Evgueni Parilov · Ilya Rosenberg · Denis Zorin

Real-time rendering of normal maps with discontinuities

CIMS Technical Report, TR2005-872

August, 2005

Abstract Normal mapping uses normal perturbations stored in a texture to give objects a more geometrically complex appearance without increasing the number of geometric primitives. Standard bi- and trilinear interpolation of normal maps works well if the normal field is continuous, but may result in visible artifacts in the areas where the field is discontinuous, which is common for surfaces with creases and dents.

In this paper we describe a real-time rendering technique which preserves the discontinuity curves of the normal field at sub-pixel level and its GPU implementation. Our representation of the piecewise-continuous normal field is based on approximations of the distance function to the discontinuity set and its gradient. Using these approximations we can efficiently reconstruct discontinuities at arbitrary resolution and ensure that no normals are interpolated across the discontinuity. We also described a method for updating the normal field along the discontinuities in real-time based on blending the original field with the one calculated from a user-defined surface profile.

Keywords Interactive rendering · real-time normal mapping · discontinuous normal field · distance function approximation · constrained Laplacian smoothing

1 Introduction

Bump and normal mapping is a widely used technique for adding complex small-scale geometric detail to models, in which only the normals are modified using perturbations stored in a texture. Bump mapping and related techniques were successfully applied in combination with a variety of rendering algorithms, and thanks to the advances in graphics hardware its use is becoming routine in interactive applications.

The regular sampling pattern used in normal maps leads to a spectrum of aliasing problems, in particular, for sharp linear features. Such features are commonplace in fine scale geometry (gratings, creases, scratches, cracks, seams etc). Two main types of artifacts may be observed: 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). The latter artifact is much more severe compared to color textures, which are blurred near discontinuities.

Recent feature-based texture representations [16,9,19] improve texture appearance by explicitly representing discontinuities. This approach eliminates or reduces many types of artifacts.

In this paper we describe a technique extending these approaches to a novel type of normal map designed for accurate interactive rendering of normal maps with sharp features. The most important feature we add to the texture representation used in these approaches is a pair of additional maps, defining the distance function

Evgueni Parilov
New York University
719 Broadway, 12th floor
New York, NY 10003 E-mail: parilov@mrl.nyu.edu

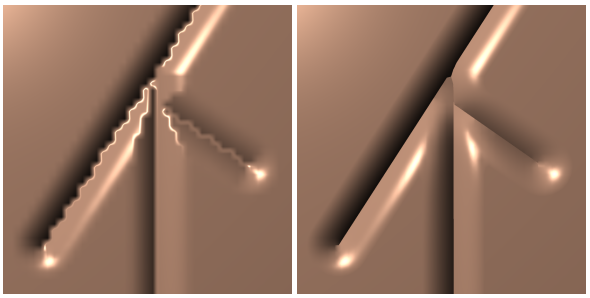

Fig. 1 Left: Typical bump and normal map artifacts. Note the bright lines at the bottom of the crease: these lines are due to the interpolation between normals pointing in opposite directions. Right: same map enhanced with rendered using our technique. The resolution of the map in both cases is the same; the visible part is 40x40 pixels.

to the discontinuity set and it gradient, and a reconstruction algorithm for the distance map and its which ensures that its zero set exactly coincides with the discontinuity set of the gradient.

The input to our algorithm is a collection of discontinuities represented by edges and a normal map. 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 the function of the distance to the feature. By using user-controlled profiles we can produce a variety of behaviors without dramatic increase in texture size.

We present algorithms for two rendering stages: preprocessing and normal computation performed at rendering time. We use two preprocessing algorithms: the algorithm for converting an arbitrary collection of edges to a simplified representation suitable for hardware rendering (Section 4.1), and an algorithm for defining a smooth distance function which is exact in the immediate neighborhood of features (Section 4.2).

The most important part of the rendering algorithm is the normal computation which ensures that the normals have prescribed behavior near discontinuity lines at arbitrary resolution and follow the normal map away from the features (Section 4.3).

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 appeared in computer graphics in many different contexts (e.g. discontinuity meshing for radiosity [6] to nonphotorealistic rendering). We focus only on the most closely related work. Salisbury et al [17] presented an illustration reproduction approach which keeps discontinuous regions sharp at any scale. The reconstruction algorithm starts from arbitrary interpolation kernel and re-weights its intensity values according to closeness to the sample. As the algorithm is based on locating shortest-paths it is difficult to adapt it for interactive applications.

Our method is based on the work on feature-based textures [16], bixels [9] and silhouette maps [19]. Bala and co-workers [1, 16] 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. Efficient hardware-accelerated implementation of feature-based textures requires conversion of input curve networks to simplified form.

Tumblin and Choudhury [9] encode discontinuities using *bixels*, i.e. pixels with additional annotation, which defines texture discontinuity segments for every pixel, allowing only a restricted set of combinations of segments. Our discontinuity maps are similar in spirit, but a different set of allowable segments is used. This method can handle non-closed boundaries. Also, due to its relative complexity, it is not well-suited for graphics hardware implementation at this time.

