Real-Time Rendering of Textures with Feature Curves

EVGUENI PARILOV and DENIS ZORIN
New York University

The standard bilinear interpolation on normal maps results in visual artifacts along sharp features, which are common for surfaces with creases, wrinkles, and dents. In many cases, spatially varying features, like the normals near discontinuity curves, are best represented as functions of the distance to the curve and the position along the curve. For high-quality interactive rendering at arbitrary magnifications, one needs to interpolate the distance field preserving discontinuity curves exactly.

We present a real-time, GPU-based method for distance function and distance gradient interpolation which preserves discontinuity feature curves. The feature curves are represented by a set of quadratic Bezier curves, with minimal restrictions on their intersections. We demonstrate how this technique can be used for real-time rendering of complex feature patterns and blending normal maps with procedurally defined profiles near normal discontinuities.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation—*Antialiasing, line and curve generation*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Color, shading, shadowing, and texture*

General Terms: Algorithms, Human Factors, Performance

Additional Key Words and Phrases: Curvilinear feature rendering, distance function, GPU algorithms, normal mapping, resolution independence

ACM Reference Format:

Parilov, E. and Zorin, D. 2008. Real-time rendering of textures with feature curves. ACM Trans. Graph. 27, 1, Article 3 (March 2008), 15 pages. DOI = 10.1145/ 1330511.1330514 http://doi.acm.org/10.1145/1330511.1330514

1. INTRODUCTION

Texture mapping is a powerful tool for adding high-resolution detail to the material properties and geometry of models. However, the regular sampling pattern used in textures leads to a range of aliasing problems, particularly for curved one-dimensional features. Such features are common in textures and include fine-scale geometry, such as gratings, creases, scratches, cracks and seams, shadow maps, and vector glyphs. For example, two main types of artifacts may be observed for normal maps: staircasing artifacts due to misalignment of linear features with the texture sample grid, and lighting artifacts due to use of bilinear interpolation in areas of discontinuity (Figure 1).

Recent feature-based texture representations [Ramanarayanan et al. 2004; Tumblin and Choudhury 2004; Sen 2004; Tarini and Cignoni 2005] improve texture appearance by explicitly representing discontinuities. This approach eliminates or reduces many types of artifacts, and focuses on a unique way of representing and real-time rendering of curvilinear features.

In this paper we describe algorithms for textures with *feature curves*, extending previously proposed feature-based texturing techniques. The distinctive elements of our approach to representing features are: a) features not only represent the object boundaries but also can be arbitrary functions of the distance and direction to curvilinear discontinuity, and b) curved features are represented by quadratic Bezier segments, rather then approximated with line segments, which is essential for resolution independence.

Another central element of our construction is an approximation of the *unsigned* distance function to the feature curve set and its gradient: the distance function vanishes *precisely* along the feature curves, is smooth away from feature curves, and is efficiently represented by a pair of auxiliary textures. Using unsigned distance functions is crucial for representing open feature curves. We present a GPU-based algorithm for reconstructing the distance map and its gradient from these textures in real time.

To avoid input restrictions, we developed a robust, incremental preprocessing algorithm, which converts complex configurations with multiple connected features into a texture representation that can be handled by our rendering algorithm simplifying complex joints as necessary. As a result, our system imposes few restrictions on the input set of feature curves, although certain configurations may cause artifacts, as discussed in Section 6. The preprocessing algorithm is interactive if linear rather than quadratic curves are used.

Our approach combines elements of texture-based and procedural geometry representation: the user can specify a profile for the immediate neighborhood of a sharp feature as a function of the distance to the feature. Through using user-controlled profiles, we can produce a variety of behaviors without dramatic increase in texture size.

Author's address: E. Parilov; email: parilov@cs.nyu.edu.

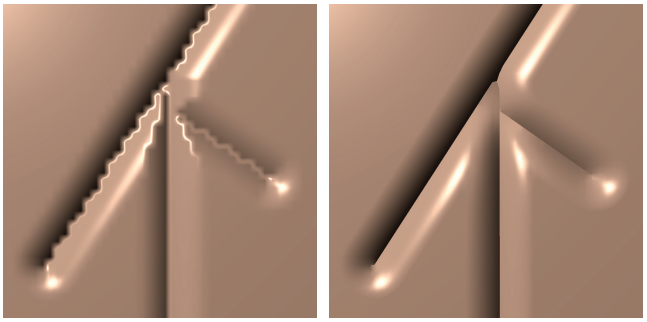

Fig. 1. Left: Typical bump and normal map artifacts. Note the bright lines at the bottom of the crease: these lines are due to the interpolation between normals pointing in opposite directions. Right: the enhanced map using our technique. The resolution of the map in both cases is the same; the visible part is 40×40 pixels.

In this paper, we focus on normal maps, for which these extensions are most relevant. However, our technique can be useful for any piecewise-smooth texture, for which it is desirable to introduce different types of feature profiles as a function of the distance to the curve (e.g., color variation as the distance to the veins on tree leaves or crease profile variation in displacement maps).

2. PREVIOUS WORK

Images and Textures with Resolution-Independent Features. The idea of explicit representation of discontinuities for data sampled on regular or unstructured grids has appeared in computer graphics in many different contexts (e.g., discontinuity meshing for radiosity [Heckbert 1992] to nonphotorealistic rendering). We focus only on the most closely related work. Salisbury et al. [1996] presented an illustration reproduction approach, which keeps discontinuous regions sharp at any scale. The reconstruction algorithm starts from an arbitrary interpolation kernel and reweights its intensity values according to the closeness to the sample. As the algorithm is based on finding shortest paths, it is difficult to adapt it for interactive applications.

Our method builds on ideas in feature-based texture work [Ramanarayanan et al. 2004], bixels [Tumblin and Choudhury 2004] and silhouette maps [Sen 2004] and adaptive distance field work [Frisken et al. 2000, 2002].

Bala et al. [2003] and Ramanarayanan et al. [2004] have developed a general framework for representing sharp features in textures, called feature-based textures. Feature boundaries are represented by Bezier segments, and texels may store complex intersections of boundaries. Only values in the same continuous region are used for texture interpolation. Our method can be regarded as an extension of this approach with distance functions and an adaptation of it to hardware implementation by conversion of the input curve networks to simplified form.

Tumblin and Choudhury [2004] encode discontinuities using *bixels*, that is, pixels with additional annotation, which define texture discontinuity segments for every pixel, allowing only a restricted set of combinations of segments. Our feature maps are similar in spirit, but they encode the distance field and are adapted to interactive rendering.

Sen's silhouette maps [Sen 2004], extending Sen et al. [2003], are similar to feature-based textures and bixels, but the interpolation approach used in this work is further simplified such that it maps well to graphics hardware. As in Tumblin and Choudhury [2004], a finite number of discontinuity boundary configurations are used for each texel. The discontinuity representation is restricted only to straight segments.

Adaptively sampled distance fields (ADF) [Frisken et al. 2000] is the most closely related technique, converting 2D (or 3D) object boundaries into a *signed* distance field and storing distance samples in a quadtree or octree of nonuniform cells. In its original form, the technique is restricted to smooth boundary curves of two-dimensional domains. The boundaries of objects are accurately represented by using a polynomial distance field approximation inside cells containing boundaries [Frisken et al. 2002]. Using signed distance fields makes it possible to avoid the problem of representing nonsmooth functions as the signed distance is smooth across boundaries. [Perry and Frisken 2005; Frisken and Perry 2006] introduced vector distance fields with specialized cells which are capable of representing corners, overlapping objects, and thin features, for example, stems, without excessive cell subdivision, but still does not allow for resolution-independent representation of arbitrary curve networks (e.g., open curves or curves with joints as shown on Figure 1, right). These patents also describe a GPU-based implementation. A crucial element of our approach is an *unsigned* distance function which vanishes exactly at the feature curves, extending distance-based techniques to arbitrary curve networks.

Unlike Perry and Frisken [2005], we use parametric form as the primary feature representation. This helps representing curve intersections in a direct way and considerably simplifies ensuring C^1 continuity across cell boundaries; conversion from commonly used vector graphics formats is more straightforward.

Real-Time Curvilinear Discontinuities. Textures with curvilinear features look more appealing than their analogues represented by the networks of linear segments. Direct representation of curvilinear features is essential when close-up rendering of textures is needed. Among the already mentioned works, the sharp strokes [Salisbury et al. 1996], feature-based textures [Bala et al. 2003; Ramanarayanan et al. 2004], and bixels [Tumblin and Choudhury 2004] accept curves as their discontinuity features. However, GPU adaptation of the latter algorithms is relatively difficult.

Tarini and Cignoni introduced *the pinchmap* [Tarini and Cignoni 2005], with which smooth textures with curvilinear discontinuities are interpolated in realtime without artifacts. The algorithm accepts arbitrary curves and is very efficient, but the feature curves cannot have intersections.

In the GPU algorithm [Loop and Blinn 2005], objects with boundaries in the form of quadratic and cubic spline curves are sharply rendered with infinite resolution. In our approach we also use quadratic splines for the discontinuous features. However, their feature representation is not suitable for interpolating distance function and gradient field around the features, essential for making discontinuity customizable as is possible for our feature curves.

Our feature curves are comparable to the concurrent work of Qin et al. [2006] on rendering font glyphs anti-aliased at arbitrary magnification. Similar to our technique, their approach is based on exact computation of a distance field to glyph boundaries approximated by line segments. The crucial difference is that we rasterize the distance field and its gradient during preprocessing (local approach), while Qin et al. [2006] looks up the closest feature and calculates the distance directly (global approach). This approach is difficult to generalize to resolution-independent curvy features and to handle multiple features meeting at a point.

