distributing fur decals across the surface, essentially creating a lapped texture [Praun et al. 2000]. Lapping decals can also be used to achieve NPR-like effects, such as in Figure 15b. Rendering NPR strokes with decals has the advantage that the strokes then automatically curve with the surface.

An example of modeling a real-world object is shown in Figure 1. A rough 3D model was generated using sketch-based modeling software, and then 22 snapshots of the clay statue were taken. A base texture was generated using the lapping technique, and then relevant features were cut out of each image and attached to the model. The entire process, from clay statue to textured 3D model, took several hours, with the most time-consuming step being the manual color-balancing necessary to compensate for the poor color reproduction of the digital camera.

Figure 16 shows a textured implicit gremlin model. The textures were taken from a variety of sources, including photographs of a frog, a painting, and the author. These textures were cleaned up and re-colored using image-processing software, and then pasted onto the suface. Many of the textures are re-used multiple times. For example, each finger and toe uses the same set of decals, although they are deformed to better fit the particular geometry.

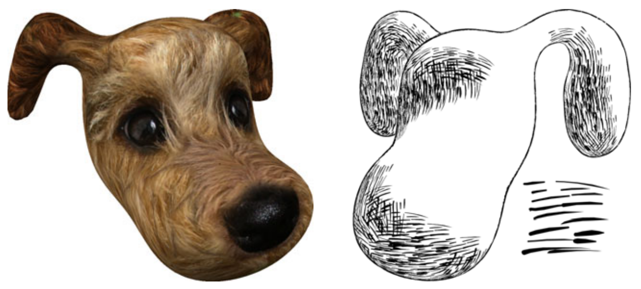


Figure 15: *In (a), an irregular piece of fur texture is automatically lapped to create a base fur texture on a model of a dogs head. The eyes, a nose, and a few additional pieces of fur were cut from other photographs. Placing them on the surface takes only a few minutes. In (b), an image of a few strokes is lapped. The decals are aligned with the first principal curvature, and decal visibility is modulated based on a 3D light position to produce an NPR effect.*

9 Conclusions and Future Work

We have described an interactive texture mapping interface based on decals generated using a new discrete exponential map approximation. Perhaps the most obvious benefit of this approach is that sampled surfaces can be textured without a global base parameterization. Surfaces created in our implicit modeling system can be immediately textured. Furthermore, if the designer finds that the surface must be modified, extensive changes can be made without losing the textures applied so far.

With our interface, the designer uses high-level tools to composite and manipulate decals, and is not forced to deal with the underlying parameterizations. ExpMap decals enable new texturing tools, such as surface vector-graphics, selection of of curved regions, and decals with holes, which were not possible using the mass-spring technique of [Pedersen 1996]. In our experience, ExpMap decals generally provide predictable and consistent results, and address difficult problems such as animating implicit surfaces and topology change.

We have found this decaling tool to be a fluid and efficient way to texture surfaces, particularly for those who are not skilled at texture painting. Even for texture painters, the ability to easily mix-and-match from existing images makes decaling a useful addition to the texture design toolbox.

We anticipate a wide range of applications for ExpMap decal parameterizations in interactive tools. Other mapping techniques such as bump and displacement mapping can benefit from an interactive decaling interface. Interactive re-meshing or re-sampling of point sets is easily implemented by sampling in the decal parameterization. When combined with the partial decals described in Section 5.3, re-sampling in parameter space also supports repair of holes or removal of unwanted features. Decals may be useful in mesh and point-set modeling contexts for implementing interactive tools such as surface deformation and geometry cut-and-paste. We are also exploring application of decal parameterization to high-quality tessellation of implicit surfaces.

Another benefit of ExpMap decals is that, because the parameterization is dynamically generated, no UV-coordinates need be stored with the geometry. Network transmission of decals is very efficient, once the decal images are transferred then only the seed points and frames must be sent. Since ExpMap decals are stable under remeshing, this may be particularly beneficial for progressive geometry.

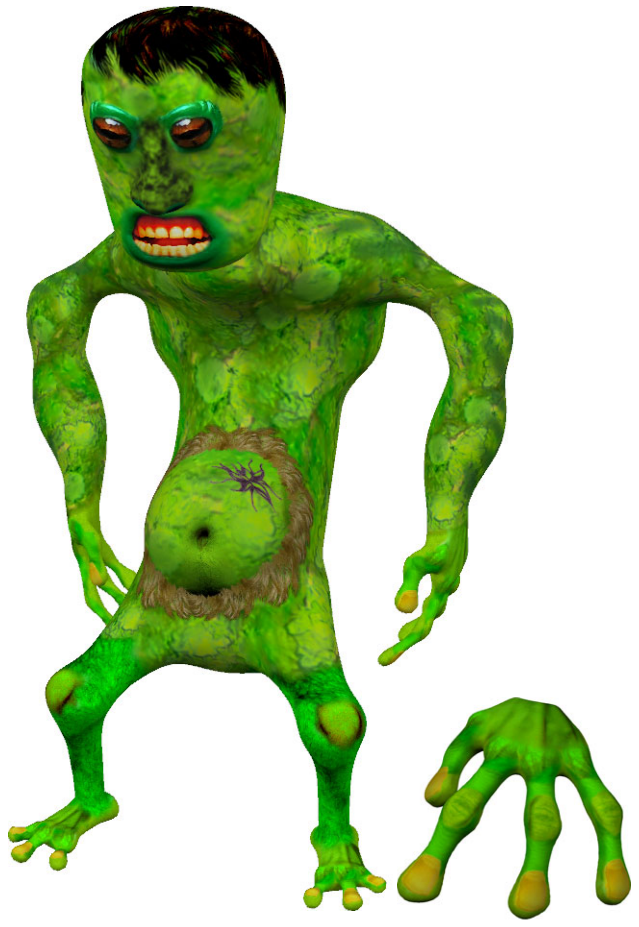


Figure 16: *The decal texture for this implicit gremlin model was created using 19 different images cut out of various photographs and then manipulated using 2D image editing software. The texture consists of 392 decals, although only 78 were placed manually, including 24 for the hand. The rest were generated using texture lapping or by duplicating the arm and leg.*

