EUROGRAPHICS 2007 / D. Cohen-Or and P. Slavík
(Guest Editors)

Volume 26 (2007), Number 3

Texturing Internal Surfaces from a Few Cross Sections

Nico Pietroni[1,2], Miguel A. Otaduy[3], Bernd Bickel[3], Fabio Ganovelli[1] and Markus Gross[3]

[1] Visual Computing Laboratory, ISTI-CNR, Pisa, Italy
[2] Endocas Center for Computer Assisted Surgery, Pisa, Italy
[3] Computer Graphics Laboratory, ETH Zurich, Switzerland

Abstract

We introduce a new appearance-modeling paradigm for synthesizing the internal structure of a 3D model from photographs of a few cross-sections of a real object. When the internal surfaces of the 3D model are revealed as it is cut, carved, or simply clipped, we synthesize their texture from the input photographs. Our texture synthesis algorithm is best classified as a morphing technique, which efficiently outputs the texture attributes of each surface point on demand. For determining source points and their weights in the morphing algorithm, we propose an interpolation domain based on BSP trees that naturally resembles planar splitting of real objects. In the context of the interpolation domain, we define efficient warping and morphing operations that allow for real-time synthesis of textures. Overall, our modeling paradigm, together with its realization through our texture morphing algorithm, allow users to author 3D models that reveal highly realistic internal surfaces in a variety of artistic flavors.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Color, Shading, Shadowing, and Texture

1. Introduction

Textures provide a simple and efficient way of modeling 3D objects by separating appearance properties from the geometric description. Textures have been profusely used in computer graphics for modeling the external structure of objects, either through photographs [SKvW*92] or procedural models [Per85, EMP*94]. When objects undergo topological changes their internal structure is revealed, and this phenomenon has motivated approaches for modeling the texture of *internal surfaces* of objects [HB95, DGF98, CDM*02, JDR04, ONOI04]. However, these approaches are either limited in their scope or do not aim at producing photorealistic results. The simple procedure of taking photographs of objects and "pasting" them on a 3D model has been successfully applied to external but not internal surfaces.

In this paper, we introduce a new modeling paradigm. As shown in Figure 1, the input data to our modeling paradigm consists of the boundary representation of a 3D model, plus *a few photographs of cross-sections* of a real object, which we refer to as *exemplars*. This data is sufficient for defining plausible appearance properties of internal structure at any point inside the 3D model and, as a result, we can generate instances of the 3D model that reveal internal surfaces with highly realistic texture. Our modeling paradigm can be applied for carving other 3D models out of organic objects, interactively texturing cut surfaces in virtual cutting or fracture simulations, transferring internal textures of real objects to arbitrary 3D models without topology restrictions, or producing artistic combinations of internal textures from different objects. We also relax the requirement of full cross-section photographs, allowing the user to input partial samples or even synthetic exemplars.

With our approach, it would be possible to compute a full 3D volumetric texture and then synthesize an internal surface as a cut on the 3D texture. However, the computation of the 3D texture is not a requirement. We have designed an efficient algorithm for on-demand computation on a per-point basis inside the 3D model, which, in our implementation, yields a throughput of 20 000 points/sec. Our on-demand solution allows for a texture resolution that is not limited by storage, at most by the resolution of the input exemplars.

Our texture morphing algorithm rests on two main technical contributions. The first contribution is the definition of a domain and a procedure for interpolating appearance properties from planar oriented exemplars in 3D. We observe that splitting a physical object with planar cuts produces a binary

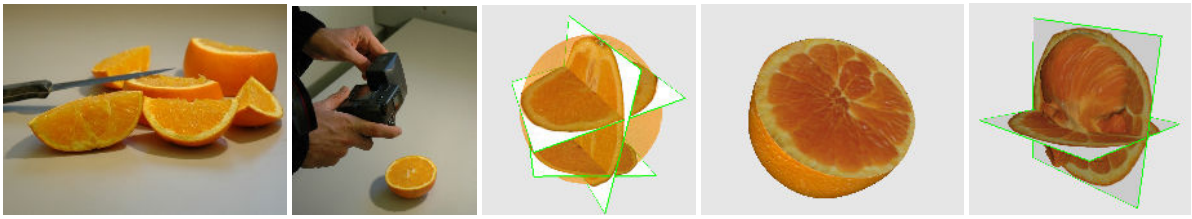

Figure 1: Our Paradigm for Digital Content Creation. *We capture images of internal surfaces of real objects and, in an interactive editor, we place them in the local reference frame of a 3D model. Relying on our two main technical contributions (a natural interpolation domain for object cross-sections and a novel texture morphing algorithm), we can, among other results, produce models that reveal internal surfaces with highly realistic texture, or carve 3D models out of organic objects.*

space partitioning (BSP) [FKN80] of the object, thus we naturally employ a BSP tree for decomposing our interpolation domain. Then, we employ curved projections and scattered data interpolation for defining source points for interpolation and their weights. The second contribution is an efficient single-point multi-exemplar morphing algorithm, inspired on the elegant warping-based morphing algorithm of Matusik et al. [MZD05], with notable differences. We adapt the computation of texture warpings to our BSP-based interpolation domain, and we present an efficient method for computing inverse warpings on a per-point basis through first-order approximation, plus a local orientation-aware histogram matching procedure for feature enhancement.