Textures Representing Small-Scale Geometry. A variety of techniques were developed for representing fine-scale geometric detail with textures. Bump maps were invented by J. Blinn [1978]. Many

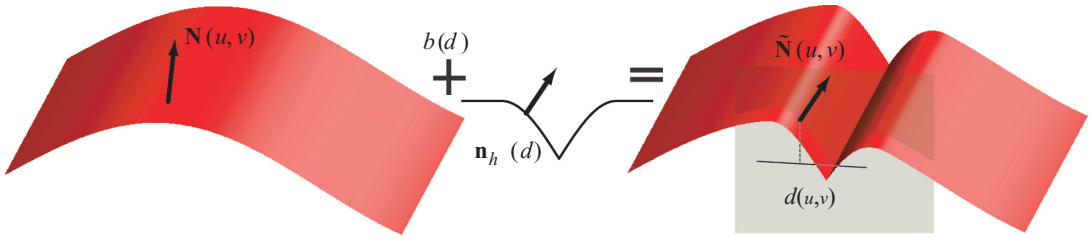

Fig. 2. Interpolation between the global smooth normal map and the local profile. We show updating height maps for clarity, but we interpolate normals, not the underlined mesh.

techniques for interactive rendering of bump maps were proposed, (e.g., Tarini et al. [2000] and references). With the appearance of programmable graphics hardware, use of basic bump mapping became commonplace. Bump map appearance can be improved by horizon maps, a technique for self-shadowing of bump maps, introduced in Max [1986, 1988]. Sloan and Cohen [2000] described how horizon maps can be accelerated using hardware. Parallax mapping and steep parallax mapping [McGuire and McGuire 2005] aim to reduce another artifact of bump maps: incorrect behavior when a surface is tilted. A more consistent way to address this problem is by using relief maps [Oliveira et al. 2000]. In many methods for high-quality rendering of normal maps, one needs to make a transition between different rendering modes as in Becker and Max [1993]. Enhancements to lighting of bump maps necessary for such transitions are described in Heidrich et al. [2000]. Displacement maps, introduced in Cook [1984], and relief maps [Oliveira et al. 2000] are a more advanced form of representing fine-scale geometry for flat surfaces (extensions to arbitrary surfaces were presented in Policarpo et al. [2005]). Distance maps [Donelly 2005] significantly improve rendering quality of small details by making use of volumetric distance field to the closest surface features while maintaining real-time performance of displacement maps.

While techniques for interactive rendering of displacement maps were recently proposed [Wang et al. 2003, 2004], these require large precomputed data sets. Interactive rendering of relief maps requires less data. While we describe our technique in the context of normal maps, it can be applied to rendering of relief maps representing the same type of geometry.

Texture and Detail Synthesis. As the application of our technique results in a combination of sampled and procedurally defined normals, our work is related to work in texture synthesis. Procedural bump maps first appeared in Perlin [1985]. There were a large number of papers on nonparametric synthesis from examples; these techniques often can be applied to produce bump maps [Zhang et al. 2003]. Our technique addresses a specific case of the problem of adding high-resolution detail to an image, focusing on sharp curvilinear linear features. A general approach to this problem can be found in Ismert et al. [2003].

To summarize, we believe our method to be the only real-time technique combining the following features: a) C^1 sharp feature curves at arbitrary resolution, b) open and intersecting feature curves, and c) smooth (away from feature curves) distance function and distance function gradient, enabling user-defined distance-based feature profiles. For linear features, our method is fully interactive.

3. OVERVIEW

The input to our algorithm consists of a network of feature curves representing desired texture sharp features or discontinuities, a texture map to which feature curves are added, and a one-dimensional profile defining how the texture is modified near the feature curves. The curves are represented by nondegenerate (i.e., no two control points coincide) quadratic Bezier segments. No further restrictions are imposed on the input.

At the *preprocessing* stage, we convert the curves to a texture-based representation (Section 4), and create additional textures representing the distance field and its gradient (Sections 3.2 and 4). The feature curves are split into texel-sized *discontinuity segments*, defined by *discontinuity signatures*.

At the *rendering* stage the original texture, profile, and additional textures generated by preprocessing are used to compute texture values at arbitrary locations needed for rasterization.

The combination of preprocessing and real-time algorithms aims to approximate, as closely as possible, a desired piecewise-smooth function (color, normal direction, or any other quantity encoded in the texture) defined by the combination of the feature curves, profiles, and a smooth map interpolating texture values.

In the rest of the algorithm description, we focus on normal maps, although the algorithm can be applied to other types of textures with minor modifications.

3.1 Normal Maps with Features

We start with a precise definition of the normal field we aim to approximate.

Suppose we are given a normal map, encoded in a texture, $\mathbf{N}(u, v) : [0..U_{max}, 0..V_{max}] \rightarrow R^3$. We split the map into two components $\mathbf{N}(u, v) = [N_x, N_y, N_z] = [\mathbf{N}_{xy}, N_z]$, where $\mathbf{N}_{xy}(u, v)$ is the 2D projection of the normal to the object's tangential plane at a point $p(u, v)$. Only $\mathbf{N}_{xy}(u, v)$ needs to be represented explicitly.

Let the distance to the closest feature curve from $p(u, v)$ be $d(u, v)$ and let the gradient of $d(u, v)$ be $\nabla d(u, v)$. Let $h(d)$ be the user-specified profile defining displacement of the fine-scale surface from the coarse geometry to which the texture is applied. In the simplest case, it depends only on the distance to the feature set d but may also depend on the closest point on the feature curve and other parameters.

We use a blending function $b(d)$ to merge given normals with the normals derived from the crease profile $h(d)$. The process of obtaining a new normal, given by $\tilde{\mathbf{N}}(d) = (1 - b(d))\,\mathbf{n}_h(d) + b(d)\mathbf{N}(d)$, is illustrated in Figure 2, where $\mathbf{n}_h$ is a normal of the crease profile pointing toward the feature curve. In the formulas, we omit the dependence of d on (u, v) where it is clear. We choose $b(d)$ to satisfy $b(d) = 0$ for $d \leq w_0$ and $b(d) = 1$ for $d > w$, where w_0 is the half-width of a band around the feature curve, for which the original normal map has no effect (w_0 can also be zero), and w is the feature curve width.

Assuming that the projection of profile normal $\mathbf{n}_h$ is aligned with the distance gradient $\nabla d(u, v)$, the (nonnormalized) desired

Fig. 3. Left: Distance function singularities for a rectangle: the medial axis is depicted in blue, and vectors denote the gradient discontinuity areas. Right: Distance function level lines and close-up rendered by using our techniques.

continuous normal map $\tilde{\mathbf{N}}(u, v)$ is

$$\begin{aligned} \tilde{\mathbf{N}}_{xy}(u, v) &= -(1 - b(d))h'(d)\nabla d + \tilde{b}(d)\mathbf{N}_{xy}(u, v) \\ \tilde{N}_z(u, v) &= (1 - b(d)) + \tilde{b}(d)N_z(u, v), \end{aligned} \tag{1}$$

where $\tilde{b}(d) = b(d)\sqrt{h'(d)^2 \cdot \nabla^2 d + 1}$

Our algorithms are designed to approximate the function defined by (1), maintaining precise behavior at the feature curves. For the rest of the paper we assume working in the texture domain $[0..U_{max}] \times [0..V_{max}]$, unless a different domain is specified. The domain is split into texels $T_{ij} = \Delta U\,[j..j + 1) \times \Delta V\,[i..i + 1)$. We use $\mathbf{p}_t = [u, v]$ to denote a sample in the texture domain. We exclude the right and top boundaries from each texel, except the right and top boundary rows of texels, such that each point in the domain belongs to a single texel.

We also assume that meshes have nice parameterizations, so that texture mapping does not noticeably distort the feature curves.

3.2 Distance Functions

The distance map representing the unsigned distance function is central to our algorithm: it determines the locations of the discontinuity curves and the distance to these curves. The direction of the gradient of the distance function for many feature profiles heavily influences the resulting value as can be seen from Equation (1). To ensure rendering quality, we would like them to satisfy the following distance function requirements (DFR):

DFR$_1$. (a) the approximate distance function should be exactly zero at feature curves so that sharp features can be created; (b) it should vary continuously; (c) it should remain close to the precise distance function near feature curves in order to be able to control feature width correctly;

DFR$_2$. the gradient of the distance function should be continuous away from feature curves;

To understand the difficulties with approximating the distance function on the grid, we observe that the Euclidean distance function has two types of singularities where the gradient is discontinuous: distance zero curves, and medial axis curves, that is, sets of points which are equidistant from two or more points on feature lines (Figure 3).

We represent the first type of singularity explicitly. Most singularities of the second type are far from feature curves, and the distance function does not affect the result in these locations. However, at corners, the medial axis meets the feature lines, so we need to address the gradient discontinuities there.

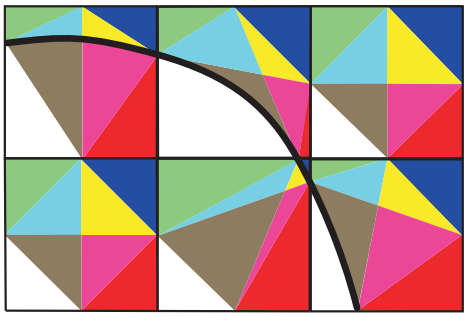

Fig. 4. Interpolation domains for the distance function near a feature line shown in different colors.

To satisfy our requirements, we compute the approximate distance function and its gradient from samples as follows:

(1) Within the texel the distance function $d(u, v)$ is linearly interpolated on curved triangular subtexel domains (Figure 4) aligned with feature curves to ensure that it is zero at these lines.

(2) The gradient $\nabla d(u, v)$ is also interpolated linearly. Note that this is *not* equivalent to computing the gradient of the interpolated distance function, as the latter would be piecewise constant.

(3) The gradient samples on the texture grid $\nabla \hat{d}_{ij}$ are smoothed away from the feature lines, to eliminate discontinuities at the medial axis.

The construction of a specific representation of the distance map is discussed in detail in Section 4. The real-time evaluation of distances and gradients is discussed in Section 5.

4. PREPROCESSING

Preprocessing has two goals: to simplify the connectivity of the feature curve network so that it can be represented using a fixed number of curve segments per texel, and to ensure that the simplification process results in a curve network with no self-intersections and sufficiently smooth curve segments. The former is achieved by a snapping process locally modifying curve segments overlapping a texel. The latter requires solving a collection of local constrained optimization problems. While computationally relatively expensive, the latter step is essential for obtaining a usable simplified curve

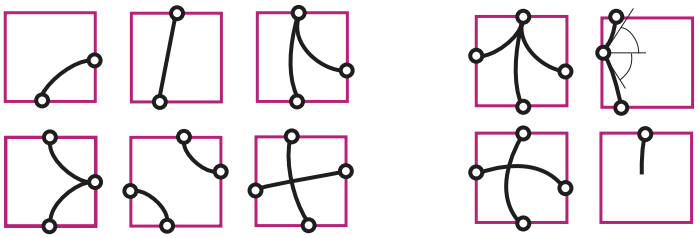

Fig. 5. Left: Valid discontinuity configurations for a texel. Right: Prohibited discontinuity configurations.