References

ALEXA, M., KLUG, T., AND STOLL, C. 2003. Direction fields over point-sampled geometry. In *Proceedings of WSCG*.

BENSON, D., AND DAVIS, J. 2002. Octree textures. *ACM Transactions on Graphics 21*, 3, 785–790.

BLINN, J., AND NEWELL, M. 1976. Texture and reflection in computer generated images. *Communications of the ACM 19*, 10, 542–547.

BOTSCH, M., AND KOBBELT, L. 2005. Real-time shape editing using radial basis functions. *Computer Graphics Forum 24*, 3, 611–621.

CHEEGER, J., AND EBIN, D. G. 1975. *Comparison Theorems in Riemannian Geometry*. North-Holland Mathematical Library.

DEBRY, D., GIBBS, J., PETTY, D. D., AND ROBINS, N. 2002. Painting and rendering textures on unparameterized models. *ACM Transactions on Graphics 21*, 3, 763–768.

DESBRUN, M., MEYER, M., AND ALLIEZ, P. 2002. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum 21*, 3, 383–392.

DEY, T. K., AND GOSWAMI, S. 2004. Provable surface reconstruction from noisy samples. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, 330–339.

DIJKSTRA, E. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 269–271.

DO CARMO, M. P. 1976. *Differential Geometry of Curves and Surfaces*. Prentice Hall.

EBERT, D. S., Ed. 2002. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann. ISBN 1558608486.

FLEISHMAN, S., COHEN-OR, D., AND SILVA, C. T. 2005. Robust moving least-squares fitting with sharp features. *ACM Trans. Graph. 24*, 3, 544–552.

FLOATER, M. S., AND REIMERS, M. 2001. Meshless parameterization and surface reconstruction. *Comp. Aided Geom. Design 18*, 77–92.

FLOATER, M. 1997. Parametrization and smooth approximation of surface triangulations. *Comp. Aided Geom Design 14*, 231–250.

GRIMM, C. 2004. Parameterization using manifolds. *International Journal of Shape Modeling 10*, 1, 51–80.

GU, X., AND YAU, S.-T. 2003. Global conformal surface parameterization. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 127–137.

HANRAHAN, P., AND HAEBERLI, P. E. 1990. Direct wysiwyg painting and texturing on 3d shapes. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, vol. 24, 215–223.

KIMMEL, R., AND SETHIAN, J. 1998. Computing geodesic paths on manifolds. *Proc. of National Academy of Sci. 95*, 15 (July), 8431–8435.

KRAEVOY, V., SHEFFER, A., AND GOTSMAN, C. 2003. Matchmaker: Constructing constrained texture maps. *ACM Transactions on Graphics 22*, 3, 326–333.

LEE, H., TONG, Y., AND DESBRUN, M. 2005. Geodesics-based one-to-one parameterization of 3d triangle meshes. *IEEE MultiMedia 12*, 1, 27–33.

LEFEBVRE, S., HORNUS, S., AND NEYRET, F. 2005. Texture sprites: Texture elements splatted on surfaces. In *ACM-SIGGRAPH Symposium on Interactive 3D Graphics (I3D)*.

LÉVY, B. 2001. Constrained texture mapping for polygonal meshes. In *Proceedings of ACM SIGGRAPH 2001*, 417–424.

MAILLOT, J., YAHIA, H., AND VERROUST, A. 1993. Interactive texture mapping. In *Proceedings of SIGGRAPH 93*, 27–34.

MITCHELL, J. 2000. *Geometric Shortest paths and network optimization*. Elsevier Science, ch. Handbook of Computational Geometry, 633–702.

PEDERSEN, H. K. 1995. Decorating implicit surfaces. In *Proceedings of SIGGRAPH 95*, 291–300.

PEDERSEN, H. K. 1996. A framework for interactive texturing operations on curved surfaces. In *Proceedings of SIGGRAPH 96*, 295–302.

PORTER, T., AND DUFF, T. 1984. Compositing digital images. In *Proceedings of SIGGRAPH 84*, vol. 18, 253–259.

PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2000. Lapped textures. In *Proceedings of ACM SIGGRAPH 2000*, 465–470.

SATHERLEY, R., AND JONES, M. 2001. Vector-city vector distance transform. *Computer Vision and Image Understanding 82*, 3, 238–254.

SHEFFER, A., LÉVY, B., MOGILNITSKY, M., AND BOGOMYAKOV, A. 2005. Abf++: fast and robust angle based flattening. *ACM Trans. Graph. 24*, 2, 311–330.