Sen's silhouette maps [19], extending [20], are similar to feature-based textures and bixels, but the interpolation approach used in this work is further simplified so that it maps well to graphics hardware. As in [9], a finite number of discontinuity boundary configurations are used for each pixel.

These methods cannot be directly applied to normal maps, as they do not provide a sufficiently flexible way to control the behavior of the the normal near discontinuities.

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 [3]. Many techniques for interactive rendering of bump maps were proposed, see e.g. [22] 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 that was introduced in [10, 11]; [21] described how horizon maps can be accelerated using hardware. Parallax mapping and steep parallax mapping [12] aims 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.)

In many methods for high-quality rendering of normal maps one needs to make a transition between different rendering modes, [2]; enhancements to lighting of bump maps necessary for such transitions are described in [7]. Displacement maps introduced in [4] and relief maps [13] are a more advanced form of representing fine-scale geometry for flat surfaces (extensions to arbitrary surfaces were presented in [15]).

While techniques for interactive rendering of displacement maps were recently proposed [24], [25] 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. Our work is related to work in texture synthesis, as our algorithm can be viewed as a combination of rendering sampled and procedurally defined normals. Procedural bump maps first appeared in [14]. There were a large number of papers on non-parametric synthesis from examples; these techniques often can be applied to produce bump maps, e.g. see recent paper [26]. Our technique addresses a specific case of the problem of adding high-resolution detail to an image, focusing on sharp linear features; a general approach to this problem can be found in [8].

One of the important features of our approach is reliance on distance fields, most often used in geometric modeling (e.g. [5]).

3 Overview

As the input to the rendering process we assume a normal map $\mathbf{B}(u,v) = [B_x, B_y, B_z] = [\mathbf{B}_h, B_z]$, where $\mathbf{B}_h$ is the projection of the normal to the plane, representing the normals for the smooth part of the surface, a collection of discontinuity curves and, in the simplest case, a user-specified profile for creases $p(d)$. The profile specifies the cross-section of the surface in the direction perpendicular to the crease for a small distance d, and may depend on the position along the discontinuity line and other parameters.

We also use a blending function $b(d)$, where d is the distance to discontinuity, to join the interpolated normal map with the crease profile. The computation of a new normal $\mathbf{n}'(d) = (1-b(d))\,\mathbf{n}_p(d) + b(d)\mathbf{B}(d)$ is illustrated on Figure 2, where $\mathbf{n}_p$ is a normal of crease profile pointing toward discontinuity curve. We choose $b(d)$ to satisfy $b(d) = 0$ for $d < w_0$ and $b(d) = 1$ for $d > w$, where w_0 is the half-width of a band around the discontinuity for which the original normal map has no effect (which can be zero).

Let $d(u,v)$ be the distance from a point (u,v) to the discontinuity set; then the (unnormalized) desired continuous normal map $\mathbf{n}'(u,v)$ is

$$\begin{aligned}\mathbf{n}'(u,v) = \mathbf{P}_0 \Big(-(1-b(d))p'(d)\nabla d + \tilde{b}(d)\mathbf{B}_h(u,v)\Big) + \\ \Big((1-b(d)) + \tilde{b}(d)B(u,v)_z\Big)\,\mathbf{e}_z\end{aligned} \tag{1}$$

where $\mathbf{P}_0\left([x,y]\right) = [x,y,0]$, $\tilde{b}(d) = b(d)\sqrt{p'(d)^2 \cdot \nabla^2 d + 1}$, and $\mathbf{e}_z$ is the unit vector along the z-axis in texture space. (In the formulas, we have omitted the dependence of d on (u,v).)

This is the continuous function our algorithm approximates. The local profile can be one-sided and depend on the distance along a curve.

4 Algorithms

Next, we describe two parts of the preprocessing stage and the interpolation algorithm itself. The goal of the preprocessing stage is to define maps used by the interactive algorithm: a discontinuity map, the distance map

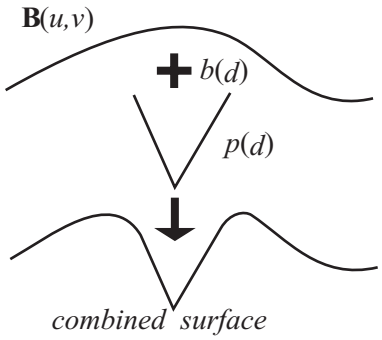

Fig. 2 Interpolation between the local profile and global smooth map. We show height maps for clarity, but in general interpolating height maps is not equivalent to interpolating normals.

and the distance gradient map. All maps are used in close coordination to perform accurate normal evaluation near discontinuities.

4.1 Simplifying discontinuity maps

The input of this stage is a collection of line segments representing discontinuities, either manually specified or obtained as an output of a tracing program (e.g. [18]). Local configurations of discontinuities can be arbitrarily complex: e.g. any number of segments can meet at a point. Following [9] and [19] we reduce the number of allowed configurations, although we choose a different set, to avoid introduction of points in the center of pixels as is done in this work.