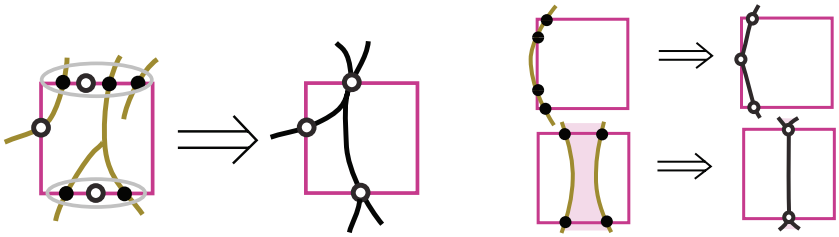

Fig. 6. Left: Edge point transformations. Black points are replaced by their center of mass (white points) to satisfy the first rule, identified discontinuity segments are deleted to follow the second rule, no hanging curves inside a texel according to the third rule. Right: Transformation artifacts: (top) reduction of a curve to two linear discontinuity segments, (bottom) change of topology.

network; this step is unnecessary only if linear curve segments are considered.

For every texel overlapping a curve, we define a *discontinuity configuration* and a *signature*. The discontinuity configurations and signatures are stored as additional textures. A discontinuity configuration defines the number of discontinuity segments within the texel and which edges are crossed by the segments. The discontinuity signature stores more detailed information: exactly where the discontinuity passes through an edge and its tangential direction.

In principle, local configurations of discontinuities can be arbitrarily complex: any number of curves can overlap a texel. Following Tumblin and Choudhury [2004] and Sen [2004], we reduce the number of allowed configurations, avoiding introduction of points in the center of texels. Valid discontinuity configurations for a single texel are defined by the following three rules:

—One boundary edge of a texel may be crossed by discontinuity curves at no more than one point.

—There are no more than two discontinuity curves inside any texel.

—A discontinuity curve ends on a texel boundary.

We call a portion of the discontinuity curve overlapped by a texel as a *discontinuity segment*. Valid configurations of discontinuity segments are shown in Figure 5 (left).

Our choice of the set of valid configurations aims to achieve a good balance between generality of networks that can be represented exactly, required texture size, and algorithm complexity. The major limitation of our choice of valid configurations is that no more than two curves may overlap a texel, curve termination inside texels is not allowed, no more than four curves can share endpoints, and curves sharing a point on a texel boundary also have to share a tangent [see Figure 5 (right)].

The algorithm modifies discontinuity curves locally and transforms an arbitrary configuration into the one that satisfies our requirements. Figure 6 (left) illustrates the changes in the discontinuity map the algorithm performs.

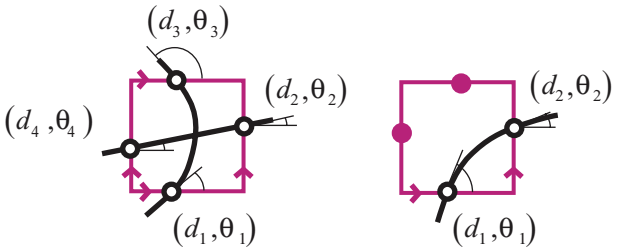

Fig. 7. Discontinuity signature: an encoding of the endpoints and tangential vectors along discontinuity segments.

Once the transformation is complete, every texel has at most two discontinuity segments, so we can store texels' configurations in the configuration map as quadruples $\hat{C}_{ij}$. Configuration $\hat{C}$ is [0, 0, 0, 0] for an empty texel, has two pairs of endpoint edge indices for a texel with two discontinuity segments: $\hat{C} = [(l_1, m_1), (l_2, m_2)]$, and two copies of endpoint edge indices of the only discontinuity segment within a texel with one discontinuity: $\hat{C} = [(l_1, m_1), (l_1, m_1)]$. The edge indices l_s, m_s, $s = 1, 2$ are in the range $1 \ldots 4$ (edges are ordered clockwise starting with from the south edge).

Based on the resulting configuration map and by using the adjusted network of feature segments, we assign discontinuity signatures for all affected texels: each texel is annotated with eight numbers representing the four distances from the left/bottom corners to discontinuity segment endpoints along the texel boundary, and four angles between tangential vectors at the endpoints and the horizon (Figure 7). The default distance/angle pair (0.5, 0) is assigned to every edge without a discontinuity point on it. For example, the default discontinuity signature for texels with no segments is $\hat{S} = [(0.5, 0), (0.5, 0), (0.5, 0), (0.5, 0)]$. If there is a segment passing the south and the east edges, then $\hat{S} = [(d_1, \theta_1), (d_2, \theta_2), (0.5, 0), (0.5, 0)]$. Finally, $\hat{S} = [(d_{l_1}, \theta_{l_1}), (d_{m_1}, \theta_{m_1}), (d_{l_2}, \theta_{l_2}), (d_{m_2}, \theta_{m_2})]$ for texels with two discontinuities.

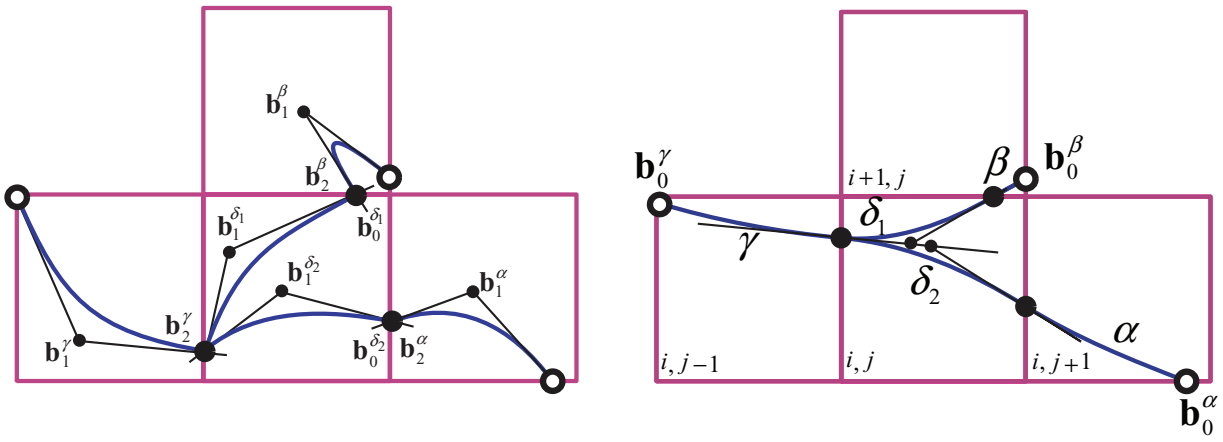

Fig. 8. Left: an example of the initial configuration for spline energy optimization, given by Equation (2), within an invalid central texel with two discontinuity segments. All the locations of filled circles are optimized. Right: optimal positions of the segments α,β,γ,δ_1, and δ_2. All the locations of open circles are fixed during optimization.

The conversion process proceeds as follows. Initially, default empty configurations are assigned to all texels. Each feature curve is scan-converted to the texture to determine all texels it intersects (visited texels). For each visited texel, we check if there are one or two intersections of the curve with the texel's edges. In the former case, only the intersection point is used in the conversion process, while the part of the curve inside the texel is disregarded. In the latter case, we use a table of texel transformations to adjust the configuration; the table maps a pair (valid texel configuration, pair of edge indices for the new segment endpoints) to a transformation defining a new configuration. Any such transformation maps a valid texel configuration to another valid texel configuration and updates the texel's discontinuity signature by calculating the center of mass of all discontinuity points included so far per every edge (e.g., Figure 6 (left)).

The advantage of this algorithm is that it is easy to see that it will always produce a valid set of configurations for all pixels; however, it may introduce artifacts in rare situations like those depicted on Figure 6 (right).

While texel configurations are guaranteed to be valid, in the case of curvilinear features, not every resulting discontinuity signature is valid. Some configuration-signature pairs may produce no Bezier segment at all (e.g., when their θs are equal), produce a Bezier segment which leaves the interior of the texel, or C^1 continuity is not maintained between connected discontinuity segments. To address these issues, we transform all invalid signatures by running a constrained optimization on the curve segments within the affected texels and their immediate edge-neighbors.

We describe the details of the optimization on a texel with two discontinuity segments sharing one edge-point; other configurations are optimized in a similar way. We minimize an energy of a set of discontinuity segments including segments within the affected texel (e.g., Bezier segments δ_1, δ_2 within the central texel in Figure 8) and connected segments in the adjacent texels (segments α, β, and γ within the east, north, and west immediate neighbors in Figure 8).