SORKINE, O., COHEN-OR, D., GOLDENTHAL, R., AND LISCHINSKI, D. 2002. Bounded-distortion piecewise mesh parameterization. In *Proceedings of IEEE Visualization '02*, 355–362.

SURAZHSKY, V., SURAZHSKY, T., KIRSANOV, D., GORTLER, S. J., AND HOPPE, H. 2005. Fast exact and approximate geodesics on meshes. *ACM Transactions on Graphics 24*, 3, 553–560.

TIGGES, M., AND WYVILL, B. 1999. A field interpolated texture mapping algorithm for skeletal implicit surfaces. In *Computer Graphics International*, 25–32.

WELCH, W., AND WITKIN, A. 1994. Free-form shape design using triangulated surfaces. In *Proceedings of SIGGRAPH 94*, 247–256.

WYVILL, B., GUY, A., AND GALIN, E. 1999. Extending the csg tree. warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum 18*, 2, 149–158.

ZHANG, E., MISCHAIKOW, K., AND TURK, G. 2005. Feature-based surface parameterization and texture mapping. *ACM Transactions on Graphics 24*, 1, 1–27.

ZHOU, K., WANG, X., TONG, Y., DESBRUN, M., GUO, B., AND SHUM, H.-Y. 2005. Texturemontage: Seamless texturing of arbitrary surfaces from multiple images. *ACM Transactions on Graphics 24*, 3, 1148–1155.

ZWICKER, M., PAULY, M., KNOLL, O., AND GROSS, M. 2002. Pointshop 3d: An interactive system for point-based surface editing. *ACM Transactions on Graphics 21*, 3, 322–329.

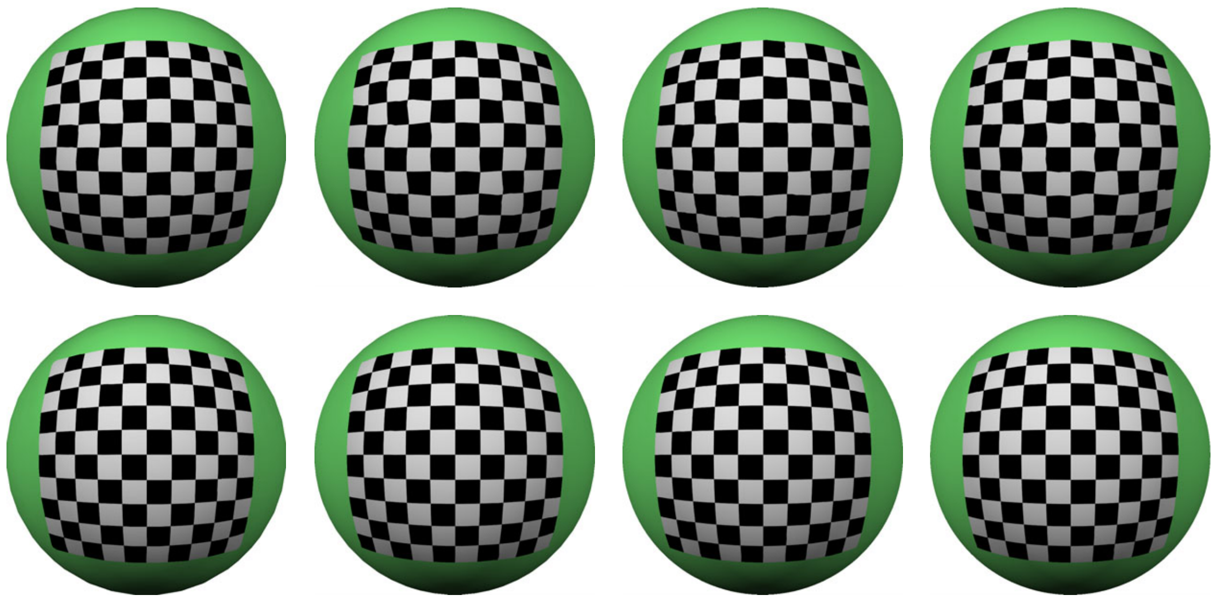


Figure 17: *Decal parameterizations with geodesic distance computed using Dijkstra's Algorithm (top) and Discrete Exponential Map (bottom). The underlying point set is extracted from a marching cubes mesh at voxel-resolution-per-edge of 30, 50, 100, 200 (left to right), with 15 neighbours for each point. The Discrete Exponential Map clearly exhibits better convergence behavior for local geodesic distances than Dijkstra's Algorithm.*

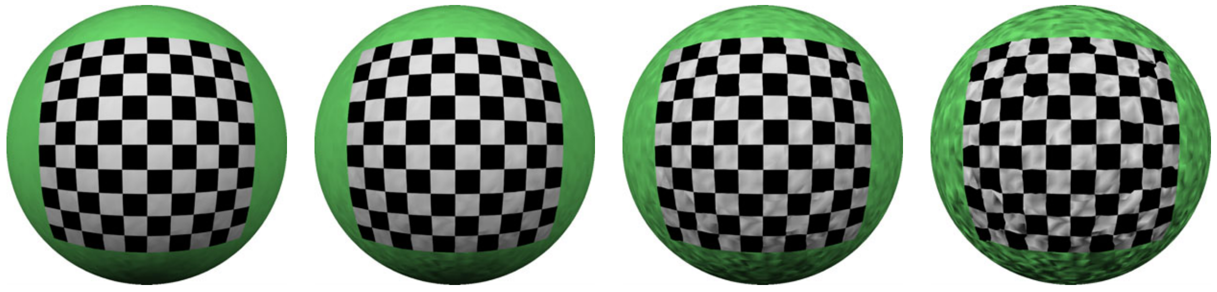
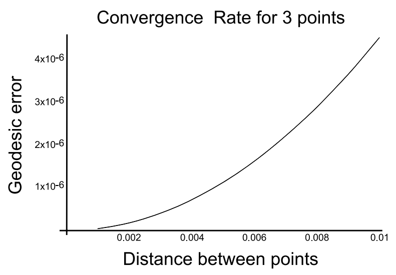
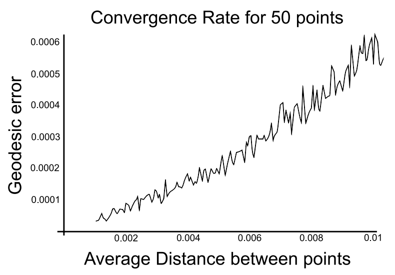