We implement our modeling paradigm through an interactive and very intuitive editor where the user can place the exemplars inside the 3D model, trigger the preprocessing of the data, and then use the precomputed data for the actual on-demand computation of internal textures. We continue the paper with a discussion of related work, and then we describe our contributions and results.

2. Related Work

The problem of texture synthesis is typically posed as producing a large (non-periodic) texture from a small example. Most of the existing approaches work on 2D textures, trying to match target pixels to those in the sample [EL99, WL00]. Wei [Wei01] applied a similar idea to extend 2D synthesis algorithms to 3D, by using orthogonal 2D views and combining them in the search for best matches. Other statistical approaches have also been applied for 3D texture synthesis, such as the equalization of histograms between 2D views and the 3D volume [HB95], or an improved method that accounts for Fourier transform coefficients [DGF98]. The latter method employs orthogonal cross-sections, but it cannot handle textures with clearly structured features. Recently, Kopf et al. [KFC*07] have incorporated a non-parametric global optimization approach. Statistics-based methods have also been used for texture mixing, which is related to 3D synthesis [BJEYLW01]. 2D texture synthesis techniques have addressed relevant problems for our application, such as multi-exemplar combination [ZZV*03], order-independent and parallelizable synthesis (for on-demand synthesis) [WL02, LH05]), or feature-aware synthesis [WY04, LH06].

Other methods for 3D texture synthesis include procedural techniques for simple patterns [LP00], procedural authoring techniques [CDM*02], stereological techniques [JDR04], or synthesis of cross-sections from one image [ONOI04]. The method by Owada et al. shares conceptual similarities with ours from the point of view of the modeling paradigm, but the user must manually determine correspondences and guide the transformation from 2D to 3D texture, while in our case textures are generated by morphing multiple exemplars.

Morphing has previously been exploited as a synthesis mechanism as well. From the point of view of the methodology, our approach is most closely related to the one of Matusik et al. [MZD05]. As discussed throughout the paper, we extend the warping-based morphing algorithm of Matusik et al. to efficiently synthesize texture on arbitrary 3D points on-demand. Moreover, they exploited morphing for 2D texture mixing rather than 3D synthesis. Their approach is derived from morphable models that combine nonlinear warps and linear analysis [JP98]. Other techniques for texture morphing include the use of prescribed patterns and flow [LLSY02], or the application of morphing to transform procedural textures [BD04].

Although not directly related to our problem, 3D texture synthesis has recently expanded to applications including the embedding of 3D microgeometry on larger-scale surfaces [PBFJ05, POC06], or texturing free-surfaces of fluids [BSM*06, KAK*06].

3. Texture Computation Pipeline

The computation of 3D textures from cross-section images can be decomposed into two procedures. At runtime, texture colors are synthesized on arbitrary 3D positions from predefined model cross-sections through a morphing operation. As a preprocessing step, the user must collect input images, set them up together with 3D geometry in an interactive 3D framework, and compute data necessary for the runtime morphing. In this section, we outline both the runtime and preprocessing tasks. Section 4 describes the interpolation do-

main in which we compute texture through morphing, while section 5 describes the morphing algorithm itself. For the problems of surface parameterization and atlas-packing, we rely on existing area-preserving approaches [CH02].

3.1. Runtime Texture Computation

Our algorithm produces the color of one target point at a time. Given the 3D position $\mathbf{p}$ and orientation $\mathbf{n}$ of the target point, together with a set of input exemplars $\{\mathscr{X}_1,\ldots,\mathscr{X}_n\}$, the synthesis function $\mathbf{c} = f(\mathbf{p},\mathbf{n},\mathscr{X}_1,\ldots,\mathscr{X}_n)$, outputs the color $\mathbf{c}$ of the point.

The texture synthesis is executed on an interpolation domain $\mathscr{D}$, which typically constitutes the interior of a 3D model $\mathscr{M}$. Given a target point $\mathbf{p} \in \mathscr{D}$, we classify its location in a binary space partitioning (BSP) of $\mathscr{D}$, constructed as described in §4.1. Each region of the BSP is defined by a subset of exemplars, which are employed for synthesizing the output color $\mathbf{c}$. We identify a source point $\mathbf{p}_i$ and an interpolation weight w_i for each exemplar, through projection and scattered data interpolation, as described in detail in §4.2.

Once we know the source point and the weight for each contributing exemplar, we apply texture morphing as described in §5.2. Our morphing algorithm is an extension of the work of Matusik et al. [MZD05], with notable differences. In our setting, the source points are different on each exemplar image, and each target point must be synthesized independently, as the contributions of the exemplars vary according to its position. Hence, we have designed an optimized algorithm for synthesizing the color of each target point on demand, allowing for real-time synthesis of thousands of pixels. Nevertheless, we account for the effect of orientation on small-scale features (See §5.3), and we ensure texture continuity across spatially-adjacent target points.