Allowable discontinuity configurations for a single pixel are defined by the following two rules:

- a texel's boundary edge may be passed by discontinuity segments at no more than one point;
- there are no more than two discontinuity segments inside any texel;
- a discontinuity segment ends on a texel boundary.

Several valid configurations are shown in Figure 3.

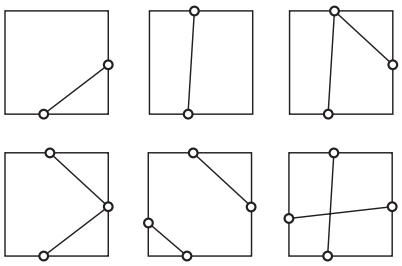

Fig. 3 Allowable discontinuity configurations for a texel.

The algorithm modifies discontinuity segments, to transform an arbitrary configuration into the one that satisfies our requirements. Figure 4 illustrates the changes in the discontinuity map the algorithm does.

Once the transformation is complete we use a simple encoding for the resulting segment network: each texel is annotated with four numbers representing the distances to discontinuity segment endpoints along the texel boundary, which we call its discontinuity signature (Figure 5). If there are no segments, the quadruple of numbers associated with the pixel is $(0, 0, 0, 0)$. If there is one discontinuity segment in the pixel it is $(d_{11}, d_{12}, 0, 0)$, and if there are two, it is $(d_{11}, d_{12}, d_{21}, d_{22})$.

The conversion process proceeds as follows. Each linear segment is scan-converted to the texture to determine all texels that it intersects (visited texels). For each visited texel, we check if there are one or two

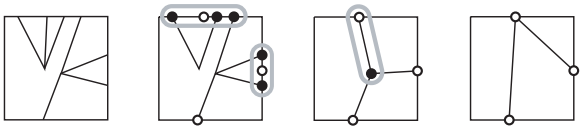

Fig. 4 Edge point transformations. Black points are replaced by their center of mass (white points) to satisfy the first rule, identified discontinuity segments are deleted to follow the second rule.

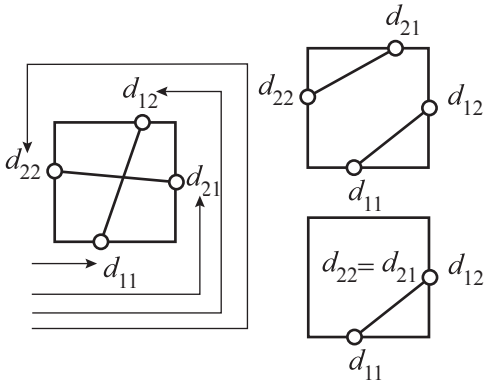

Fig. 5 Discontinuity signature: an encoding of the endpoints of discontinuity segments for a texel.

intersections with its edges. In the former case, only the intersection point is used in the conversion process, while the linear part of the segment is disregarded. Then, for the latter case, there is a table of texel transformations, which replaces a current configuration to an adjusted one, depending on intersection points of current segment. 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 included so far discontinuity points per every edge.

4.2 Distance functions

The second stage of preprocessing is to compute a distance map for the discontinuity segments. The distance map plays a dual role: it determines the location of the discontinuity segments, and the distance to these segments. The direction of the gradient of the distance function for many feature profiles heavily influences the normal, as can be seen from Equation (1).

Choosing the right way to approximate the distance function is crucial for maintaining rendering quality. We would like it to satisfy the following conditions:

- the approximate distance function should be exactly zero at feature lines so that sharp features can be created; it should stay close to the real distance function near feature lines to be able to control feature width correctly and it should vary continuously;
- the gradient of the distance function should be continuous away from feature lines, and vary smoothly as it is used directly in the normal calculation.

To understand the difficulties with approximating the distance function on the grid, we observe that Euclidean distance function have two types of singularities where the gradient is discontinuous: distance zero curves, and medial axis curves, i.e. sets of points which are equidistant from two or more points on feature lines (Figure 6).

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

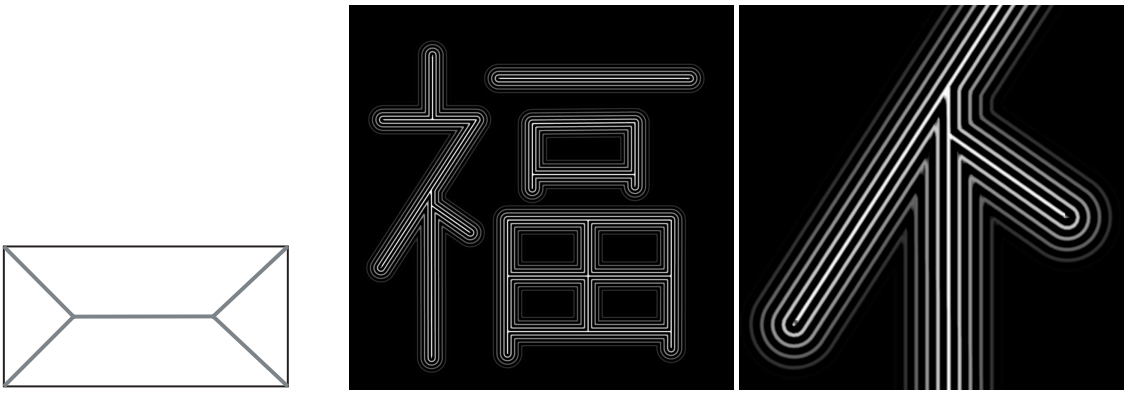

Fig. 6 Left: Distance function singularities for a rectangle: the medial axis is depicted in gray; it intersects the rectangle at vertices. Right: Distance function level lines and close-up.