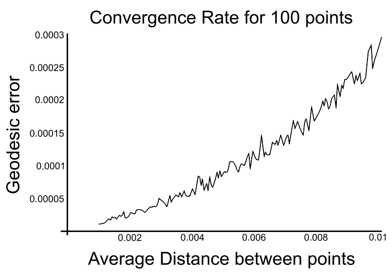
Figure 18: *Decal parameterizations using the discrete exponential map on point sets with noise in the normal vectors. The underlying point set is the same as in the 100-resolution sphere in Figure 1, however random noise has been added to the normal vectors. The maximum noise is (left to right) 5%, 10%, 25%, and 50%. Innaccurate normal vectors introduce very local distortion, however the overall shape of the parameterization is preserved.*



(a)

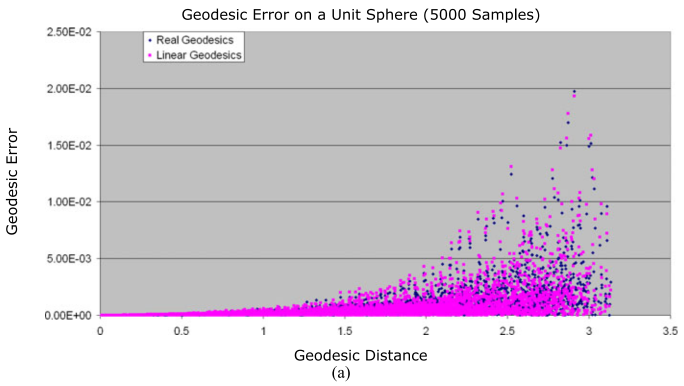


(b)

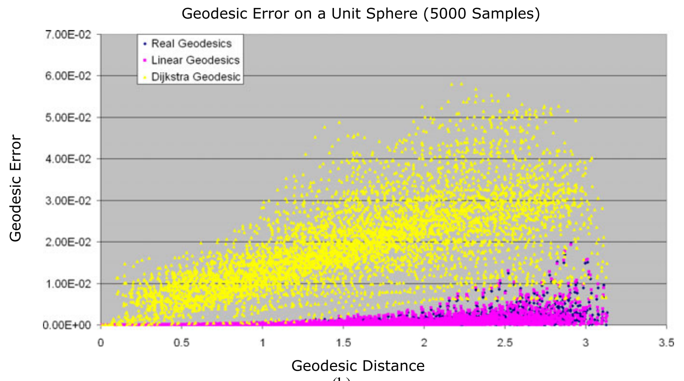


(c)

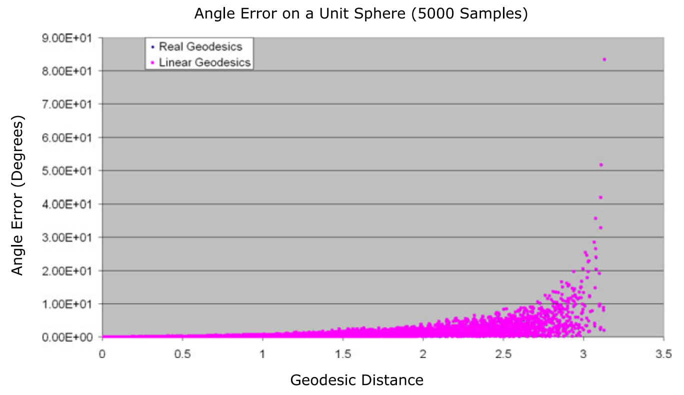
Figure 19: *Results of a numerical convergence test computed in Mathematica. Geodesic distances computed with the discrete exponential map are compared with analytic geodesic distances on a sphere. In (a), the base-case of 3 points is tested. As the distance between the points (X axis) increases, the geodesic error (Y axis) increases quadratically. In (b) and (c), a similar test is run using piecewise-linear approximate geodesics with 50 and 100 segments, respectively. The intermediate points are initialized on the geodesic and then randomly offset along the surface by up to a maximum geodesic distance (X axis). Each data point is the average of 10 runs. In these cases the geodesic error increases linearly with the average offset distance. The relationship between number of data points and offset distance wrt geodesic error is more complex - generally when increasing the number of data points, the offset distance must be decreased for error to decrease.*



(a)



(b)



(c)