3.2. Preprocessing of Input Exemplars

The preprocessing is composed of four main tasks: (i) generating exemplars and placing them in the interpolation domain $\mathscr{D}$ (See §4.3); (ii) constructing the BSP of the interpolation domain $\mathscr{D}$ (See §4.1); (iii) performing precomputations (e.g., feature detection, histograms, etc.) on each of the exemplars independently (See §5.3 and §5.1); and (iv) defining pairwise warping functions between exemplars that bound common BSP regions (See §5.1). We also allow for some user interaction in the placement of exemplars or for introducing preferred morphing paths (See §4.2).

Thanks to the versatility of our algorithm, exemplars may be photographs of cross sections of a real object, or synthetic images. Similarly, photographs of a certain object can be used for synthesizing textures on yet a different object, enabling efficient 3D texture transfer.

4. Texture Interpolation Domain

In this section, we describe the construction of the BSP of the interpolation domain $\mathscr{D}$ using exemplars, and how interpolation is performed inside each region of the BSP. Then, we propose approaches for defining input exemplars and placing them in the interpolation domain.

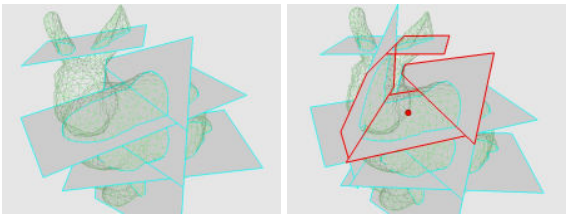

Figure 2: BSP-Tree Produced by Exemplars. *Left: The intersection of the top exemplar plane with the ears of the bunny yields two connected components. Right: This situation is fixed by adding another exemplar. The exemplars bounding the target point (in red) are highlighted.*

4.1. Binary Space Partitioning

As mentioned earlier, we compute textures on a domain $\mathscr{D} \subset \mathbb{R}^3$ corresponding to the interior of a model $\mathscr{M}$. Using a sorted sequence of input exemplars $\{\mathscr{X}_1,\ldots,\mathscr{X}_n\}$, we construct a BSP of $\mathscr{D}$ in the following way. We assume that each exemplar $\mathscr{X}_i$ constitutes a planar region in $\mathscr{D}$. Then, given a subsequence of exemplars $\{\mathscr{X}_1,\ldots,\mathscr{X}_i\}$, defining BSP regions $\{\mathscr{D}_1,\ldots,\mathscr{D}_m\}$, the addition of the next exemplar $\mathscr{X}_{i+1}$ subdivides one of the regions, $\mathscr{D}_j$, into two new ones. Figure 2 shows the BSP of a bunny model produced by a set of exemplars. It is important to highlight that a BSP is actually a natural choice as data structure in our application. Cutting a solid object by successive bisection of one of the existing pieces with a planar cut produces indeed a BSP of the original object.

Each region $\mathscr{D}_j$ of the BSP of $\mathscr{D}$ is bounded by curved surfaces $\{\mathscr{S}_i\} \in \partial\mathscr{D}$, and planar exemplar subdomains $\{\mathscr{X}_{i,j} = \mathscr{X}_i \cap \mathscr{D}_j\}$. The exemplar subdomains $\{\mathscr{X}_{i,j}\}$ must be topologically equivalent to a disk. If this condition does not hold, as shown in Figure 2, the user must introduce additional exemplars. In §5.1, we discuss the computation of pairwise warpings between exemplar subdomains.

4.2. Source Image Points and Interpolation Weights

In order to compute the color at a point $\mathbf{p}$ in the BSP region $\mathscr{D}_j$, the texture morphing algorithm takes as input source points on the exemplars that bound $\mathscr{D}_j$, along with associated interpolation weights. Here, we describe an algorithm based on (possibly curved) projections for finding the source points and computing their weights.

Conceptually, we define the source point $\mathbf{p}_i$ of an exemplar subdomain $\mathscr{X}_{i,j}$ by projection of $\mathbf{p}$ onto $\mathscr{X}_{i,j}$ along a path line $\gamma(\mathbf{p}_i)$ emanating from $\mathscr{X}_{i,j}$ and flowing through $\mathscr{D}_j$. If no assumption is made on texture isotropy, there is no ideal projection scheme a priori. Any method producing smooth, evenly distributed path lines emanating from $\{\mathscr{X}_{i,j}\}$ and covering the entire subdomain $\mathscr{D}_j$ would be valid. In our

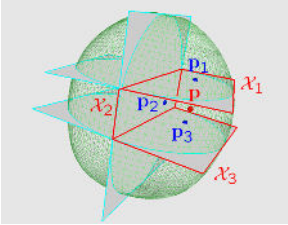

Figure 3: Interpolation Inside BSP Regions. *Given a target point* $\mathbf{p}$*, we define source points* $\mathbf{p}_i$ *on the exemplars* $\mathscr{X}_i$ *that bound the BSP region where* $\mathbf{p}$ *is located.*

implementation, we have adopted the following approach. We compute the barycenter of the exemplar $\mathscr{X}_{i,j}$, we define a curve γ emanating from the barycenter, and we sweep the exemplar plane along γ, until it contains the target point $\mathbf{p}$. In this configuration, we compute the ray from the swept barycenter to $\mathbf{p}$, and we map this ray onto the original exemplar $\mathscr{X}_{i,j}$ by sweeping back along γ. We define the location of the source point $\mathbf{p}_i$ along the mapped ray by preserving the ratio of distances w.r.t. the barycenter and the boundary of the subdomain in the original and swept configurations. We let the user choose the curve γ based on a priori knowledge about the textures. In our current implementation, γ may be a straight line normal to the exemplar, or a great arc on the sphere (which works well for the oranges in Figure 1).