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

1. the distance function is linearly interpolated on triangular subtexel domains (Figure 8) aligned with feature lines to ensure that it is zero at these lines;
2. the gradient 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 are smoothed away from the feature lines, to eliminate discontinuities at the medial axis.

We consider interpolation, which happens at rendering time, in the next section. Here we only discuss smoothing of the gradient field. We smooth the sampled distance gradient field using constrained Laplacian smoothing (see e.g. [23] for related techniques). Specifically, each vertex moves toward the barycenter of its neighbors, excluding the neighbors on the other side of discontinuity. We found that approximately 10 smoothing iterations is sufficient. As is well known, Laplacian smoothing quickly eliminates high frequency features, while reducing low frequency features slowly, which is the desired result in our case.

Note that no smoothing is done on the distance function itself; we found it important to maintain its shape as close to the original as possible for linear features.

The brute-force computation of the distance function can be expensive, but it can be accelerated. We use the following simple approach. For each linear segment we find the distances to all grid points located in a rectangular subgrid which completely covers the segment, and record the values at grid points if they are less than previously recorded values. Clearly, there are more sophisticated acceleration algorithms using graphics hardware in particular, but as we use this only as a precomputation this algorithm is not time critical.

The gradient of the distance function at grid points is computed as a unit vector in the direction of the closest point on the nearest discontinuity segment.

4.3 Normal computation

The input to the interactive algorithm includes

- the distance function map d and the gradient map g;
- the discontinuity map, i.e. the four distances to discontinuity segment endpoints;
- any other texture maps used for rendering, including the normal map;
- user-defined crease profile $p(d)$, where d is the distance to the crease in texel units, specified by the user. It is stored in a one-dimensional texture.

For a point $x = (u, v)$ in a texel, the normal is computed at this point using only samples at the corners of the texel and the four discontinuity map values associated with the texel. The normal at a point x is calculated using Equation (1).

The principal ingredient of this calculation are interpolated values for the distance function $d(u, v))$ and distance gradient $\nabla d(u, v)$ at x. The behavior of these two functions is different and different interpolation techniques can be used.

Distance function interpolation. If there are no discontinuity segments in the pixel, then standard bilinear interpolation is used to compute the distance function value and its gradient at the point.

If there is one or two discontinuity segments, the distance function is required to be exactly zero on the discontinuity line. As our initial discontinuity map is simplified to have a limited number of local configurations, there is an 8-triangle partition of a texel containing all possible discontinuity segments as edges of the triangles (Figure 7).

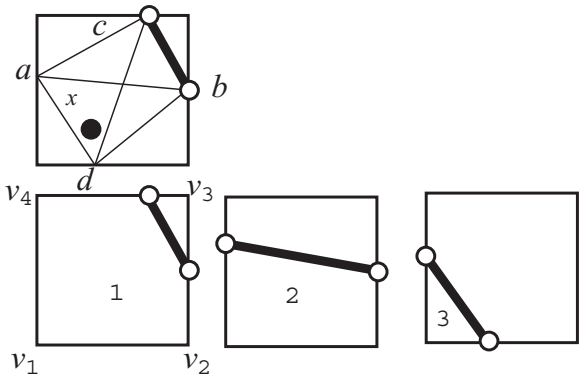

Fig. 7 Top: the distance function is interpolated on one of 8 subtriangles. Bottom: gradient interpolation cases.

We identify the triangle which contains x, and calculate the distance at x as an interpolation between distance values at the triangle's corners by using barycentric coordinates of x. To calculate the distance function at the triangle corners, we interpolate the distances given at the texel's corners. We set it to zero at discontinuity segment endpoints (as at points c and b on the top of Figure 7), while the values at points on the edges of the texel, which are not on a discontinuity line, are obtained by linear interpolation between adjacent corner points of the texel (as at points a and d).

Only the corner in the center needs special treatment. For efficiency, we choose it to be the average between linear interpolants of values at points a and b, and c and d (Figure 7).

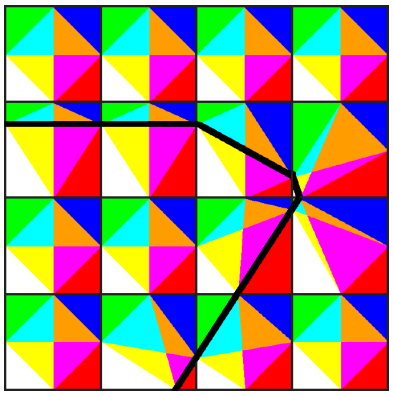

Fig. 8 Interpolation domains for the distance function near a feature line shown in different colors.