Figure 20: *Approximation error at each point for the discrete exponential map algorithm measured on a sphere regularly sampled with 5000 points, computed using 32-bit floating point. In (a) and (b), points are plotted along the X axis based on actual geodesic distance from the seed point (at the "north pole"), and the Y axis indicates absolute geodesic error. In (a), the discrete exponential map is computed with analytic local geodesics ("Real Geodesics") and linear local geodesics. Neither is consistently more accurate. In (b), geodesic distance computed with Dijsktra's algorithm is also plotted, showing significantly higher error even at small distances. In (c), the polar geodesic angle error is plotted on the Y axis. A similar pattern to (a) is observed.*

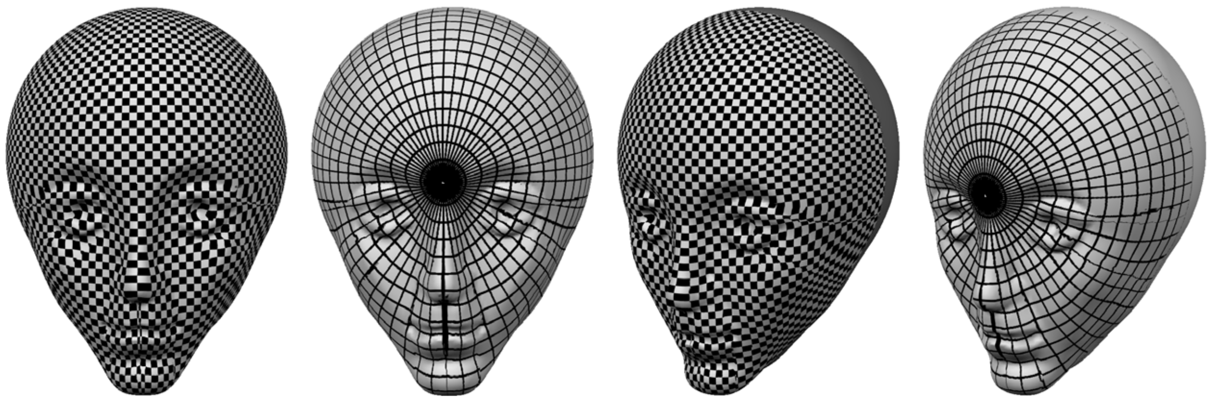


Figure 21: *Global exponential map parameterization generated for a complex shape. The seed point is at the middle of the bridge of the nose. Compression regions trail off the edges of the eyes and nose, and the parameterization is stretched at tip of nose, but otherwise the result is quite regular and un-distorted.*

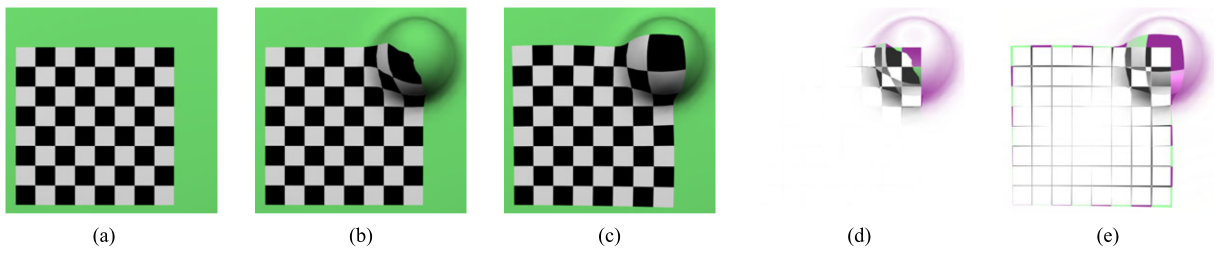


Figure 22: *First (a) and last frames from a simple animation of a bump rising out of a flat surface with ExpMap (b) and conformal (c) decals. The ExpMap decal exhibits significantly more frame coherence because the parameterization is stable outside the regions of the mesh that are changing, as shown by the difference image in (d). The conformal decal (e) distorts globally as the bump rises. This distortion oscillates rapidly as the decal mesh changes shape. The ExpMap decal also oscillates rapidly, but only in the deforming region. Also, the oscillation in the ExpMap decal is due to inaccuracy in the approximation - a better geodesic approximation would reduce or remove the incoherence completely.*