Given the set of source points $\{\mathbf{p}_i\}$, we define interpolation weights for morphing based on scattered data interpolation. Specifically, we compute the (pre-normalized) weight w_i of each source point $\mathbf{p}_i$ based on Shepard interpolation: $w_i = \frac{1}{\|\mathbf{p}-\mathbf{p}_i\|}$. In BSP regions that do not lie on the boundary of $\mathscr{D}$, the source points $\{\mathbf{p}_i\}$ define a convex polyhedron that bounds $\mathbf{p}$, and then it is also possible to use convex weights given by e.g., mean-value coordinates [Flo03].

Our interpolation procedure is guaranteed to be continuous inside regions $\mathscr{D}_j$ of the BSP, as well as across regions. The projection operation and the interpolation weights are all continuous inside a given region. When a BSP boundary is crossed, the interpolation is dominated by a single source point, thus ensuring continuity as well. Since the morphing algorithm described in §5 is also continuous, the complete texture synthesis procedure is C^0 continuous. Of course, the features of the input exemplars may not be continuous, therefore we do not enforce continuity on the output textures.

4.3. Defining Input Exemplars

The generation of input data for the algorithm encompasses (i) the definition of texture attributes for the exemplars, (ii) the definition of a polygonal representation of the model $\mathscr{M}$, and (iii) the placement of exemplar planes in the domain $\mathscr{D}$. We typically collect exemplars by taking photographs of cross sections of real objects, but it is also possible to let an artist define exemplars and their attributes. For example, Figure 6 shows a case where exemplars were generated using 2D texture synthesis techniques [LH05].

When trying to simulate the textures of a real object, the model $\mathscr{M}$ should approximate the boundary of the real-world object that is cut for generating the exemplars. High quality results could be generated by scanning the real-world object, but a coarse approximation (such as the orange model used in Figure 1) proved to be enough in our examples. Once the representation of $\mathscr{M}$ is available, we provide the user with an interactive tool for placing exemplar planes in $\mathscr{D}$, as shown in the accompanying video. The user may typically judge how many exemplars to add based on visual examination of the textures, but we have also incorporated a warping quality metric [MZD05] to aid with this judgement.

Since the model $\mathscr{M}$ may not exactly correspond to the real cut object and it is very hard to accurately place the exemplars, we allow the user to place the exemplars approximately, and then we warp them so that the texture images exactly fit the boundary of $\mathscr{M}$. In our implementation, we constrain the image boundaries to the boundary of $\mathscr{M}$ and we perform a relaxation process in the interior.

5. Texture Morphing

In this section, we describe our algorithm for computing the color of a 3D point as a morphing operation. First, we summarize the texture morphing algorithm of Matusik et al. [MZD05], which comprises a preprocessing part for computing image warpings, and the actual runtime morphing algorithm. Our algorithm presents some differences in the preprocessing of warpings, such as finding multilevel feature correspondences. But, most importantly, our runtime morphing algorithm is designed for efficiently computing the texture of individual 3D points on-demand. This aspect is essential when computing texture on the internal surface of an object, as no pair of surface points shares interpolation weights for morphing. In the second part of this section, we introduce the definition of warping in our interpolation domain, and we describe an approximation to the inverse warping that allows for fast morphing at a single point at a time. Moreover, we present an approach for feature enhancement based on local high-frequency histogram computation.

5.1. Morphing Images

Given a set of input images $\{\mathscr{X}_i\}$, Matusik et al. defined morphing as a convex combination of warped versions of the images. Hence, their algorithm relies heavily on the computation of a warping for each pair of images $(\mathscr{X}_i, \mathscr{X}_j)$. They defined a bijective mapping $\mathbf{f}_{ij} : \mathscr{X}_i \subset \mathbb{R}^2 \rightarrow \mathscr{X}_j \subset \mathbb{R}^2$, such that a point $\mathbf{p}_i \in \mathscr{X}_i$ maps to a point $\mathbf{f}_{ij}(\mathbf{p}_i) \in \mathscr{X}_j$, where $\mathbf{f}_{ij}(\mathbf{p}_i) = \mathbf{p}_i + W_{ij}(\mathbf{p}_i)$. The 2D vector W_{ij} is referred to as the *warping*, and can be regarded as a translation.