To make the process local, we fix the signature angles and positions in the adjacent texels located on the edges not shared with the central texel (open circles on the Figure 8). Locations of control points (excluding those coincident with fixed signature locations) of all Bezier segments are the variables in the optimization (filled circles on Figure 8). Linear inequality constraints keep all segments inside their original texels, and nonlinear equality constraints ensure tangent continuity with adjacent segments (δ_1 with β and γ, and δ_2 with α and γ on Figure 8). The complete optimization problem is

$$\left\{\begin{array}{l} E_{conf} = E_\alpha + E_\beta + E_\gamma + E_{\delta_1} + E_{\delta_2} \to \min \\ \text{subject to linear inequalities} \\ \quad \mathbf{b}_1^\alpha \in T^o_{i,j+1},\ \mathbf{b}_1^\beta \in T^o_{i+1,j},\ \mathbf{b}_1^\gamma \in T^o_{i,j-1}, \\ \quad \mathbf{b}_2^\alpha \in \partial_e T_{i,j},\ \mathbf{b}_2^\beta \in \partial_n T_{i,j},\ \mathbf{b}_2^\gamma \in \partial_w T_{i,j},\ \mathbf{b}_1^{\delta_1} \in T^o_{ij},\ \mathbf{b}_1^{\delta_2} \in T^o_{ij} \\ \text{subject to nonlinear equalities} \\ \quad P(\mathbf{b}_1^\alpha, \mathbf{b}_2^\alpha, \mathbf{b}_1^{\delta_2}) = P(\mathbf{b}_1^\beta, \mathbf{b}_2^\beta, \mathbf{b}_1^{\delta_1}) = 0 \\ \quad P(\mathbf{b}_1^\gamma, \mathbf{b}_2^\gamma, \mathbf{b}_1^{\delta_1}) = P(\mathbf{b}_1^\gamma, \mathbf{b}_2^\gamma, \mathbf{b}_1^{\delta_2}) = 0 \\ \text{subject to linear equalities} \\ \quad \mathbf{b}_0^{\delta_1} = \mathbf{b}_2^\beta \\ \quad \mathbf{b}_0^{\delta_2} = \mathbf{b}_2^\alpha,\quad \mathbf{b}_2^\gamma = \mathbf{b}_2^{\delta_1} = \mathbf{b}_2^{\delta_2}. \end{array}\right. \tag{2}$$

In the optimization problem formulation, T^o and $\partial_{e/n/w} T$ denote the interior and the (east/north/west) boundary of texel T, and $\mathbf{b} \in T^o_{i,j}$ is explicitly expressed by the inequalities $\left\{0 < \hat{b}_x - j < 1,\ 0 < \hat{b}_y - i < 1\right\}$. The points on the texel's edges, e.g. the west edge, $\mathbf{b} \in \partial_w T_{i,j}$ are identified using a function P as $\left\{P(\hat{\mathbf{b}}, [j,i], [j,i+1]) = 0,\ 0 < \hat{b}_y - i < 1\right\}$, where $\hat{\mathbf{b}} = [b_x/\Delta U, b_y/\Delta V]$ and $P(\mathbf{p},\mathbf{q},\mathbf{r})$ is zero whenever its arguments lie on the same line. We use $P(\mathbf{p},\mathbf{q},\mathbf{r}) = (\mathbf{p}-\mathbf{q})\cdot(\mathbf{q}-\mathbf{r}) - |\mathbf{p}-\mathbf{q}|^2|\mathbf{q}-\mathbf{r}|^2$.

The energy E_χ of a quadratic Bezier segment χ given in Bernstein form [Farin 1997], $\mathbf{B}^\chi(t)$, with control points $\mathbf{b}_i^\chi$, $i = 1, 2, 3$, is the standard thin-plate energy:

$$E_\chi = \int_0^1 \left|\frac{d^2}{dt^2}\mathbf{B}^\chi(t)\right|^2 dt = 4\left|\mathbf{b}_0^\chi - 2\mathbf{b}_1^\chi + \mathbf{b}_2^\chi\right|^2. \tag{3}$$

We use sequential quadratic programming to solve the optimization problem for each local configuration separately. An example is shown in Figure 8.

If only linear discontinuities are presented in the original texture, then every resulting discontinuity signature is valid. In this case, the preprocessing algorithm avoids signature optimization and is sufficiently fast for interactive feature line modification.

Rasterization of Distance Field and Its Gradient. The distance field values and the gradient values are calculated at the texel corners simultaneously with configuration transformations. We initialize the distance field with some "large" value and merge the local distance fields of individual features one by one by keeping the smallest distance values within the merging domains. Gradient values corresponding to the smallest distances are stored in the gradient texture.

We smooth the resulting gradient texture $\nabla\hat{d}_{ij}$ by using constrained Laplacian smoothing (e.g., Taubin [1995] for similar techniques). Specifically, each vertex moves toward the barycenter of its neighbors, excluding the neighbors on the other side of the discontinuity. We found that approximately 10 smoothing iterations is sufficient. As is well known, Laplacian smoothing quickly eliminates high frequency features, while slowly reducing low frequency features, which is the desired result in our case.

The significant improvement in appearance due to smoothing the gradient field is shown in Figure 12: compare two images on the right of the top row. Note that no smoothing is done on the distance function itself; we found it important to keep it unchanged, for example, for maintaining approximately constant width of creases when desired.

5. REAL-TIME RENDERING

5.1 Normal Computation During Rendering

The input to the interactive part of our algorithm includes

—the distance map $\hat{d}_{ij}$ and the gradient map $\nabla\hat{d}_{ij}$,

—the discontinuity configurations $\hat{C}_{ij}$ and discontinuity signatures $\hat{S}_{ij}$,

—any other texture maps used for rendering, including the normal map $\hat{\mathbf{N}}_{ij}$,

—user-defined crease profile $h(d)$. It may be stored in a one-dimensional texture $\hat{h}_k$.

Our rendering algorithm, implemented as a pixel shader, uses the normal map, per-texel discontinuity configurations and signatures and profiles to compute the normal values. Once all necessary quantities are interpolated, Equation (1) is used to obtain the normal. The goal is to interpolate the distance field, its gradient, and normal texture in a way that respects feature curves. In particular, there should be no averaging of values across the discontinuity segments. We achieve this by using a three-stage interpolation procedure which partition the texel.

Discontinuity segments partition a cell into several domains. For valid configurations, the segments are topologically equivalent to a subset of the boundaries of an eight-triangle partition of the domain (Figure 4). For example, for a single discontinuity segment connecting two adjacent sides, one domain is a union of seven curvilinear triangles on one side of the segment and the other is one triangle located on the opposite side of the segment. For any configuration, we generate an *eight-triangle partition*, adding additional vertices when necessary. The vertices of the partition triangles located on the texel's edges (*edge vertices*) are either determined by the discontinuity endpoints (as for vertices q_b and q_d on Figure 9) or placed at the edge center when there is no discontinuity passing such edge (as for vertices q_a and q_c on Figure 9). The common *central vertex* r_m is an intersection of horizontal segment $\{q_b, q_d\}$ with vertical segment $\{q_a, q_c\}$. Partition triangle vertices are only created temporarily during the rendering step.

Our algorithm can be summarized as follows: we interpolate/extrapolate corner values of the texel contained in the same domain

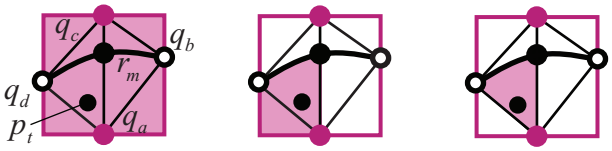

Fig. 9. Steps in locating the partition's triangle containing $\mathbf{p}_t$; discontinuity feature passes through points q_b and q_d.

of the discontinuity partition as $\mathbf{p}_t$ (note there may be only one corner value available) to obtain the values at the vertices of the curvilinear triangle containing $\mathbf{p}_t$. Then, we obtain values at $\mathbf{p}_t$ by interpolating inside the triangle.

The curvilinear triangle which contains $\mathbf{p}_t$ is localized in two steps: first, by finding one of the four quadrants which contains the point, and second, by choosing the triangle containing $\mathbf{p}_t$ within the quadrant. This process is illustrated in Figure 9. It requires running a test to determine on which side of a discontinuity curve a point is (Section 5.2).

To determine the values at the edge vertices, we interpolate the samples at texel corners and then use the values at edge vertices to interpolate the value at the central vertex. The values at the edge and central are obtained using only samples from *reachable* texel corners: the corners in the same discontinuity partition domain with $\mathbf{p}_t$.

Once the samples at triangle corners are obtained, triangle's corners are interpolated to calculate the values at point $\mathbf{p}_t$: $d(u, v)$,$\nabla d(u, v)$, and $\mathbf{N}(u, v)$. Our procedure ensures the distance function is exactly zero on the discontinuity segments: if two edge vertices are on a discontinuity segment, the distance function value at these vertices is zero; the value at the central vertex is interpolated from these two values and is also zero. Our technique for interpolation on curvilinear triangles, described in Section 5.3, ensures all intermediate values along the discontinuity segment connecting an edge vertex with the central vertex are also zero.

5.2 Side Test

The *side test* determines on which side of a curve a point is located. It is straightforward in the linear case, so we focus on the curvilinear case.

To perform the side test for quadratic Bezier curves, we convert them to an implicit form, $f_B(x, y) = 0$. By using a well-known result from algebraic geometry [Cox et al. 1998], we find the implicit form of a Bezier segment, written in power basis as $\mathbf{B}(s) = [q_x(s), q_y(s)]$ with $q_i(s) = a_i s^2 + b_i s + c_i$, by equating the resultant of the two polynomials $q_x(s) - x$ and $q_y(s) - y$ to zero:

$$f_B(x, y) = \mathrm{Res}_{2,2}[q_x(s) - x; q_y(s) - y] = \begin{vmatrix} -a_x & -b_x & x - c_x & 0 \\ 0 & -a_x & -b_x & x - c_x \\ -a_y & -b_y & y - c_y & 0 \\ 0 & -a_y & -b_y & y - c_y \end{vmatrix} = 0. \quad (4)$$

The two sides of the curve are defined by the sign of f_B. The side test, based on implicitization, may yield incorrect results for highly curved segments: a curved segment may pass a unit square twice. For example, a Bezier segment $B[\mathbf{b_0}, \mathbf{b_1}, \mathbf{b_2}]$ on Figure 10 (left) has a parabola γ as its implicit form which intersects the texel a second time at the NW corner such that samples $\mathbf{p_1}$ and $\mathbf{p_2}$ are on the same side of the implicit curve but on different sides of the Bezier segment B. We use the following simple test to detect such cases.

Consider a parabola which corresponds to the Bezier segment with control points, $\mathbf{b_0} = [-1/2, 0]$, $\mathbf{b_1} = [x_0, y_0]$, and

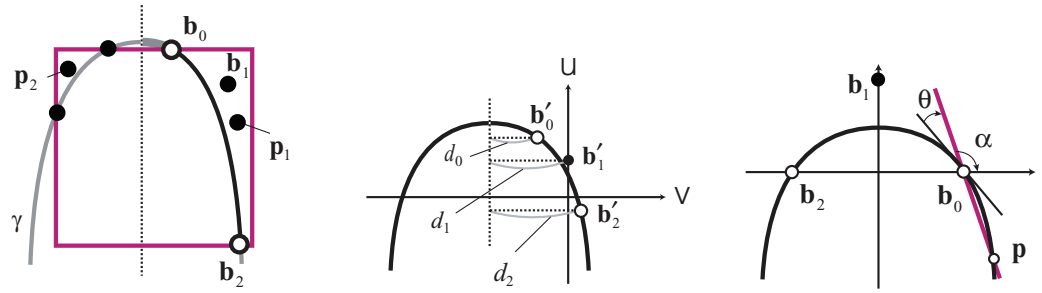

Fig. 10. Left: An example of an implicit form which assigns the same side for two points separated by a discontinuity. Center-right: resolving side test ambiguity for the case $x_0 \neq 0$ (center), and $x_0 = 0$ (right).