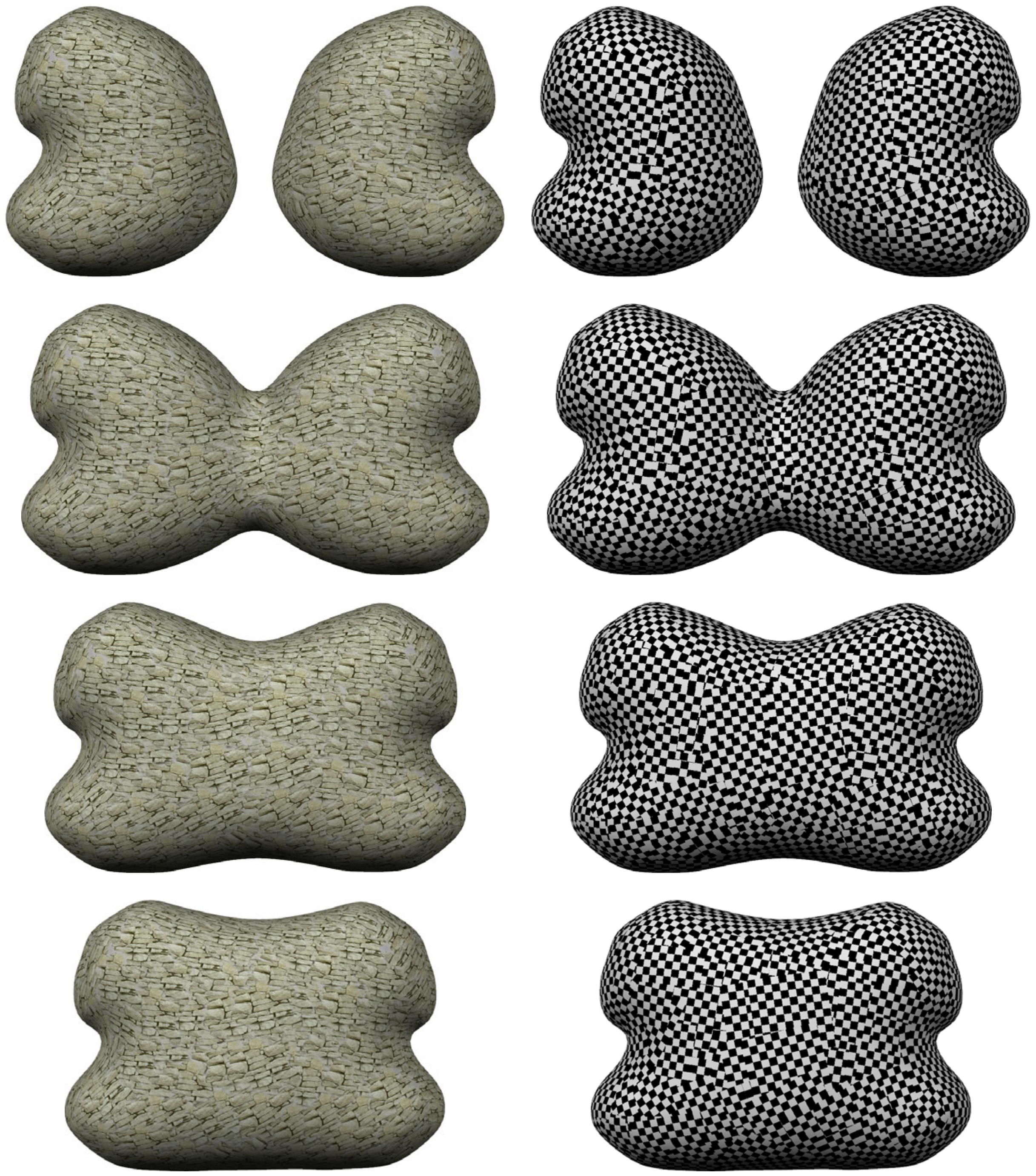


Figure 23: *Snapshots from an animation of two implicit surfaces blending, where each surface is textured with lapped decals. The left column shows the lapped textures, the right column shows the same decals with a checkerboard texture. The decals smoothly flow into the blending region, and appear to slide together in the video. Only the decals covering the portion of the changing surface are affected, decals on the far sides of each object are stationary. There is very little distortion in any particular decal, as can be seen from the checkerboards.*