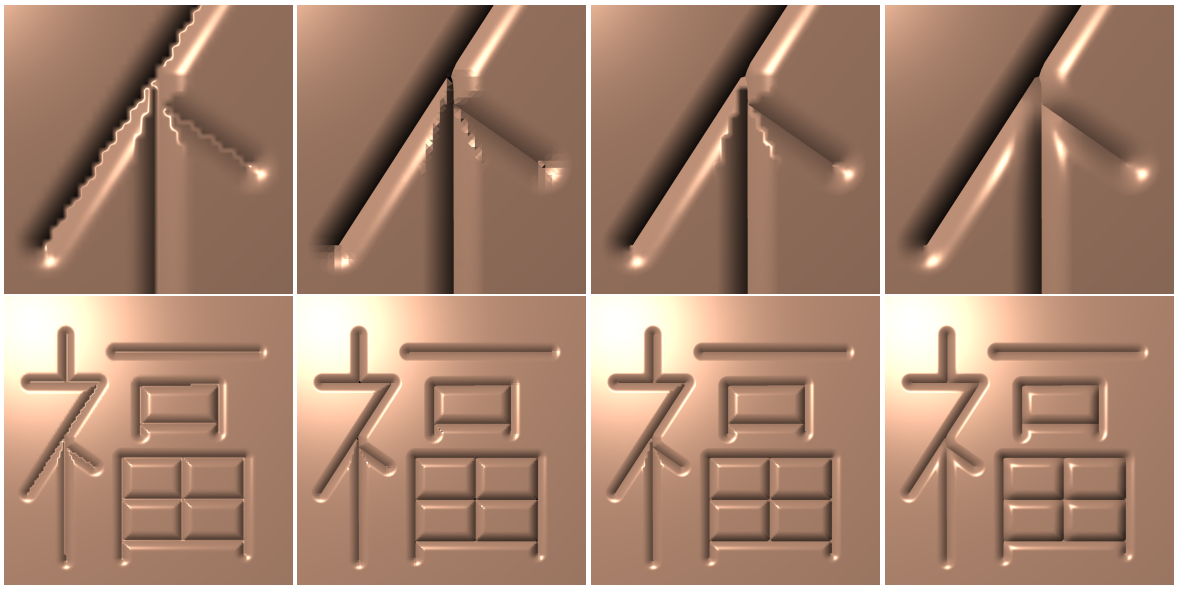

Fig. 9 The appearance of features for different interpolation methods, from left to right: standard bilinear interpolation of samples representing the profile; piecewise linear distance function and piecewise constant gradient computed directly from the distance function; piecewise linear interpolation for gradient and distance function, without constrained smoothing of the gradient field; same with smoothing.

Gradient interpolation. Our procedure here is similar to [19]. The gradient of the distance function is discontinuous at the interface, so we linearly interpolate from the available gradients on the same side of all discontinuities at the point we evaluate, which can vary from one to three gradients (bottom of Figure 7):

$$\begin{aligned} g(x) &= (1-u-v)g(v_1) + ug(v_2) + vg(v_4) \quad \text{for case 1} \\ g(x) &= (1-u)g(v_1) + ug(v_2) \quad \text{for case 2} \\ g(x) &= g(v_1) \quad \text{for case 3} \end{aligned} \tag{2}$$

Gradient magnitudes after interpolation are forced to be one for consistency with the exact distance function. However, note that the directions of gradients vary smoothly inside and across the edges of the texels.

5 Results

We have implemented the interactive interpolation algorithm as a vertex shader in `Cg` and tested the implementation on NVIDIA GeForce 6800, running on Pentium 4, 2.8MHz computer. The frame rates that we have obtained for 512x512 image were in the range from 40 to 200 frames per second, depending on the complexity of lighting.

We use an extra RGBA float texture to encode texel discontinuity signatures: the first and the fourth quadruple items are stored in ”RG” components (while the second and the third items are located in the corresponding ”RG” components of the east and the north texel neighbors), the indices of texel edges which are passed by discontinuity segments (encoded by at most four two-bit numbers) are held in ”B” component. Distance values are stored in the unused ”A” component of the normal map texture.

Figure 9 compares different modes of rendering normal maps with sharp features, enabling different parts of our approach one-by-one.

Figure 10 shows several examples of user-specified profiles defined using our method. All textures are 128x128 (one of the examples shows texel sizes). The close-up views show that our technique results in few artifacts even in complex situations.

Figure 11 shows several screenshots of feature-based normal maps created using our technique. The original real-time demos may be located at `www.mrl.nyu.edu/~parilov/demos/techreps/normmaps`.