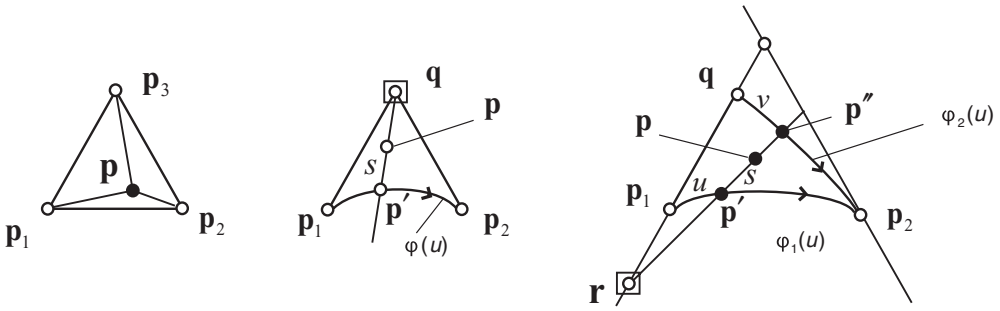

Fig. 11. Interpolation schemes at a sample point **p** for triangles of different edge types. Left: a straight-edge triangle. Center: a 1C-triangle with one curvilinear edge. Right: a 2C-triangle with two curvilinear edges.

$\mathbf{b_2} = [1/2, 0]$. It has the following unique form up to a constant factor:

$$4(-yx_0 + xy_0)^2 + 2yy_0 - y_0^2 = 0. \tag{5}$$

After applying a linear transformation $u = (x_0x + y_0y)/|\mathbf{b}_1|$, $v = (-y_0x + x_0y)/|\mathbf{b}_1|$, the parabola becomes

$$\frac{2|\mathbf{b}_1|^3}{y_0^2}v^2 + \frac{x_0}{y_0}v - \frac{|\mathbf{b}_1|}{2} = -u. \tag{6}$$

The larger the leading coefficient is in Equation 6, the narrower the parabola will be, and more likely that both branches of the parabola will intersect the texel.

We consider the following two cases: $x_0 \neq 0$ and $x_0 = 0$. In the first case, the distance from a point $\mathbf{b}'_1$, the image of $\mathbf{b}_1$ under the linear transformation, to the parabola's medial axis, d_1, equals $|x_0y_0|/4|\mathbf{b}_1|^3$, and the corresponding distances d_0, d_2 for $\mathbf{b}'_0$ and $\mathbf{b}'_2$ are $|x_0y_0|/4|\mathbf{b}_1|^3 \pm |y_0|/2|\mathbf{b}_1|$ [Figure 10 (center)]. If the smallest such distance is larger than the diameter of the unit square, then the second parabola's branch never intersects the texel, making the side test unambiguous. Therefore, we use the following as a safety test:

$$\min(d_0, d_1, d_2) > \sqrt{2}. \tag{7}$$

If the inequality (7) fails, we update the position of the second control point $\mathbf{b}_1$ so that $x_0 = 0$, leading us to the second case.

The second case is always unambiguous if we assume a lower bound on possible values of the angles θ between the tangential direction and the texel edge [Figure 10 (right)]. If a line, representing a texel's edge with $\mathbf{b}_0$ on it and having a parametrization $[1/2, 0] + [\cos\alpha, \sin\alpha]t$, were to intersect the parabola $y = -2y_0x^2 + y_0/2$, derived from Equation (5) with $x_0 = 0$, at point **p**, the distance $d(\mathbf{b}_0, \mathbf{p})$ to the intersection point is

$$d(\mathbf{b}_0, \mathbf{p}) = \left(\frac{1}{2y_0}\tan\alpha + 1\right)^2 \left(1 + \tan^2\alpha\right). \tag{8}$$

This distance is always larger than 1 (guaranteeing that the parabola does not intersect the texel again) if $\tan\alpha < -4y_0$. As the derivative of the parabola equation at point $\mathbf{b}_0$ equals $(-2y_0)$, we can approximate the smallest possible angle $\theta_{\min} = \arctan(-2y_0) - \arctan(-4y_0) \approx 2y_0/(1 + 16y_0^2)$. To ensure this condition holds, we always convert curvilinear discontinuity segments with small values of $\theta < \theta_{\min}$ to straight segments.

5.3 Interpolation in a Curvilinear Triangle

We apply three different interpolation techniques to reconstruct the unknown values inside a triangle given three values at the triangle's corners, depending on the following types of the triangle's edges: (a) straight-line triangle: all edges are straight lines; (b) curvilinear *1C-triangle*: one curvilinear edge represented by a quadratic Bezier segment; (c) curvilinear *2C-triangle*: two quadratic Bezier edges (Figure 11).

To find an unknown value $V(\mathbf{p})$ (i.e., distance, gradient, or normal) at a point **p** inside of a straight triangle $\langle \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3 \rangle$, given corner values $V_i \equiv V(\mathbf{p}_i)$, we apply the standard linear interpolation based on the barycentric coordinates $\{u_i\}$ of $\mathbf{p}_i$: $u_i = S(\mathbf{p}_{i-1}, \mathbf{p}_{i+1}, \mathbf{p})/S(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$, $V(\mathbf{p}) = \sum_i u_i V_i$.

For a 1C-triangle $\langle \mathbf{p}_1, \mathbf{p}_2, \mathbf{q} \rangle$, where **q** is adjacent to the straight edges (Figure 11, center), we define a map from the unit square with coordinates (s, u) to the triangle's interior, being degenerate at the edge $s = 0$. For any point in the triangle, we obtain the values of s and u by intersecting a ray $[\mathbf{q}, \mathbf{p})$ with the curvilinear edge

$\{\mathbf{p}_1, \mathbf{p}_2\}$ to locate a point $\mathbf{p}'$. As the edge $\{\mathbf{p}_1, \mathbf{p}_2\}$ is parameterized by a quadratic function $\vec{\varphi}(u)$, the system of equations for s and u is

$$\begin{aligned} \mathbf{p}' &= \vec{\varphi}(u) \\ s &= \left|\mathbf{p}' - \mathbf{p}\right| / \left|\mathbf{p}' - \mathbf{q}\right|. \end{aligned} \tag{9}$$

Solving the system reduces to solving a quadratic equation. We note that there is always exactly one intersection, which can be shown by using the variation-diminishing property of Bezier curves. Finally, we use bilinear interpolation to compute $V(\mathbf{p})$: $V(\mathbf{p}) = s \bullet \left(u \bullet (V_1, V_2), V_q\right)$, where $u \bullet (V, W) \equiv (1-u)V + uW$.

We use a different approach for 2C-triangles, as a simple radial parametrization of the type we use for 1C triangles is impossible.

For a 2C-triangle $\langle \mathbf{p}_1, \mathbf{p}_2, \mathbf{q} \rangle$ with curvilinear edges $\{\mathbf{p}_1, \mathbf{p}_2\}$ and $\{\mathbf{q}, \mathbf{p}_2\}$, parameterized by $\vec{\varphi}_1(u)$ and $\vec{\varphi}_2(v)$ respectively, we choose an auxiliary point $\mathbf{r}$ on the ray passing the straight edge $[\mathbf{p}_1, \mathbf{q}]$. Consider the ray $[\mathbf{r}, \mathbf{p})$ going through point $\mathbf{p}$. Again, by the Bezier curve variation-diminishing property, one can show there will be exactly one intersection with curvilinear edges. For the sample $\mathbf{p}$, we find the intersection points $\mathbf{p}'$ and $\mathbf{p}''$ of the ray $[\mathbf{r}, \mathbf{p})$ with the curvilinear edges by applying the well known Bezier clipping algorithm described in Nishita et al. [1990]. Then, s is found the same way as it is done for a 1C-triangle by solving the following system:

$$\begin{aligned} \mathbf{p}' &= \vec{\varphi}_1(u) \\ \mathbf{p}'' &= \vec{\varphi}_2(v) \\ s &= \left|\mathbf{p}' - \mathbf{p}\right| / \left|\mathbf{p}' - \mathbf{p}''\right|, \end{aligned} \tag{10}$$

while the final interpolation formula is given by $V(\mathbf{p}) = s \bullet \left(u \bullet (V_1, V_2), v \bullet \left(V_q, V_2\right)\right)$.

6. IMPLEMENTATION AND RESULTS

We have implemented the real-time interpolation algorithm as a fragment shader in Cg and tested the implementation on an NVidia GeForce 7800GS AGP 256M, running on Pentium 4, 2.8MHz. The frame rates that we have obtained for 512×512 images are in the range from 25 to 215 frames per second (fps) for interpolating curvilinear features, and from 65 to 380 fps for interpolating features approximated by linear segments. Such wide ranges is the result of sensitivity of the performance to the ratio of discontinuity pixels, configuration complexity of feature curves being rendered in the current frame, and to the size of the scene.

A normal map with feature curves is rendered in two passes. We run the interpolation shader in the first pass. It interpolates the normal texture with discontinuities represented by feature textures and stores the resulting normals in the Framebuffer object (FBO) [Green 2005] attached to the current fragment output. FBO is faster during switching than its former alternative p-buffer, and it maintains up to 32 bit float images as the fragment destination texture. On the second pass, the FBO with stored normals is detached from the fragment and is used as a general texture image that provides pixel normals for rendering.