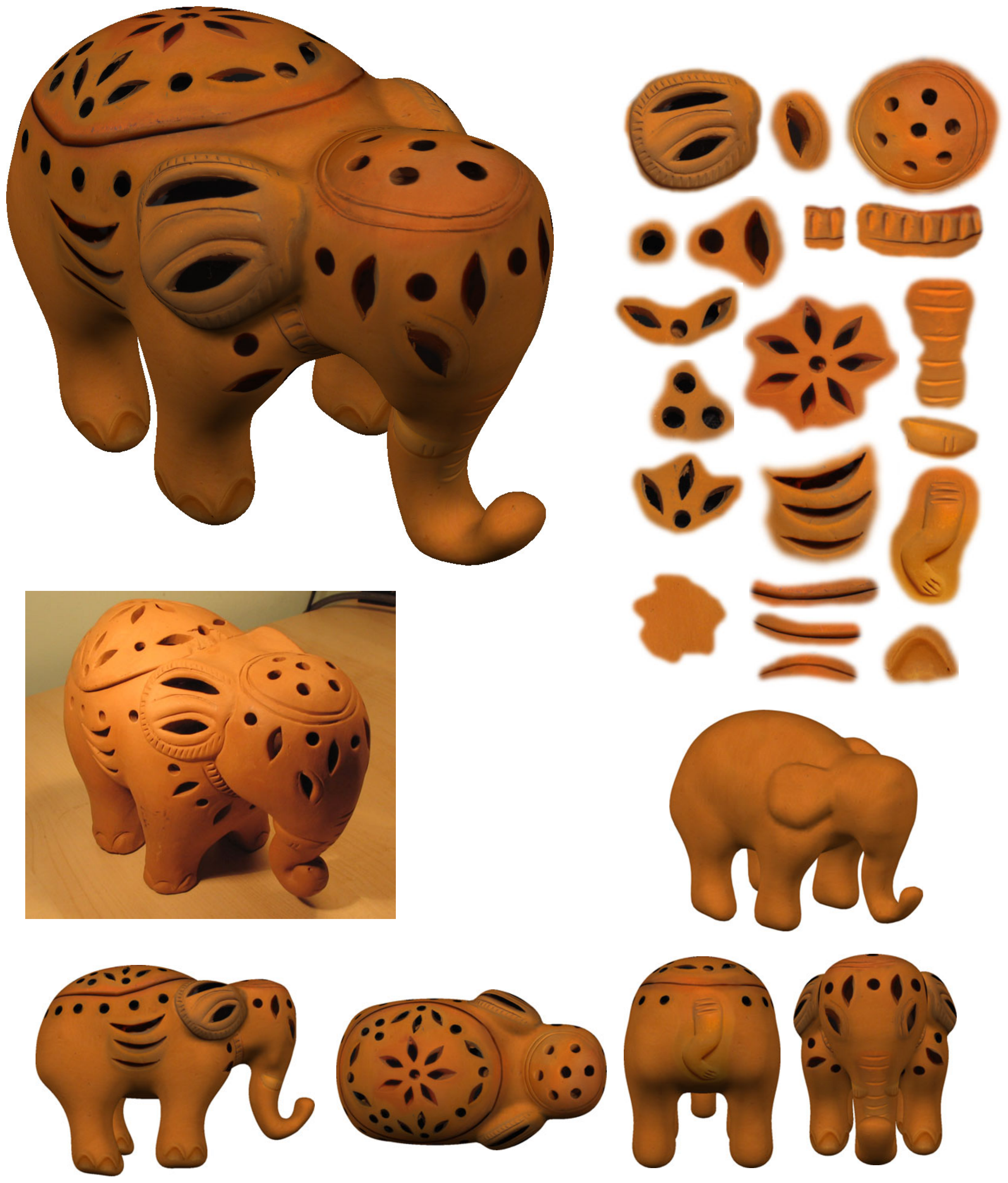


Figure 24: *An implicit model of the clay elephant statuette in the photograph was created with sketch-based modeling software in under an hour. Then, photographs of the statuette were taken and from those photographs a set of feature textures were extracted (upper right). A base texture was also extracted, which was lapped to create a base model. Then the feature textures were interactively placed over the base model. Texture placement was very quick, the total time including taking photographs and cutting out the relevant features was approximately an hour. However, low lighting conditions and "automatic color balancing" hardware on the digital camera resulted in each photograph (and it's feature textures) having a slightly different color balance from the base texture and eachother. Manual color balancing took an additional two to three hours.*

