Rendering Surface Details with Diffusion Curves

Stefan Jeschke*
Arizona State University

David Cline†
Arizona State University

Peter Wonka‡
Arizona State University

Figure 1: *Real-time diffusion curve renderings. (Left) The inlay ornaments on the vase and bowl show the sharp detail available with diffusion curve textures. (Right) The crisp displacement boundaries on the stone plate and obelisk result from our dynamic feature embedding.*

Abstract

Diffusion curve images (DCI) provide a powerful tool for efficient 2D image generation, storage and manipulation. A DCI consist of curves with colors defined on either side. By diffusing these colors over the image, the final result includes sharp boundaries along the curves with smoothly shaded regions between them. This paper extends the application of diffusion curves to render high quality surface details on 3D objects. The first extension is a view dependent warping technique that dynamically reallocates texture space so that object parts that appear large on screen get more texture for increased detail. The second extension is a *dynamic* feature embedding technique that retains crisp, anti-aliased curve details even in extreme closeups. The third extension is the application of dynamic feature embedding to displacement mapping and geometry images. Our results show high quality renderings of diffusion curve textures, displacements, and geometry images, all rendered interactively.

Keywords: Line and Curve rendering, Diffusion curves, Displacement mapping, Geometry images

*e-mail:jeschke@cg.tuwien.ac.at
†e-mail:clinedav@gmail.com
‡e-mail:pwonka@gmail.com

1 Introduction

Vector graphics, as alternatives to raster graphics, have the advantage of easy shape manipulations and compact representation that provide high quality even when rendered at very high resolution. The recently introduced diffusion curve images [Orzan et al. 2008] (DCIs) complement existing vector graphics with a great new representation. While diffusion curves and vector graphics share the idea that image boundaries are defined by curves, the rendering process is fundamentally different, since in the case of diffusion curves, the underlying operation is solving a diffusion equation. Diffusion curves provide an intuitive and efficient image representation which is at the same time general in the sense that a wide variety of images can be generated.

This paper maps diffusion curves to surfaces as textures, displacement maps and geometry images. The fundamental challenge in this context is that the diffusion process is tied to a regular grid. One could simply use a diffusion curve image with a specific resolution as a texture or displacement map, but this approach eliminates many of the advantages of the diffusion curve representation, in particular crisp curve boundaries, as illustrated in figures 2 and 6. To take advantage of the DCI representation when mapping diffusion curves to surfaces, we need algorithms that can reconstruct sharp curve boundaries at high zoom levels.

Contributions. The contributions of this paper are three algorithms that enable real-time DCI rendering on surfaces while retaining the vector character that is the hallmark of the diffusion curve style. The first contribution is a view dependent texture space warping function that dynamically allocates more texture resolution for parts of the object that are close to the viewer. This alone may not be sufficient and in closeups, curve edges can still become blurry, destroying the vector character of the rendering. Our second contribution remedies this problem by providing *dynamic* local feature embedding and reconstruction, which allows the solution on the regular grid to be magnified while retaining sharp, anti-aliased curve boundaries. In combination with the warping feature embedding automatically retains curve details at very high zoom levels.