The interpolation shader starts with reconstructing the feature curves inside a texel containing a texture sample encoded by the texture coordinates of a point currently observed by the fragment shader. It partitions the texel interior into 8 curvilinear triangles, and locates a triangle which covers the texture sample. It then propagates known distance/gradient/normal values from the texel's corners to the corners of the triangle (see Figure 9). Finally, the normal at the sample is interpolated within the triangle by applying the corresponding interpolating algorithm, described earlier in (Section 5.3). All the calculations are performed in single precision arithmetic (floats). More detailed description of the shader can be found in the Appendix.

Memory Consumption. We use two extra textures: RGBA float and 16-bit grayscale. The former encodes discontinuity signatures $\hat{S}_{ij}$: its "RG" component contains the south edge pair of the signature, and its"BA" component carries the west edge pair. The east and the north edge signature pairs are located in the corresponding "RG" and "BA" components of the east and the north texel neighbors. The latter stores discontinuity configurations $\hat{C}_{ij}$. Distance values are stored in the unused "A" component of the normal map texture. Therefore, our approach requires to consume in total extra 5 floats per texel.

Figure 12 compares different modes of rendering normal maps with sharp features, enabling different parts of our approach one-by-one. The obvious artifacts at the bottom of the crease on the images from the first column are eliminated by using a reduced version of our method which approximates the gradient field from the distance samples. However, the resulting normal field is not smooth along the texel edges as can be clearly seen at the high intensity spots in the second column images. This occurs because the interpolated distance field is only C_0 continuous across the texel edges. The images in the third column look more smooth at the spots after separating the distance and gradient interpolation into two independent processes. Finally, the artifacts along the distance medial axis are smoothed away by a preliminary filtering of the gradient field at the preprocessing stage.

Feature Curves. Figure 13 shows several examples of applying user-specified profiles. All discontinuity features are curvilinear (quadratic Bezier curves) and stored in the 128×128 textures (one of the examples shows texel sizes). The close-up views show that our technique results in few artifacts even in complex situations. The rendering frame rates for these images were 65 fps, with 75 fps for the close-up views and up to 215 fps when the object only partially covers the rendering fragment. These examples have a simple feature pattern and the smallest mesh size, which makes them the fastest among the examples with curvilinear features that follow.

Figures 14–16 show several images of models of feature-based normal maps rendered by our technique with different types of features and surface properties. 512×512 textures were used for rendering images on Figures 14, 16, while the snake mesh of Figure 15 was wrapped by a 1024×1024 texture.

The coke can with the fingerprints on its surface shown in Figure 14(a) is rendered at 46 fps (with at least 93 fps when the can is scaled to one-half of the image size). Its magnified version, shown on the right, runs as low as 26 fps. To clarify the reasons for inferior performance relative to the previous example, we considered the frame rate dependence on the level of complexity of feature curves and the size of underlying mesh.

We measured the *feature coverage ratio* (FCR) for the can image, and found a strong impact of the feature pattern complexity within the current rendering frame on the overall performance (we define FCR as a ratio of pixels whose texture samples are located within discontinuity texels to the total number of pixels in the current output frame). Indeed, the FCR was approximately 5.5% for the full view can image, while discontinuity features covered more than 19% of the current frame for the magnified version of the can. However, the FCR only partially explains the resulting performance drop. The images of the Chinese character "luck" on the top row of Figure 13 also have an FCR approximately equal to 5%, but they run almost two times faster than the full view image of the can. The obvious difference between the two images is in the locality of discontinuity pixels in the current frame. Usually, the performance

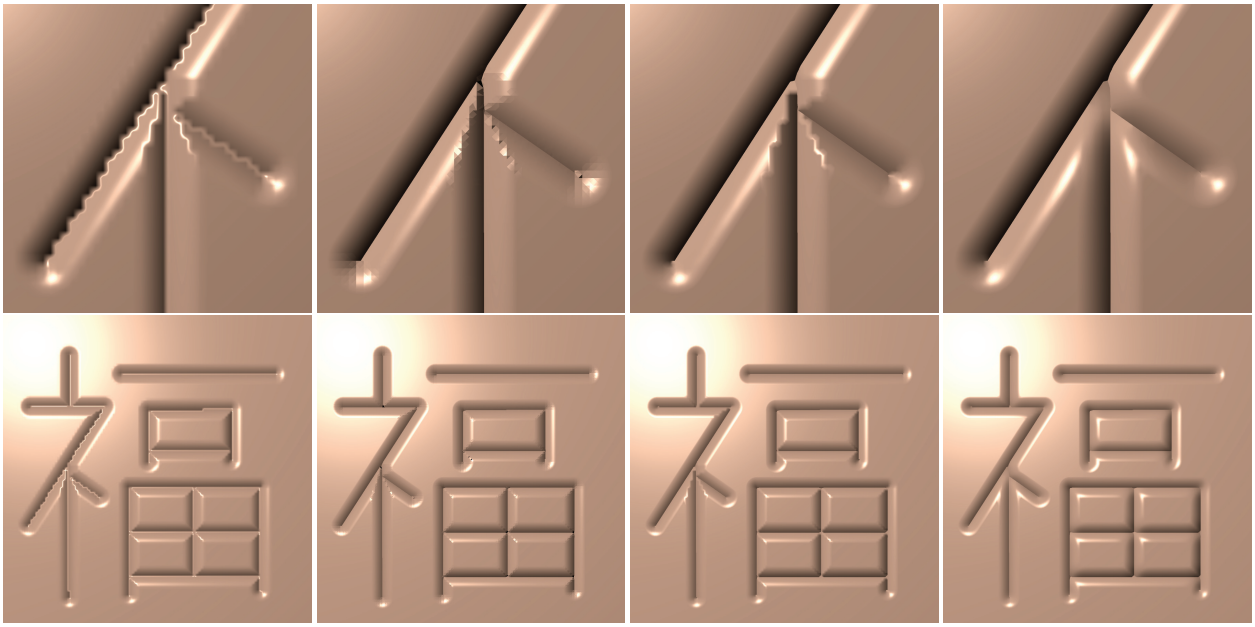

Fig. 12. The appearance of features for different interpolation methods, from left to right: standard bilinear interpolation of samples representing the profile; piecewise linear distance function and piecewise constant gradient computed directly from the distance function; piecewise linear interpolation for gradient and distance function, without constrained smoothing of the gradient field; same with smoothing.

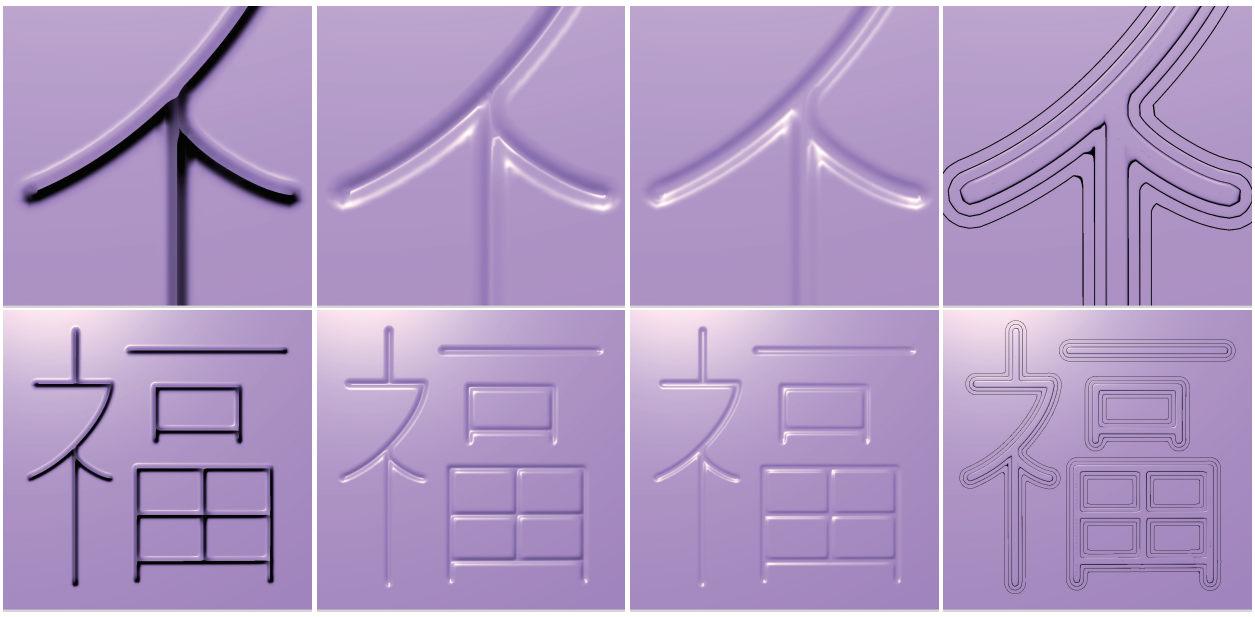

Fig. 13. Examples of user-defined profiles. The discontinuity map is the same in all cases; note that in some cases, the discontinuity is removed entirely: a spectrum of creases of variable sharpness is possible.

Fig. 14. Several examples of models with details added using our technique; a magnified view is shown for each texture.

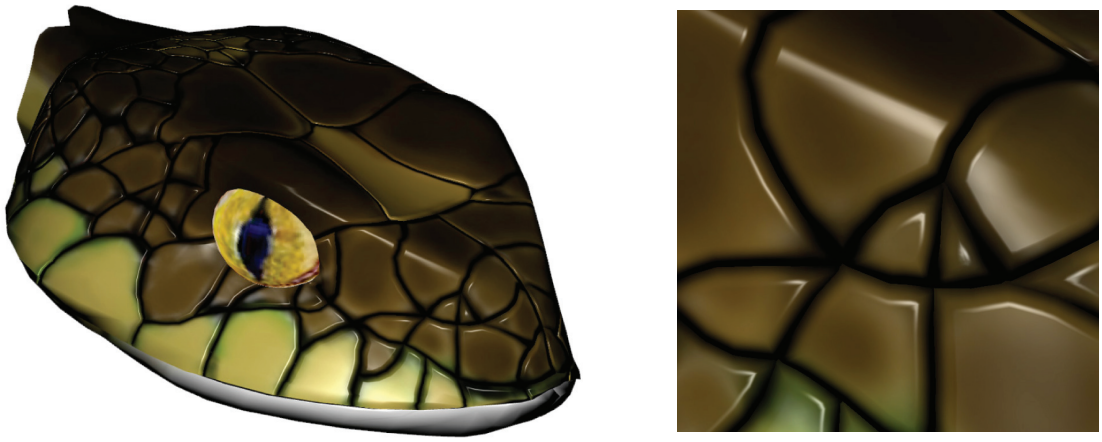

Fig. 15. An example of a snake model with curvilinear features approximated by a set of linear segments.