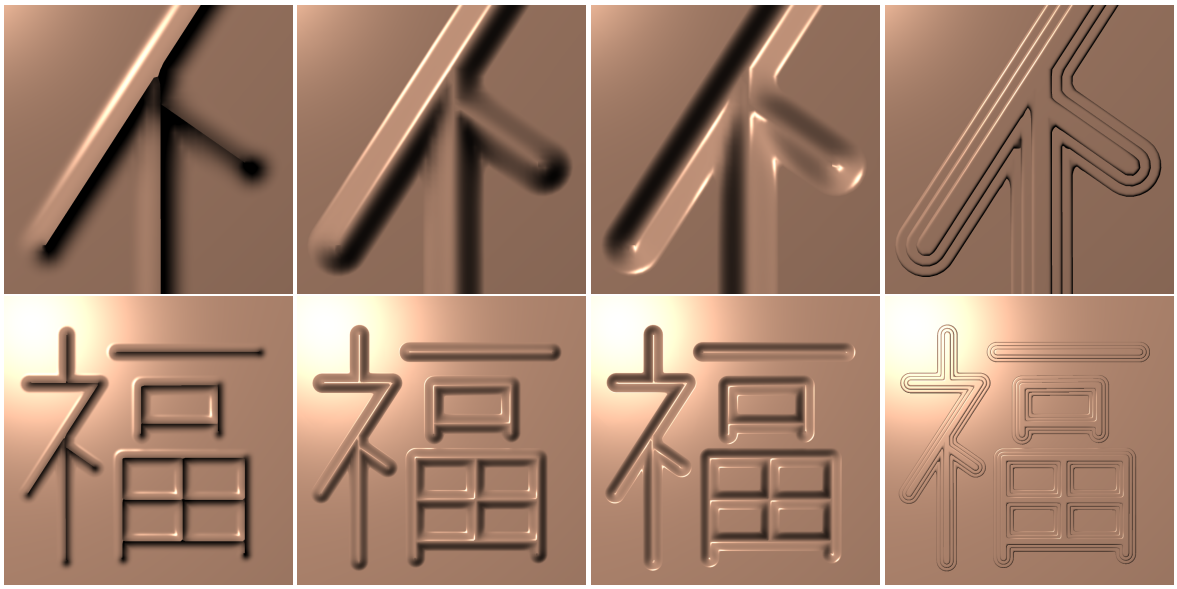

Fig. 10 Examples of user-defined profiles. The discontinuity map is the same in all cases; note that in some cases, the discontinuity is removed entirely: a spectrum of creases of variable sharpness is possible.

6 Conclusions and Future Work

We have demonstrated a simple and efficient technique for rendering feature-based normal maps. Our approach combines procedural and image-based techniques to enable high-quality interactive rendering of sharp small-scale features on surfaces. The preprocessing required by our approach is relatively simple and for textures of moderate size can be done interactively. The main downside of these type of techniques is the constraints imposed on local configurations of feature lines. Furthermore, this technique works best for piecewise linear features. While curves can be easily approximated substantially higher resolution would be required.

Similar approaches can be applied to more advanced types of geometric mapping, such as relief and displacement maps. Further advances in the speed of graphics processors may make it possible to use low-order polynomials instead of straight lines for accurate curve representation. As is discussed briefly in [9], it is possible to remove some of the topological restrictions by considering higher resolution textures localized to the areas of multiple curve intersections.

Acknowledgements We would like to thank Jeff Han for his valuable comments on GPU implementation of normal maps.

References

1. Bala, K., Walter, B.J., Greenberg, D.P.: Combining edges and points for interactive high-quality rendering. ACM Transactions on Graphics **22**(3), 631–640 (2003)
2. Becker, B.G., Max, N.L.: Smooth transitions between bump rendering algorithms. In: Proceedings of SIGGRAPH 93, Computer Graphics Proceedings, Annual Conference Series, pp. 183–190 (1993)
3. Blinn, J.F.: Simulation of wrinkled surfaces. In: Computer Graphics (Proceedings of SIGGRAPH 78), vol. 12, pp. 286–292 (1978)
4. Cook, R.L.: Shade trees. In: Computer Graphics (Proceedings of SIGGRAPH 84), vol. 18, pp. 223–231 (1984)
5. Frisken, S.F., Perry, R.N., Rockwood, A.P., Jones, T.R.: Adaptively sampled distance fields: a general representation of shape for computer graphics. In: SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 249–254. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (2000). DOI http://doi.acm.org/10.1145/344779.344899
6. Heckbert, P.: Discontinuity meshing for radiosity. In: 3rd Eurographics Workshop on Rendering, pp. 203–226 (1992)
7. Heidrich, W., Daubert, K., Kautz, J., Seidel, H.P.: Illuminating micro geometry based on precomputed visibility. In: Proceedings of ACM SIGGRAPH 2000, Computer Graphics Proceedings, Annual Conference Series, pp. 455–464 (2000)
8. Ismert, R.M., Bala, K., Greenberg, D.P.: Detail synthesis for image-based texturing. In: 2003 ACM Symposium on Interactive 3D Graphics, pp. 171–175 (2003)