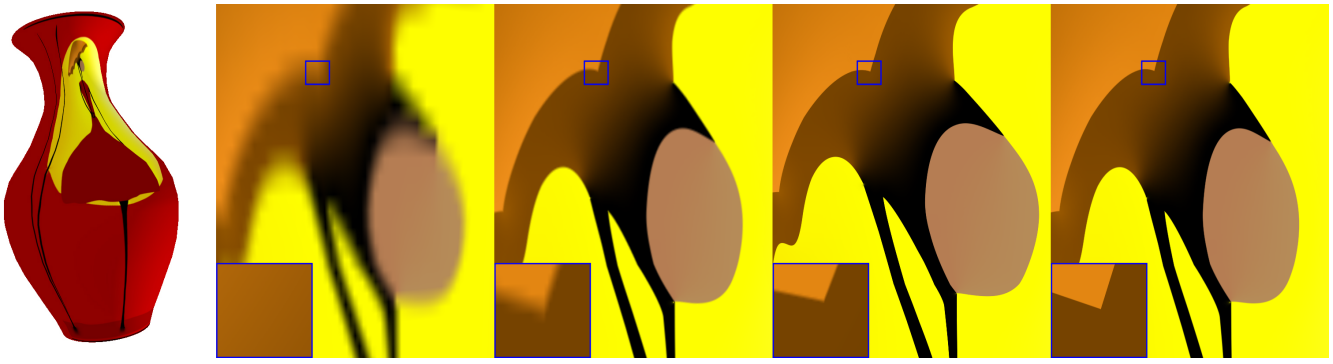

Figure 2: *Texture warping and feature reconstruction. (a) A vase rendered with diffusion curves as color decals (inspired by the African artist ONA). (b) Common texture mapping with an* 800^2 *texture shows very blurry edges in a closeup view. (c) Texture warping gives a virtual resolution of* 39000×42000 *for the same closeup. The edges are still not crisp. (d) Even a warped* 4096^2 *texture (virtual resolution* 159000×176000*) does not provide sharp edges for extreme closeups. In addition, memory usage raises 25 times and the frame rate drops more than 15 times. (e) Dynamic feature embedding guarantees crisp edges even for infinite closeups, here only using an* 800^2 *texture.*

The case of diffusion curves used as displacements is even more difficult, since a naive tessellation creates errors in normals on curve boundaries no matter how high the base resolution (see figure 6). Our third contribution is a feature sensitive meshing technique for displacement maps and geometry images that fixes these artifacts by shifting vertices on the surface to follow the curve boundaries. We demonstrate the visual quality and speed of our techniques with examples of diffusion curve textures, displacement mapped diffusion curves, and diffusion curve geometry images, all rendered at interactive frame rates, typically between 20 and 115 fps.

2 Background

Diffusion Curve Images. Representing an image by its edges was motivated by Elder et al. [2001], who showed that edges are a near complete image representation. Orzan et al. [2008] take Elder's representation a step further by vectorizing edges (calling them *diffusion curves*) and sampling curve attributes such as colors and blur values. The DCI approach uses only a single curve primitive, while the diffusion process brings the curve information together and forms the final image. Current work [Jeschke et al. 2009] provides a fast and robust DCI rendering algorithm based on rendering the Voronoi diagram of the curves, and performing the diffusion with a *variable stencil size* diffusion solver. We adopt this DCI rendering method in this paper.

Vector Graphics Mapped onto Objects. When applying vector graphics onto objects it is essential to have a random-access data structure to locally evaluate the current color, typically in a pixel shader [Nehab and Hoppe 2008]. Previous work mainly varies by the type and amount of local information that is provided. Such local information can be line segments [Sen et al. 2003; Sen 2004; Tumblin and Choudhury 2008; Lefebvre and Hoppe 2006], bilinear curves [Tarini and Cignoni 2005], a cubic curve [Ray et al. 2005], two quadratic segments [Parilov and Zorin 2008] or a fixed number of corner features [Qin et al. 2006]. Some approaches enhance raster images with sharp embedded features [Sen et al. 2003; Sen 2004; Tumblin and Choudhury 2008; Lefebvre and Hoppe 2006] while others use more general vector graphics primitives that are registered to a discrete grid [Qin et al. 2006; Nehab and Hoppe 2008]. If only a fixed number of primitives are stored per cell, some places with high local complexity can dramatically raise the required resolution of the whole grid. This is avoided if cells can have variable complexity [Ramanarayanan et al. 2004; Nehab and Hoppe 2008] at the expense of required texture indirections, a more complicated registration for dynamic layout changes, and replicated data for neighboring cells [Nehab and Hoppe 2008].

Our work tries to combine the simplicity of methods that have fixed storage cost per texel with the flexibility of methods that provide variable memory size according to local image complexity. On the one hand, we locally store a small, fixed amount of feature information per texel (a single point on a curve). At the same time we allocate more texture space for visible regions close to the observer and embed features, all *dynamically per frame* which adapts the amount of detail in a view dependent fashion. This overcomes the limited resolution problem of simpler feature embedding schemes while preserving the convenience of locally having a fixed number of features to evaluate.