Fig. 16. An example of different profiles; a magnified view is shown for each texture.

Table I. Feature Curves Performance Measured in Frames Per Second (fps) and Superimposed to Feature Coverage Ratio (FCR) and the Underlying Mesh Size. "res" = Texture Resolution, "dscnt" = Memory Used for Storing Discontinuities

	"Luck" res.=128^2, dscnt.=163K, Fig. 13			Coke can res.=512^2, dscnt.=2.6M, Fig. 14 (a)				Escher's pattern res.=512^2, dscnt.=2.6M, Fig. 14 (b)	
	close-up	full view	full view zoom 1/2×	close-up	full view		full view zoom 1/2×	close-up	full view
fps	75	65	215	26	46	49	93	35	25
FCR	5	1.6	0.3	19.0	5.5	5.5	2.3	8	4
♯faces	2	2	2	4.2K	4.2K	2	4.2K	4.3K	4.3K

of GeForce 6/7 series cards suffer from frequently occurring incoherent branching in the pixel shaders. Our shader has a number of conditional statements, including the main one, which checks whether a pixel overlaps a discontinuous texel. The fragment of the magnified luck character image has many more coherent continuous pixels than the coke can image where continuous and discontinuous pixels are mixed. As a result, a combination of the FCR ratio with pixel coherence dominates overall performance.

The performance is only slightly sensitive to the size of the underlying mesh. The mesh of the coke can has 4.2K faces. However, performance improved only 6%, from 46 fps to 49 fps, for a two-triangle square with the same normal map applied.

The image of the plate with an Escher pattern [Figure 14(b)] runs as low as 25 fps (35 fps for the magnified image). Performance is relatively poor in this case versus other cases with comparable FCR because the pattern contains more discontinuity texels with two feature curves in them.

Table I summarizes the performance of our feature curve shader running with textures we just discussed. Pixel coherence and the value of the FCR are the parameters which greatly influence the performance, while the size of the mesh is a less important parameter.

Linear Features. Our algorithm can be easily transformed into a shader for interpolating normal maps with linear features, which runs significantly faster at any resolution. Indeed, there is no need for time-consuming curve reconstruction in step 2 (see Appendix); the side test for the linear segments does not require computationally intensive curve implicitization in step 3; and, an interpolation within the straight triangle does not require solving quadratic equations as in the case for curvilinear triangles in step 5. Moreover, our version of the shader for linear features has only one branching: the one which checks the type of the pixel. Memory expense decreases to 3 floats per texel as tangential directions need not be stored.

A piece of glass depicted in Figure 14(c) has a very intricate pattern of linear features with joints having as many as 7 meeting features. The resulting feature texture has 91% discontinuous texels. The rendering rate in this case is 222 fps (114 fps for the first magnified image, and up to 380 fps for the version scaled by 75%) with FCR being close to 37% (90% and 20.2% for these two cases).

We approximated feature curves by linear segments for the snake mesh depicted in Figure 15. It runs at 124 fps with 10% FCR, and at the same performance with 35% FCR in the case of the magnified version (the mesh has 3.3K faces). Approximate feature curves look smooth enough at certain magnifications. However, the corners are already noticeable in the magnified version, which is not yet the closest view.

Teapots with features created by Delaunay triangulation shown on Figure 16 are examples of using user-defined feature profiles. They illustrate that using different profiles may change the way the teapot is perceived. Both the full view image and its magnified version run at 75 fps, though their FCR ratios are different, equaling 25.4% and 89.2%, respectively. We found a clear dependence of linear feature performance on the mesh size, in contrast to a negligible size-performance dependence for feature curves. The teapot mesh (6.3K faces) is almost two times larger than the snake mesh. The magnified versions of both examples run faster when features are mapped to the plane (keeping the same FCR values): the teapot

Table II. Performance Measured for Linear Feature Shader. "res" = Texture Resolution, "dscnt" = Memory Used for Storing Discontinuities

	Glass res.=512^2, dscnt.=1.5M, Fig. 14 (c)			Snake head res.=1024^2, dscnt.=6.2M, Fig. 15			Teapot res.=512^2, dscnt.=1.5M, Fig. 16					
	close-up	full view	full view zoom 2/3×	close-up		full view	close-up		full view	full view reduced mesh		
fps	114	222	380	124	156	124	75	131	75	98	115	140
FCR	91	37	20.2	35	35	10	89.2	90.2	25.4	24.5	40.4	46.7
♯faces	2	22	2	3.3K	2	3.3K	6.3K	2	6.3K	4.7K	3.9K	3.1K

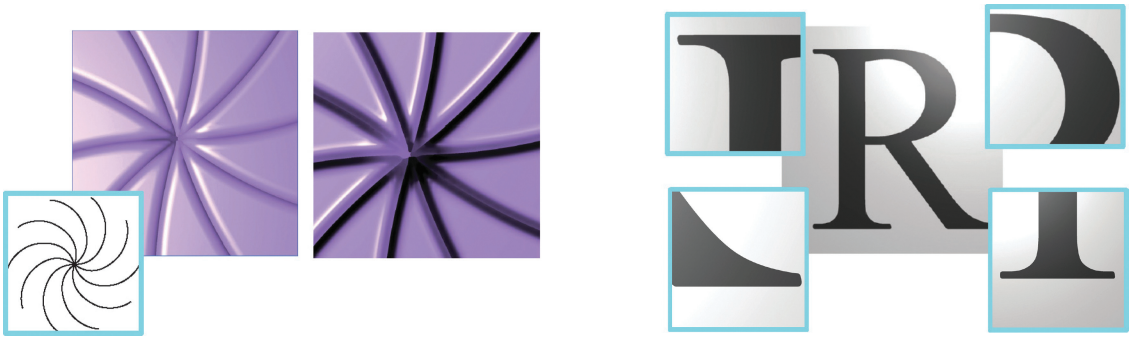

Fig. 17. Left: magnified view (×15) of feature curves constructed from spiral arrangement of Bezier curves (small picture on the left). Right: example of rendering sharp edges. Discontinuity map resolutions: 512 × 512 (left), 80 × 80 (right).

plane fragment runs at 131 fps, and the snake plane fragment runs at 156 fps as opposed to 75 fps and 124 fps reached for the 3D meshes (More evidence on the performance dependence of the linear version of our shader on the mesh size is collected in Table II).

Finally, we demonstrate how our algorithm handles complex star configurations, such as the one shown in Figure 17 (left), on which most of the existing GPU algorithms fail. While artifacts are inevitable by design (our discontinuity texel can not have more than two intersecting curves), we keep them minimal and in the majority of the patterns we guarantee holding C^1 continuity along the feature curves. At the preprocessing step, the central intersection point is split into several new points, which represent simpler intersections of nearby curves in the neighboring texels: Figure 17 (right).

7. CONCLUSIONS AND FUTURE WORK

We have described a technique for representing and real-time rendering of textures with discontinuity feature curves. The main difference of the feature curves from existing types of discontinuities is that the interpolated map, computed by the algorithm, is a function of distance to the discontinuity and its gradient. Separate interpolation of distance and gradient functions were shown to be crucial for obtaining high quality results.

The algorithm uses Bezier curve representation for features directly, and no linear approximation artifacts appear at any resolution.

While only a limited number of local configurations are allowed by the rendering-time representation of the feature curves, we describe a preprocessing step that simplifies a general network to the form that can be used by the rendering algorithm. The preprocessing step is relatively simple, and, for texture with piecewise linear features, can be done interactively.

We demonstrated effectiveness of our method for interpolating normal maps. However, similar approaches can be applied to more advanced types of geometric mapping, such as relief and displacement maps. As discussed briefly in Tumblin and Choudhury [2004], one can remove some of the topological restrictions by considering higher resolution textures, localized to the areas of multiple curve intersections. A variant of the ADF approach [Frisken et al. 2000] can be used to optimize memory consumption of our technique by employing the fact that a majority of texels are not affected by discontinuities. While this approach saves considerable amount of memory, it appears to be less GPU-friendly. Perry and Frisken [2005] describe a cell-based rendering technique for adaptive distance fields. Every cell is associated with geometric primitive which is scan-converted to provide the cell's index to covered pixels. This index is used to lookup cell information similar to our signatures, stored in an array. Using this technique allows to reduce memory consumption at the expense an additional rendering pass rendering a potentially large number of geometric primitives and an indirect texture lookup.

APPENDIX

CODE OUTLINE FOR THE NORMAL INTERPOLATION SHADER

We summarize the part of our fragment shader program which estimates a desired normal $\tilde{\mathbf{n}}(u, v)$ at a given texel's sample $\mathbf{p}_t = [u, v]$ by reconstructing the values $\mathbf{n}(u, v)$, $d(u, v)$, and $\nabla d(u, v)$, and by applying Equation (1) on the reconstructed values. The inputs for the shader are the texture coordinates (u, v) of a current pixel's sample, corners' samples $\mathbf{n}_k$, d_k, and ∇d_k, $k = 1..4$, of the texel $T_{\lfloor v/\Delta V \rfloor, \lfloor u/\Delta U \rfloor}$, and the discontinuity configuration/signature pair (C, S) for that texel. The output is an estimated normal $\tilde{\mathbf{n}}(u, v)$. As an example, Figure 18 shows normal interpolation for a texel with two discontinuity segments inside. The steps are as follows:

1 Find locations of intermediate discontinuity points $\mathbf{q}_a$, $\mathbf{q}_b$, $\mathbf{q}_c$, $\mathbf{q}_d$ from the texel's discontinuity signature S.