5.1.1. Computation of Warpings

In essence, we follow the same steps as Matusik et al. for computing each mapping $\mathbf{f}_{ij}$: (i) apply an edge detec-

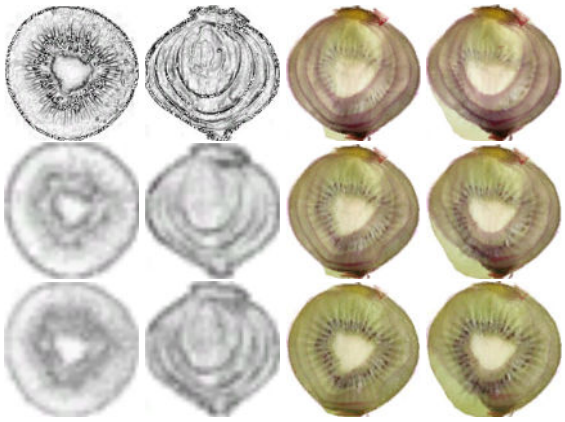

Figure 4: Morphing Using Gaussian Stacks. *The two leftmost columns show feature maps for a kiwi and an onion. On the top, feature map at full resolution; on the center, feature map at a level of our Gaussian stack; and on the bottom, feature map at the same level without Gaussian stack. Notice the blurry region in the center of the kiwi's feature map. The two rightmost columns show several steps in a morphing sequence. On the left, with our Gaussian stack; on the right, without Gaussian stack, which noticeably increases blur.*

tor [RT01] to $\mathscr{X}_i$ and $\mathscr{X}_j$ to compute feature images; (ii) compute a stack of feature images by downsampling; and (iii) perform automatic multilevel feature matching by minimizing the Euclidean norm of feature image differences, with a regularization term that measures image deformation.

In our setting, for every pair of exemplars that share a boundary, we first align and scale them such that their axis-aligned bounding boxes match. As opposed to Matusik et al., we apply multilevel feature detection, by computing Gaussian stacks [LH05] of the input exemplars, and applying the edge detector and further downsampling to each image of the Gaussian stack independently. Figure 4 shows the improvement obtained with Gaussian stacks.

5.1.2. Computation of Morphed Images

Based on all pairwise mappings, Matusik et al. computed the morphed image from convex color interpolation weights α_i and convex warping interpolation weights β_i. In order to compute the color $\mathbf{c}$ at pixel $\mathbf{p}$, their algorithm performs a convex combination of the images evaluated at pixels $\mathbf{q}_i$, $\mathbf{c}(\mathbf{p}) = \sum_i \alpha_i \mathbf{c}_i(\mathbf{q}_i)$, where each $\mathbf{q}_i - \mathbf{p}$ represents the inverse warping of $\mathbf{p}$ based on the convex combination of warpings. In other words, $\mathbf{q}_i$ is the pixel in image $\mathscr{X}_i$ that maps to $\mathbf{p}$: $\mathbf{q}_i + \sum_j \beta_j W_{ij}(\mathbf{q}_i) = \mathbf{p}$. The complete morphing function can then be expressed using the concept of inverse warping as:

$$\mathbf{c}(\mathbf{p}) = \sum_i \alpha_i \mathbf{c}_i(\mathbf{p} + \left(\sum_{j \neq i} \beta_j W_{ij}\right)^{-1}(\mathbf{p})). \qquad (1)$$

The evaluation of the inverse warping requires that each warping W_{ij} must be scaled by its associated weight, and the weighted sum is then computed over the complete image, searching for the pixel that maps to $\mathbf{p}$. This search can be implemented, e.g., by rasterization of the warped mesh with original locations as attributes. When the morphing algorithm is applied to a complete texture image, the cost of computing the inverse of the scaled warping is amortized over all target pixels. However, this approach is far from optimal when the interpolation weights vary for each target point, as is our case. In §5.2, we describe our optimized solution for single-point computation.

Matusik et al. complete the morphing process by applying a histogram matching for feature enhancement [HB95]. Histogram computation also presents a cost dependent on the size of the exemplars, which is amortized when computing a full texture, but not in our case. In §5.3, we again present an optimized solution for single-point computation.

5.2. Single-Point Multi-Exemplar Morphing

We now present the definition of warping in our BSP-based interpolation domain, together with an efficient approximation of the inverse warping that can be explicitly evaluated.

5.2.1. Warping in the Interpolation Domain

In the interpolation domain $\mathscr{D}$, morphing takes place among different source points $\mathbf{p}_i$ for each exemplar. For two exemplars $(\mathscr{X}_i, \mathscr{X}_j)$, the warping vector W_{ij} cannot be defined as the difference vector between a point $\mathbf{q} \in \mathscr{X}_i$ and its corresponding point $\mathbf{f}_{ij}(\mathbf{q}) \in \mathscr{X}_j$. Instead, we account for the translation between the reference systems of the two exemplars, and we define the warping as:

$$W_{ij}(\mathbf{q}) = (\mathbf{f}_{ij}(\mathbf{q}) - \mathbf{p}_j) + (\mathbf{p}_i - \mathbf{q}). \qquad (2)$$

As noted in §5.1, it is highly inefficient to compute the exact inverse warping in the context of single-point morphing, but we have devised an efficient approximation, presented next.