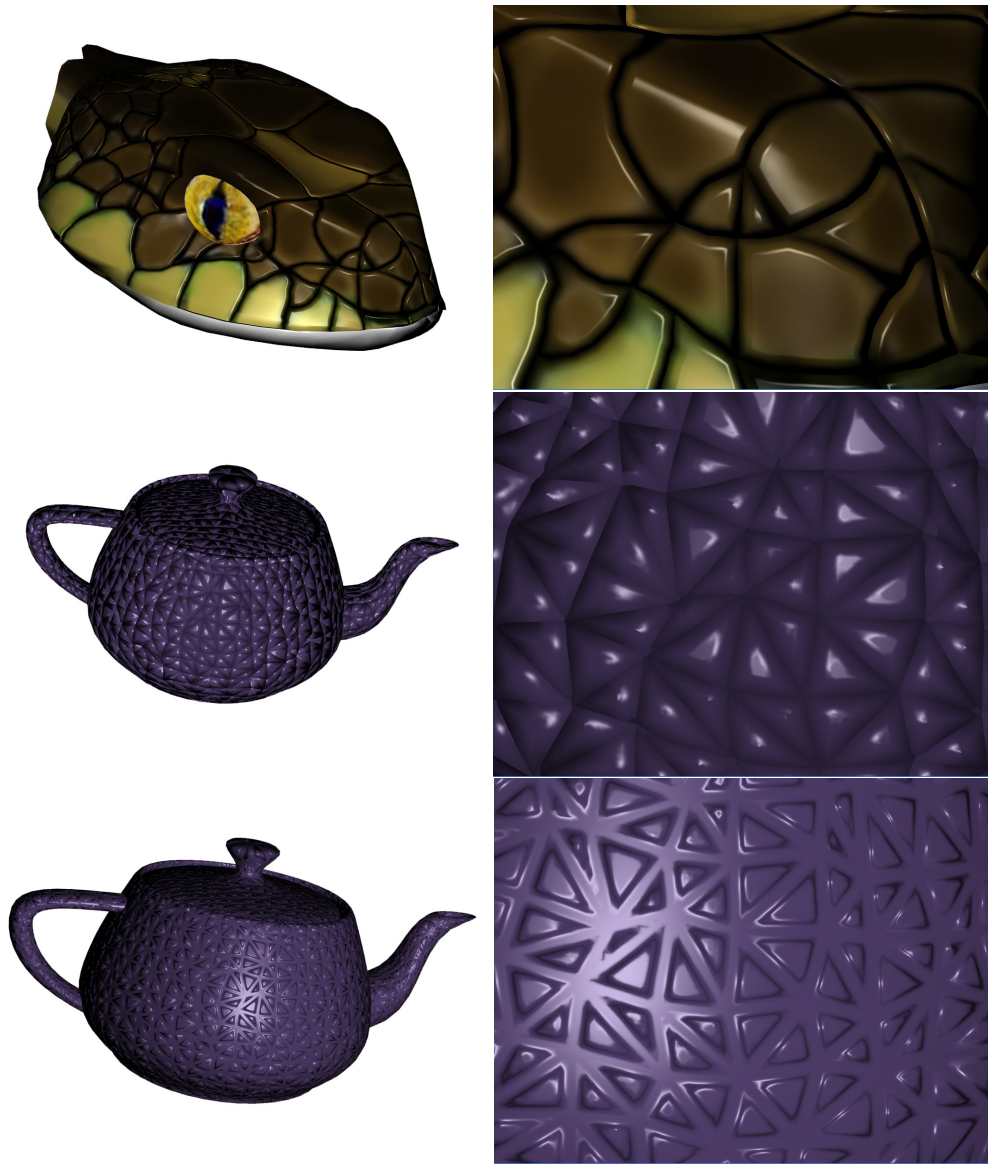

Fig. 11 Several examples of models rendered using our technique.

9. Jack Tumblin, P.C.: Bixels: Picture samples with sharp embedded boundaries. In: Proceedings of the 15th Eurographics Workshop on Rendering Techniques (2004)
10. Max, N.L.: Shadows for bump-mapped surfaces. In: Advanced Computer Graphics (Proceedings of Computer Graphics Tokyo '86), pp. 145–156 (1986)
11. Max, N.L.: Horizon mapping: shadows for bump-mapped surfaces. The Visual Computer **4**(2), 109–117 (1988)
12. McGuire, M., McGuire, M.: Steep parallax mapping (2005)
13. Oliveira, M.M., Bishop, G., McAllister, D.: Relief texture mapping. In: Proceedings of ACM SIGGRAPH 2000, Computer Graphics Proceedings, Annual Conference Series, pp. 359–368 (2000)
14. Perlin, K.: An image synthesizer. In: SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques, pp. 287–296. ACM Press, New York, NY, USA (1985). DOI http://doi.acm.org/10.1145/325334.325247
15. Policarpo, F., Oliveira, M.M., ao L. D. Comba, J.: Real-time relief mapping on arbitrary polygonal surfaces. In: SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games, pp. 155–162. ACM Press, New York, NY, USA (2005). DOI http://doi.acm.org/10.1145/1053427.1053453
16. Ramanarayanan, G., Bala, K., Walter, B.: Feature-based textures. In: Rendering Techniques 2004: 15th Eurographics Workshop on Rendering, pp. 265–274 (2004)
17. Salisbury, M., Anderson, C., Lischinski, D., Salesin, D.H.: Scale-dependent reproduction of pen-and-ink illustrations. In: Proceedings of SIGGRAPH 96, Computer Graphics Proceedings, Annual Conference Series, pp. 461–468 (1996)