2 Fetch the texel's edge indices for end-points of its feature curves from the discontinuity configuration C. Reconstruct the

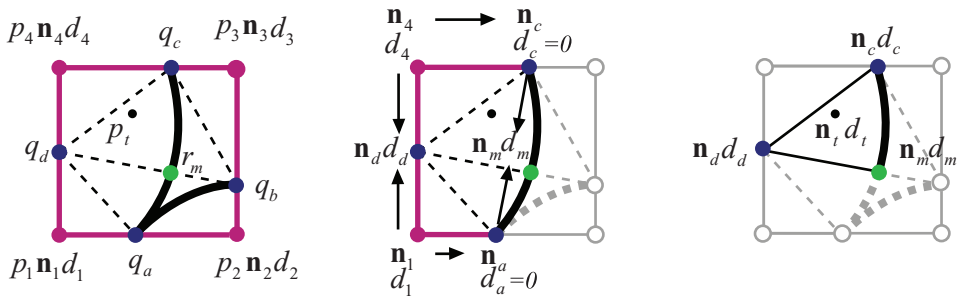

Fig. 18. Computing the normal $\mathbf{n}_t$ and distance d_t at a texel sample $\mathbf{p}_t$. Left: $\{\mathbf{q}_a, \mathbf{q}_b\}$ and $\{\mathbf{q}_a, \mathbf{q}_c\}$ are the discontinuity segments inside the texel; locations of $\mathbf{q}_a$, $\mathbf{q}_b$, $\mathbf{q}_c$, and $\mathbf{r}_m$ are calculated from discontinuity signature; $\mathbf{q}_d$ is in the default position. Center: computing normals and distances at $\mathbf{q}_a$, $\mathbf{q}_c$, $\mathbf{q}_d$, and $\mathbf{r}_m$. Right: 1C-triangle $\langle \mathbf{q}_c, \mathbf{q}_d, \mathbf{r}_m \rangle$ covers $\mathbf{p}_t$; values at the triangle's corners are used to compute $\mathbf{n}_t$ and d_t.

Bernstein forms of the curves—$\mathbf{B}_1(s)$, $\mathbf{B}_2(s)$—by calculating their middle control points $\mathbf{b}_1^1$ and $\mathbf{b}_1^2$ as the intersection points of the rays going from the points $\mathbf{q}_*$ and into the directions given by the discontinuity signature S. Set $\mathbf{r}_m$ as an intersection of (possibly curvilinear) segments $\{\mathbf{q}_a, \mathbf{q}_c\}$ and $\{\mathbf{q}_b, \mathbf{q}_d\}$.

3 Reconstruct normals, distances, and gradients at the active discontinuity points—the points from $\mathbf{q}_a \ldots \mathbf{q}_c$, which are on the same side of discontinuity segments as $\mathbf{p}_t$. Use the side test equations from (Section 5.2).
  3.1 Depending on the number of corners reachable from a given active point, choose one of the following three cases to reconstruct the normal[1]:
    3.1.1 two corners: interpolate the normal from the values at these corners;
    3.1.2 one corner: copy the normal from the corner;
    3.1.3 none (such an active point is a common end-point of the discontinuity segments with the sample point located between the segments): use the normal from the opposite end-point of either segment.
  3.2 Set the resulting distance to zero and assign a predefined value for the z-component of the resulting normal at every active point which is located on the discontinuity.

4 Determine the triangle containing point $\mathbf{p}_t$ by running the side test with respect to curvilinear edges of the triangles from the texel's partition; if needed, reconstruct the normal and distance at $\mathbf{r}_m$, interpolating between the values at the endpoints of the discontinuity with known values.

5 Depending on the type of the resulting curvilinear triangle, compute the interpolated normal $\mathbf{n}(u, v)$, the distance d_t, and its gradient ∇d_t at $\mathbf{p}_t$ from known samples at the triangle corners, using the corresponding scheme from (Section 5.3); apply Equation 1 to the resulting values to calculate the final blended normal $\tilde{\mathbf{n}}(u, v)$.

REFERENCES

BALA, K., WALTER, B. J., AND GREENBERG, D. P. 2003. Combining edges and points for interactive high-quality rendering. *ACM Trans. Graph. 22,* 3 (July), 631–640.

BECKER, B. G. AND MAX, N. L. 1993. Smooth transitions between bump rendering algorithms. In *Proceedings of SIGGRAPH*. Computer Graphics Proceedings, Annual Conference Series. 183–190.

BLINN, J. F. 1978. Simulation of wrinkled surfaces. In *Proceedings of SIGGRAPH*. Vol. 12. 286–292.

COOK, R. L. 1984. Shade trees. In *Proceedings of SIGGRAPH*. Vol. 18. 223–231.

COX, D., LITTLE, J., AND O'SHEA, D. 1998. Using algebraic geometry. *Graduate Texts in Mathematics*. Springer-Verlag, 78–82.

DONELLY, W. 2005. Per-pixel displacement mapping with distance functions. In *GPU Gems 2: Programming Techniques for High- Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, Chapter 8.

FARIN, G. 1997. *Curves and Surfaces for Computer-Aided Geometric Design*, 4th Ed. Academic Press. 4.

FRISKEN, S. F. AND PERRY, R. N. 2006. Method for generating an adaptively sampled distance field of an object with specialized cells. US Patent 7,042,458.

FRISKEN, S. F., PERRY, R. N., AND JONES, T. R. 2002. Detail-directed hierarchical distance fields. US Patent 6,396,492.

FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'00)*. ACM Press. New York, NY, 249–254.

GREEN, S. 2005. The opengl framebuffer object exntension. In *Proceedings of the Game Developers Conference*. San Francisco, CA.

HECKBERT, P. 1992. Discontinuity meshing for radiosity. In *Proceedings of the 3rd Eurographics Workshop on Rendering*. 203–226.

HEIDRICH, W., DAUBERT, K., KAUTZ, J., AND SEIDEL, H.-P. 2000. Illuminating micro geometry based on precomputed visibility. In *Proceedings of ACM SIGGRAPH*. Computer Graphics Proceedings, Annual Conference Series. 455–464.

ISMERT, R. M., BALA, K., AND GREENBERG, D. P. 2003. Detail synthesis for image-based texturing. In *Proceedings of the ACM Symposium on Interactive 3D Graphics*. 171–175.

LOOP, C. AND BLINN, J. 2005. Resolution independent curve rendering using programmable graphics hardware. *ACM Trans. Graph. 24,* 3, 1000–1009.

MAX, N. L. 1986. Shadows for bump-mapped surfaces. In *Proceedings of Advanced Computer Graphics*. 145–156.

MAX, N. L. 1988. Horizon mapping: shadows for bump-mapped surfaces. *Visual Comput. 4,* 2 (July), 109–117.

MCGUIRE, M. AND MCGUIRE, M. 2005. Steep parallax mapping. Poster, Symposium on Interactive 3D Graphics and Curves.

[1] the same rules are applied for reconstructing the distance and its gradient.

NISHITA, T., SEDERBERG, T. W., AND KAKIMOTO, M. 1990. Ray tracing trimmed rational surface patches. *SIGGRAPH Comput. Graph. 24,* 4, 337–345.

OLIVEIRA, M. M., BISHOP, G., AND MCALLISTER, D. 2000. Relief texture mapping. In *Proceedings of ACM SIGGRAPH*. Computer Graphics Proceedings, Annual Conference Series. 359–368.

PERLIN, K. 1985. An image synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. ACM Press, New York, NY, 287–296.

PERRY, R. N. AND FRISKEN, S. F. 2005. Method and apparatus for rendering cell-based distance fields using texture mapping. US Patent 6,917,369.

POLICARPO, F., OLIVEIRA, M. M., AND AO L. D. COMBA, J. 2005. Real-time relief mapping on arbitrary polygonal surfaces. In *Proceedings of the Symposium on Interactive 3D Graphics and Games (SI3D'05)*. ACM Press, New York, NY, 155–162.

QIN, Z., MCCOOL, M. D., AND KAPLAN, C. S. 2006. Real-time texture-mapped vector glyphs. In *Proceedings of the Symposium on Interactive 3D Graphics and Games (SI3D'06)*. ACM Press, New York, NY, 125–132.

RAMANARAYANAN, G., BALA, K., AND WALTER, B. 2004. Feature-based textures. In *15th Eurographics Workshop on Rendering*. 265–274.

SALISBURY, M., ANDERSON, C., LISCHINSKI, D., AND SALESIN, D. H. 1996. Scale-dependent reproduction of pen-and-ink illustrations. In *Proceedings of SIGGRAPH*. Computer Graphics Proceedings, Annual Conference Series. 461–468.

SEN, P. 2004. Silhouette maps for improved texture magnification. In *Graphics Hardware*. 65–74.

SEN, P., CAMMARANO, M., AND HANRAHAN, P. 2003. Shadow silhouette maps. *ACM Trans. Graph. 22,* 3 (July), 521–526.

SLOAN, P. AND COHEN, M. F. 2000. Hardware accelerated horizon mapping. In *11th Eurographics Workshop on Rendering*. 281–286.

TARINI, M. AND CIGNONI, P. 2005. Pinchmaps: textures with customizable discontinuities. *Comput. Graph. For. 24,* 3. To appear.

TARINI, M., CIGNONI, P., ROCCHINI, C., AND SCOPIGNO, R. 2000. Real time, accurate, multi-featured rendering of bump mapped surfaces. *Comput. Graph. For. 19,* 3 (Aug.).

TAUBIN, G. 1995. A signal procesesing approach to fair surface design. In *SIGGRAPH Conference Proceedings*, R. Cook, Ed. Annual Conference Series. ACM SIGGRAPH, Addison Wesley, 351–358.

TUMBLIN, J. AND CHOUDHURY, P. 2004. Bixels: Picture samples with sharp embedded boundaries. In *Proceedings of the 15th Eurographics Workshop on Rendering Techniques*.

WANG, L., WANG, X., TONG, X., LIN, S., HU, S., GUO, B., AND SHUM, H.-Y. 2003. View-dependent displacement mapping. *ACM Trans. Graph. 22,* 3 (July), 334–339.

WANG, X., TONG, X., LIN, S., HU, S., GUO, B., AND SHUM, H.-Y. 2004. Generalized displacement maps. In *15th Eurographics Workshop on Rendering*. 227–234.

ZHANG, J., ZHOU, K., VELHO, L., GUO, B., AND SHUM, H.-Y. 2003. Synthesis of progressively variant textures on arbitrary surfaces. *ACM Trans. Graph. 22,* 3 (July), 295–302.

Received June 2006; revised April 2007, September 2007; accepted October 2007