5.2.2. Efficient Warping Approximation

For texture morphing in the interpolation domain $\mathscr{D}$, we slightly modify (1). The source points $\{\mathbf{p}_i\}$ vary across exemplars, and we use the same weights $\{w_i\}$ (defined in §4.2) for color and warping interpolation. Then, we obtain the following morphing equation:

$$\mathbf{c}(\mathbf{p}) = \sum_i w_i \mathbf{c}_i(\mathbf{p}_i + \left(\sum_{j \neq i} w_j W_{ij}\right)^{-1}(\mathbf{p}_i)), \qquad (3)$$

where $\mathbf{q}_i - \mathbf{p}_i = \left(\sum_{j \neq i} w_j W_{ij}\right)^{-1}(\mathbf{p}_i)$ is the inverse warping of $\mathbf{p}_i$.

In order to find the inverse warping on an exemplar $\mathscr{X}_i$, we approximate the warping from each other exemplar $\mathscr{X}_j, j \neq i$ based on the value at the corresponding point of its source point, $\mathbf{f}_{ji}(\mathbf{p}_j)$. In other words, $W_{ij}(\mathbf{q}) \approx W_{ij}(\mathbf{f}_{ji}(\mathbf{p}_j)) = \mathbf{p}_i - \mathbf{f}_{ji}(\mathbf{p}_j)$. From this approximation, we reach the estimate for

Figure 5: Feature Enhancement with Local Histograms. *Morphing from an onion to a cabbage. The rightmost column compares a portion of the morphed texture, with feature enhancement through our local histogram computation (top), and without feature enhancement (bottom).*

the inverse warping in (3), which yields

$$\mathbf{q}_i \approx w_i \mathbf{p}_i + \sum_{j \neq i} w_j \mathbf{f}_{ji}(\mathbf{p}_j). \tag{4}$$

The approximation results in the evaluation of the convex combination of the source texel $\mathbf{p}_i \in \mathscr{X}_i$ and the corresponding points of all other source points $\mathbf{p}_j \in \mathscr{X}_j, j \neq i$. Remarkably, this approximation produces the accurate result if one of the weights $w_k = 1$, and this contributes to the continuity of the morphing as the source point crosses boundaries of BSP regions. Our approximation obviously does not yield the same morphs as using the exact warping, as the warping is a highly nonlinear function, but our approach produces plausible, sharp results.

5.3. Feature Enhancement

The morphing algorithm we just described inevitably produces a certain blending of the source exemplars. It exploits the warping for morphing large and medium scale features effectively, but small scale features are blended. As mentioned before, Matusik et al. [MZD05] proposed a histogram matching technique for reinserting small scale features into the final texture. The basic idea is to compute the histogram of high-frequency spectra in the target texture, and replace colors based on the probability distribution functions of the source exemplars. Matusik et al. employed steerable pyramids [HB95] for matching histograms at high-frequencies and then recovering the full texture colors.

Although effective, this feature enhancement approach requires the computation of the histogram on the full final texture image, which is inefficient for our per-point on-demand computation. However, one can observe that the histogram in the high-frequency spectrum can be well approximated by windowing the computation. In other words, it suffices to compute the histogram in a local kernel around the target point. We exploit this observation by precomputing local histograms (with a 7×7 kernel) for every pixel of the input exemplars, and similarly computing at runtime only a local histogram around the target point. In fact, since textures are computed on a surface, we anyway must compute the texture on a local kernel around each target point. Reusing the texture values from neighboring points also produces an orientation-aware histogram matching, as the small-scale features depend on the local orientation of the surface. Figure 5 shows the successful feature enhancement achieved with our local histogram computation.

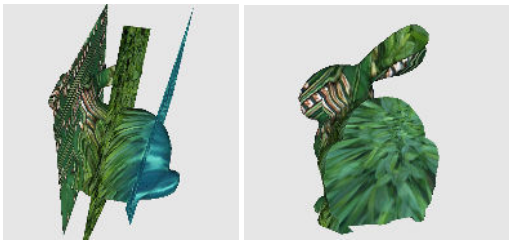

Figure 6: Bunny with Patterned Textures. *Left: exemplars (a circuit board, leaves, and water reflections) and bunny's surface; Right: cross-section of the textured bunny.*

6. Results

We have applied our appearance modeling framework in a variety of examples that show the diversity of problems where it can be used, as well as its versatility in terms of input data. Our examples have been generated on a laptop, with 1.7 GHz Intel Centrino processor and 1 GB of RAM. With such commodity hardware, our on-demand texturing algorithm is capable of producing a throughput of approximately 20000 pixels/sec. Histogram matching, with a 7×7 kernel, takes 50% of the computations.