18. Selinger, P.: Potrace: a polygon-based tracing algorithm. http://potrace.sourceforge.net/potrace.pdf
19. Sen, P.: Silhouette maps for improved texture magnification. In: Graphics Hardware 2004, pp. 65–74 (2004)
20. Sen, P., Cammarano, M., Hanrahan, P.: Shadow silhouette maps. ACM Transactions on Graphics **22**(3), 521–526 (2003)
21. Sloan, P., Cohen, M.F.: Hardware accelerated horizon mapping. In: Rendering Techniques 2000: 11th Eurographics Workshop on Rendering, pp. 281–286 (2000)
22. Tarini, M., Cignoni, P., Rocchini, C., Scopigno, R.: Real time, accurate, multi-featured rendering of bump mapped surfaces. Computer Graphics Forum **19**(3) (2000)
23. Taubin, G.: A signal processing approach to fair surface design. In: R. Cook (ed.) SIGGRAPH 95 Conference Proceedings, Annual Conference Series, pp. 351–358. ACM SIGGRAPH, Addison Wesley (1995)
24. Wang, L., Wang, X., Tong, X., Lin, S., Hu, S., Guo, B., Shum, H.Y.: View-dependent displacement mapping. ACM Transactions on Graphics **22**(3), 334–339 (2003)
25. Wang, X., Tong, X., Lin, S., Hu, S., Guo, B., Shum, H.Y.: Generalized displacement maps. In: Rendering Techniques 2004: 15th Eurographics Workshop on Rendering, pp. 227–234 (2004)
26. Zhang, J., Zhou, K., Velho, L., Guo, B., Shum, H.Y.: Synthesis of progressively variant textures on arbitrary surfaces. ACM Transactions on Graphics **22**(3), 295–302 (2003)

A Fragment Shader program

The pseudo-code below describes the part of our fragment shader program which estimates a desired normal $\mathbf{n}'(p_x)$ at a given pixel sample p_x by applying Equation (1) on the reconstructed values $\mathbf{B}(p_x)$, $d(p_x)$, and $\nabla d(p_x)$. The input for the program are the (u, v) coordinates of p_x, normals n_i and distances d_i at the corners of the texel containg p_x, and the discontinuity signature for that texel. The output is an estimated normal at p_x.

1 find locations of discontinuity points p_a, p_b, p_c, p_d from the texel's discontinuity signature; set p_e as an intersection of segments $[p_a, p_c]$ and $[p_b, p_d]$;
2 fetch the texel edge indices of end-points of discontinuity segments for the texel;
3 reconstruct normals and distances at the active discontinuity points - the points from $p_a ... p_c$, which are on the same side of discontinuity segments as p_x;
  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 normal from the values at these corners;
3.1.2 one corner: copy the normal from the corner;
3.1.3 none (such active point is a common end-point of the discontinuity segments): use the normal from the opposite end-point of either segment;
  3.2 set distance to zero at every active point which is located on the discontinuity; otherwise, interpolate the distance from the values at reachable corners;
4 determine the triangle containing point p_x; if needed, reconstruct the normal and distance at p_e interpolating between the values at the endpoints of the discontinuity with known values;
5 using the barycentric coordinates of point p_x, compute the interpolated normal and the distance at p_x from known values at the triangle vertices; apply Equation (1) to the resulting values to calculate the final blended normal $\mathbf{n}'(p_x)$.

Figure 12 shows normal interplation for a texel with two discontinuity segments inside.

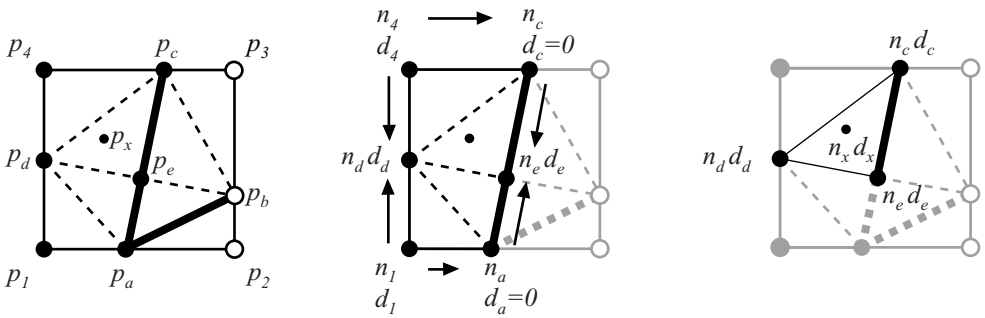

Fig. 12 Computing the normal n_x and distance d_x at a pixel sample p_x. Left: $[p_a, p_b]$ and $[p_a, p_c]$ are the discontinuity segments inside the texel; locations of p_a, p_b, p_c, p_d, and p_e are calculated. Center: computing normals and distances at p_a, p_c, p_d, and p_e. Right: triangle (p_c, p_d, p_e) covers p_x; values at the triangle's corners are used to compute n_x and d_x.

[1] the same rules are applied for reconstructing the gradient of the distance function;