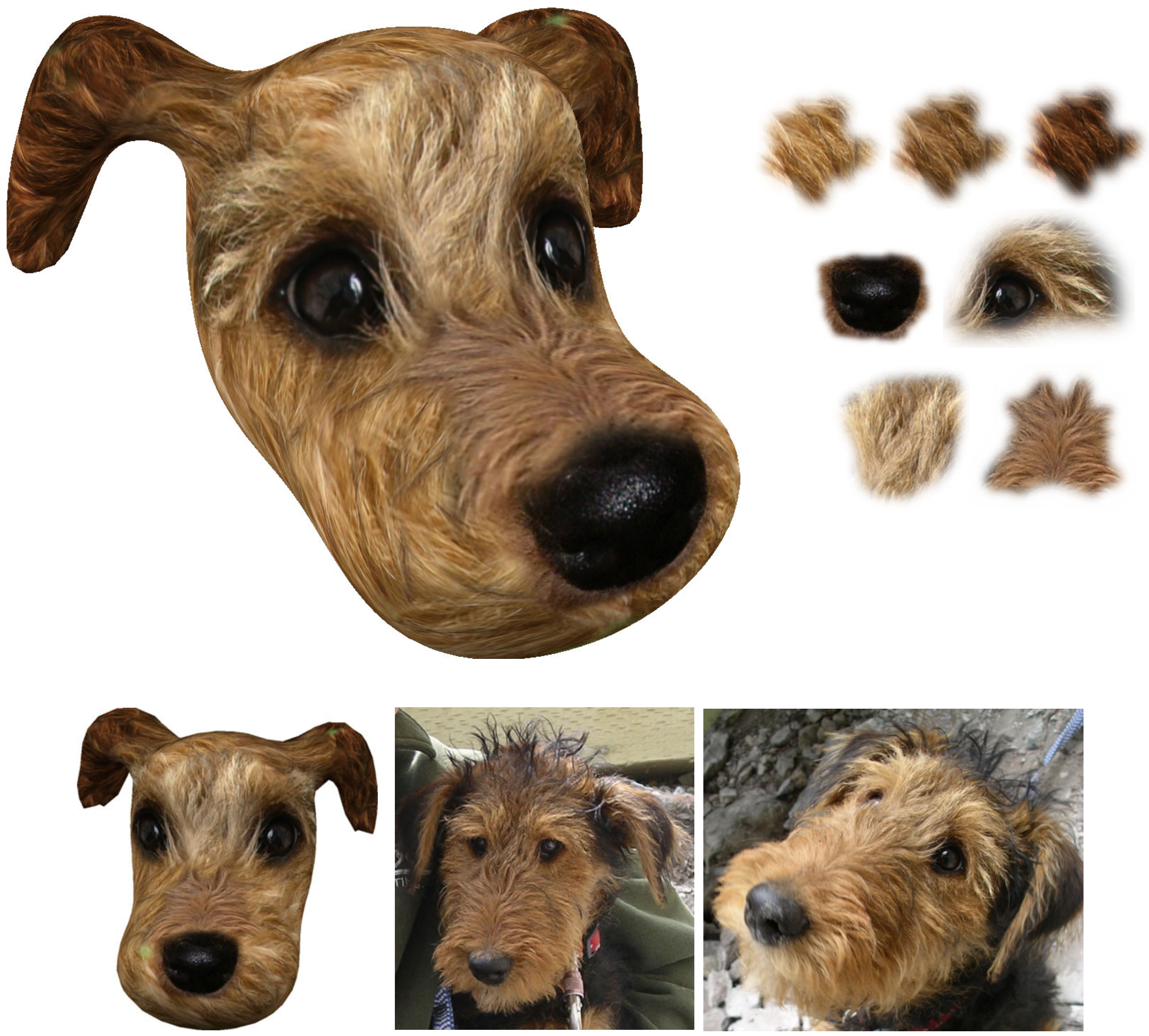


Figure 25: *The base texture and feature textures for this dog were segmented from a few photographs, two are shown in the figure. Generating the lapped base fur texture takes from a few seconds to a few minutes, depending on the number of decals and the sampling rate of the underlying point set. The eyes, nose, and extra fur textures take only a minute to position. The extra fur textures (and the fur on the edge of the eyes) help break up the semi-regularity of the base texture, creating a much more compelling result.*

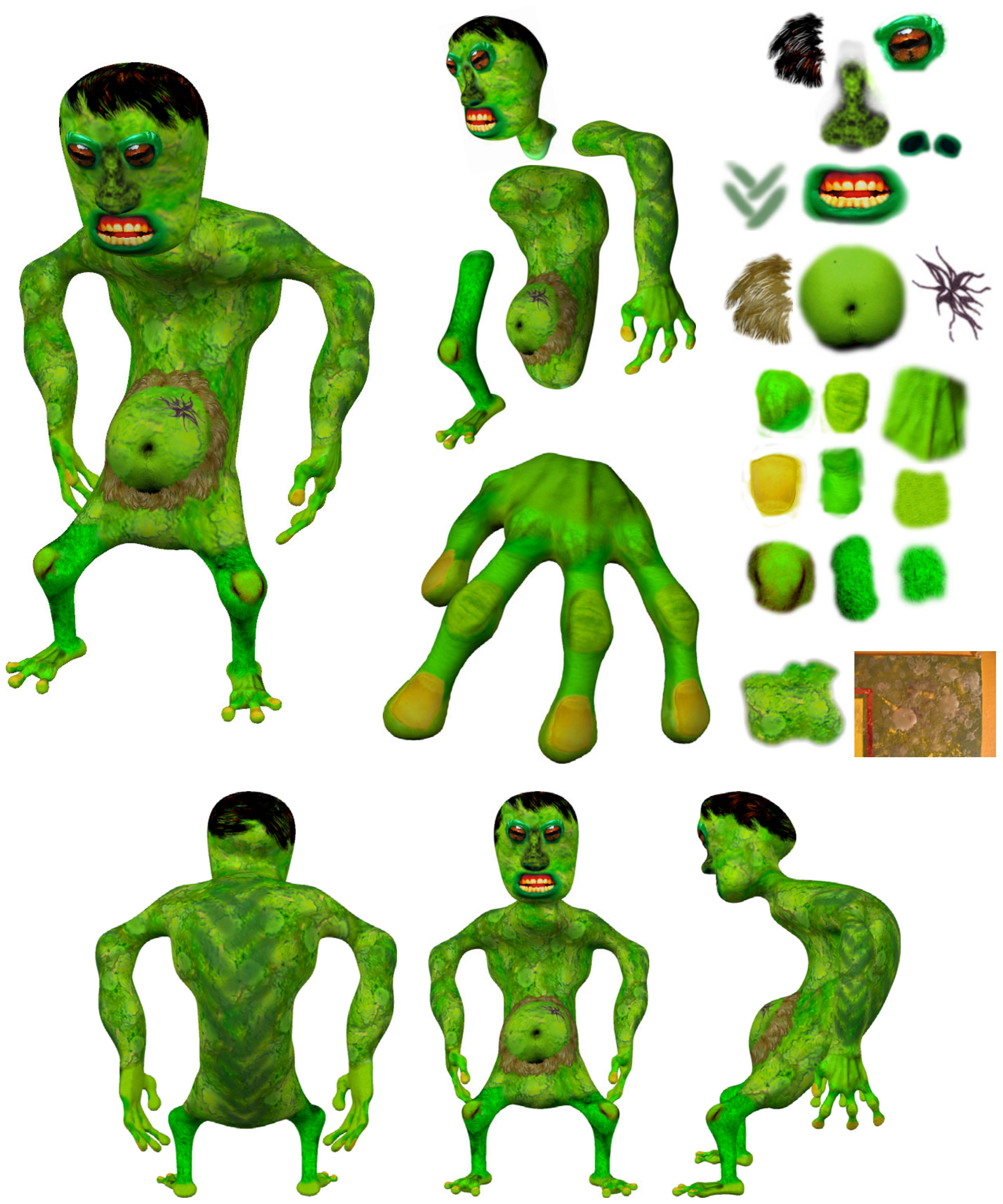


Figure 26: *This implicit gremlin model was textured using the 19 images shown in the top right. The base skin texture is created by lapping an image taken from the photograph shown on the middle right. The model was textured separately in 4 parts - arm, head, body, and leg. The arm and leg were duplicated, and then the parts were assembled. Of the 392 decals used in total, 78 were manually placed, including 24 for the hand. The rest were generated automatically, either with texture lapping or by duplicating the arm and leg.*