One of the applications where our modeling paradigm shows great benefit is the simulation of cutting and fracture. Figure 8 shows two examples of interactive cutting simulation. Note that the simulations were created interactively, although they were later ray-traced offline. During interactive cutting or fracture, the on-demand computation of texture on internal surfaces plays a crucial role on the richness and realism of the results. The top row of Figure 8 depicts the modeling of internal surfaces of the apple with the texture of an orange. The slices appear crisp and clear, and one can easily distinguish the border and the different features of the orange, even though we only performed three cross-sections on the real orange. Notice that the cuts in the simulation are not planar, yet our technique successfully captures large-scale texture features with changing orientation. The bottom row of Figure 8 depicts a similar animation, where texture was computed from cross-sections of a cabbage and an onion (See Figure 5). The morphing between onion texture (bottom of the apple) and cabbage texture (top of the apple) is clearly visible, while features are sharply captured.

Our versatile texturing approach allows the combination of highly diverse textures, as shown in Figures 6 and 7. In these examples, we use a simple sphere as the containing object $\mathscr{M}$. Notice also that the colors of exemplars do not match at their intersections, but our morphing was able to handle this situation without artifacts.

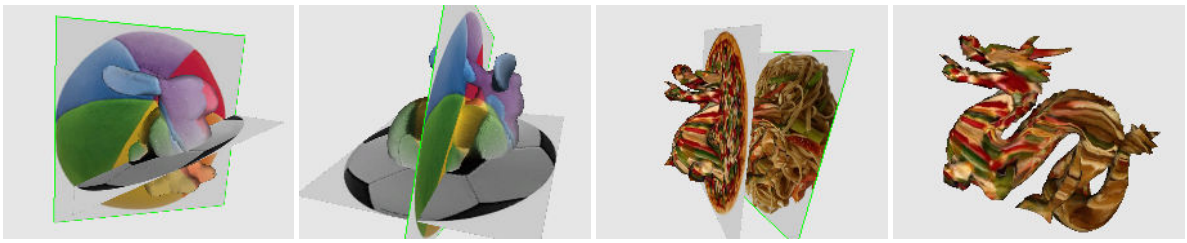

Figure 7: Versatile Texturing. *Highly diverse textures are morphed onto a bunny and a dragon.*

Figure 8: New Generation of Transgenic Fruits. *Virtual slicing of an apple, showing internal textures generated with our algorithm. Top: orange texture morphed from three cross-sections. Bottom: morph between onion and cabbage cross-sections.*

7. Conclusions and Future Work

The texturing technique presented in this paper provides a very simple yet powerful paradigm for creating appearance models for 3D objects. We have shown its application for carving objects or texturing internal surfaces in cutting simulation. The simplicity of the method, where a user takes cross-section photographs of real objects and places them in an interactive 3D editor, makes it highly amenable.

As shown in the examples, our method produces highly realistic textures for internal surfaces of models that resemble real objects, but it also produces plausible textures for non-realistic examples. Our algorithm captures successfully the morphing of global and medium-scale features through multi-level warping, and reintroduces small features efficiently through local histogram matching. Moreover, an efficient approximation of warping enables the implementation of the algorithm as an on-demand routine for texturing internal surfaces during cutting simulation.

In many of the examples we have produced (e.g., morphing between onion and cabbage, the bunnies, or the dragon), it is practically impossible to find a warping between the exemplars that completely avoids blur. In essence, the feature images are not pure deformations of each other, hence a warping is not sufficient for capturing the transition. Although our method succeeds at producing plausible results with little blur, the automatic feature matching may lead to warpings that produce high feature distortion when morphing between images. A purely morphing-based technique may not be the best solution for such examples, and it would be interesting to study combinations of morphing-based texturing with techniques from stereology [JDR04] or pattern-matching optimization [KFC*07].

Our current implementation is limited to synthesis of color, but it would be interesting to investigate other appearance attributes. It is not obvious, however, that our morphing-based approach will be applicable to e.g., bump mapping, as its major strength is capturing global features.

We also consider the possibility of designing a parallel implementation on graphics hardware, as this could accelerate the morphing stage of the algorithm. Moreover, there could be cases where the texture does not need to be stored, as the user simply looks at an internal surface once, while sweeping through the object. The BSP-tree poses probably the biggest difficulties for a parallel implementation, and one option could be to limit the cross-sections to be axis-aligned, and implement the BSP-tree as a K-d tree.

Acknowledgements

We would like to thank the anonymous reviewers for their comments, members of the Computer Graphics Lab in Zurich, in particular Filid Sadlo, Simon Heinzle and Jens Puwein, for their help, and Martin von Siebenthal and XXX for texture data. This research was supported in part by the NCCR Co-Me of the Swiss National Science Foundation.

References

[BD04] BOURQUE E., DUDEK G.: Procedural texture matching and transformation. *Proc. of Eurographics* (2004).

[BJEYLW01] BAR-JOSEPH Z., EL-YANIV R., LISCHINSKI D., WERMAN M.: Texture mixing and texture movie synthesis using statistical learning. *IEEE Transactions on Visualization and Computer Graphics 7*, 2 (2001), 120–135.

[BSM*06] BARGTEIL A. W., SIN F., MICHAELS J. E., GOKTEKIN T. G., O'BRIEN J. F.: A texture synthesis method for liquid animations. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2006).

[CDM*02] CUTLER B., DORSEY J., MCMILLAN L., MÜLLER M., JAGNOW R.: A procedural approach to authoring solid models. *Proc. of ACM SIGGRAPH* (2002), 302–311.

[CH02] CARR N. A., HART J. C.: Meshed atlases for real-time procedural solid texturing. *ACM Trans. on Graphics 21*, 2 (2002).

[DGF98] DISCHLER J. M., GHAZANFARPOUR D., FREYDIER R.: Anisotropic solid texture synthesis using orthogonal 2d views. *Proc. of Eurographics* (1998), 87–96.

[EL99] EFROS A. A., LEUNG T. K.: Texture synthesis by non-parametric sampling. *Proc. of IEEE ICCV* (1999).

[EMP*94] EBERT D., MUSGRAVE K., PEACHEY D., PERLIN K., WORLEY S.: *Texturing and Modeling: A Procedural Approach*. Academic Press, 1994.

[FKN80] FUCHS H., KEDEM Z. M., NAYLOR B. F.: On visible surface generation by a priori tree structures. *Proc. of ACM SIGGRAPH* (1980), 124–133.

[Flo03] FLOATER M. S.: Mean value coordinates. *Computer Aided Geometric Design 20*, 1 (2003), 19–27.

[HB95] HEEGER D. J., BERGEN J. R.: Pyramid-based texture analysis/synthesis. *Proc. of ACM SIGGRAPH* (1995), 229–238.

[HS90] HIGHAM N. J., SCHREIBER S. S.: Fast polar decomposition of an arbitrary matrix. *SIAM Journal on Scientific and Statistical Computing 11*, 4 (1990), 648–655.

[JDR04] JAGNOW R., DORSEY J., RUSHMEIER H.: Stereological techniques for solid textures. *Proc. of ACM SIGGRAPH* (2004), 329–335.

[JP98] JONES M. J., POGGIO T.: Multidimensional morphable models. *Proc. of ICCV* (1998), 683–688.

[KAK*06] KWATRA V., ADALSTEINSSON D., KWATRA N., CARLSON M., LIN M. C.: Texturing fluids. *ACM SIGGRAPH Sketch* (2006).

[KFC*07] KOPF J., FU C.-W., COHEN-OR D., DEUSSEN O., LISCHINSKI D., WONG T.-T.: Solid texture synthesis from 2d exemplars. *Proc. of ACM SIGGRAPH* (2007).

[LH05] LEFEBVRE S., HOPPE H.: Parallel controllable texture synthesis. *Proc. of ACM SIGGRAPH* (2005), 777–786.

[LH06] LEFEBVRE S., HOPPE H.: Appearance-space texture synthesis. *Proc. of ACM SIGGRAPH* (2006), 541–548.

[LLSY02] LIU Z., LIU C., SHUM H.-Y., YU Y.: Pattern-based texture metamorphosis. *Proc. of Pacific Graphics Conference* (2002).

[LP00] LEFEBVRE L., POULIN P.: Analysis and synthesis of structural textures. *Proc. of Graphics Interface* (2000).

[MZD05] MATUSIK W., ZWICKER M., DURAND F.: Texture design using a simpicial complex of morphable textures. *Proc. of ACM SIGGRAPH* (2005), 787–794.

[ONOI04] OWADA S., NIELSEN F., OKABE M., IGARASHI T.: Volumetric illustration: Designing 3d models with internal textures. *Proc. of ACM SIGGRAPH* (2004), 322–328.

[PBFJ05] PORUMBESCU S., BUDGE B., FENG L., JOY K.: Shell maps. *Proc. of ACM SIGGRAPH* (2005), 626–633.

[Per85] PERLIN K.: An image synthesizer. *Proc. of ACM SIGGRAPH* (1985), 287–296.

[POC06] POLICARPO F., OLIVEIRA M., COMBA J.: Real-time relief mapping on arbitrary polygonal surfaces. *Proc. of Symposium on Interactive 3D Graphics and Games* (2006), 55–62.

[RT01] RUZON M., TOMASI C.: Edge, junction, and corner detection using color distributions. *IEEE Trans. on Pattern Analysis and Machine Intelligence 23*, 11 (2001), 1281–1295.

[SKvW*92] SEGAL M., KOROBKIN C., VAN WIDENFELT R., FORAN J., HAEBERLI P.: Fast shadows and lighting effects using texture mapping. *Proc. of ACM SIGGRAPH* (1992).

[Wei01] WEI L.-Y.: *Texture Synthesis by Fixed Neighborhood Searching*. PhD thesis, Stanford University, 2001.

[WL00] WEI L.-Y., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. *Proc. of ACM SIGGRAPH* (2000).

[WL02] WEI L.-Y., LEVOY M.: *Order-Indpendent Texture Synthesis*. Tech. Rep. TR-2002-01, Stanford University, 2002.

[WY04] WU Q., YU Y.: Feature matching and deformation for texture synthesis. *Proc. of ACM SIGGRAPH* (2004), 362–365.

[ZZV*03] ZHANG J., ZHOU K., VELHO L., GUO B., SHUM H.-Y.: Synthesis of progressively varying textures on arbitrary surfaces. *Proc. of ACM SIGGRAPH* (2003), 295–302.