Dynamic Texture Space Allocation. Reallocating texture space to locally provide more detail has been presented in several different contexts. Sloan et al. [1998] and Sander et al. [2002] change object parametrization according to the amount of local detail in a texture. Kraus and Ertl [2002] and Carr and Hart [2004] use texture atlases. Perspective shadow maps [Stamminger and Drettakis 2002] and its further developments [Lloyd 2007] reallocate texture space depending on the current viewpoint to provide higher shadow map resolution near the camera. Dachsbacher et al. [2004] warp geometry images in a view dependent fashion for terrain rendering. We present a view dependent texture warp approach that dynamically reparametrizes an object. To do this, we determine the texture space needs of each polygon and reallocate the texture accordingly.

3 Diffusion Curves on Surfaces

Given an object with appropriate parametrization and a set of diffusion curves defined in parameter space, we want to use the diffusion curve as a texture. Simply rendering the DCI at some resolution and using the resulting image as a texture does not retain sharp features, however. The extent of this problem is not really intuitive and is easily underestimated. This is because the vector character of diffusion curves leads to sharp boundaries at all zoom levels. Figure 2(b) is a case in point. At the zoom level shown, an image texture would need a resolution of about 50000^2 to faithfully reproduce the crisp curve boundaries. Even if this much texture memory were available, the diffusion would take more than a full minute at current GPU render rates. The resolution situation becomes even worse when diffusion curves are used as displacements. For example, the

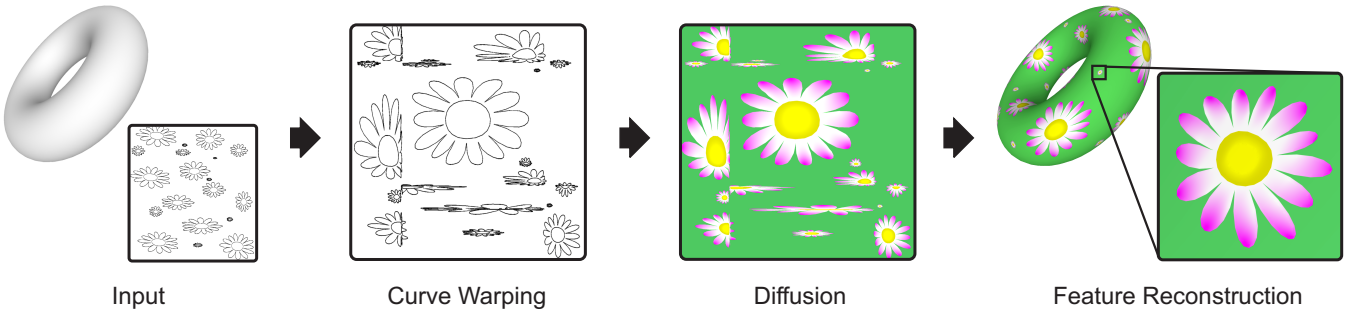

Figure 3: *Overview of our rendering approach: given a parameterized input model and curves defined in texture space, we (optionally) apply curve warping to locally increase texture resolution (by a factor of about 4.5 in this case). The curves are diffused and the resulting image is rendered as a texture on the object. Finally, local feature reconstruction is used to maintain sharp curve boundaries at all resolutions.*

top middle image in figure 6 shows a displacement rendered with a (virtual) resolution of 11000×11000. A naive triangulation of the displacements along the curve boundaries leads to gross errors in surface normals no matter what resolution is used.

3.1 Overview

In this section we present two methods to tackle the resolution problems related to diffusion curve textures. First, prior to diffusing the image we optionally warp the curves to locally increase the texture resolution for visible curves near an observer, at the expense of invisible and more distant ones. Second, during rendering we locally reconstruct curve features using the distance map generated by the diffusion solver. This combination allows us to render detailed, anti-aliased diffusion curves on surfaces at very high quality with a modest base resolution. Figure 3 gives an overview of our approach, which can be summarized as follows:

- Starting with a polygon model, the current viewpoint, and a piecewise linear approximation of the curves defined in texture space, we compute a *texture space warping function*. The curve segments are then warped according to this function (Section 3.2).
- The diffusion is carried out by the variable stencil diffusion solver described in [Jeschke et al. 2009].
- We render the object, applying feature reconstruction to retain sharp anti-aliased features (Section 3.3). Local feature reconstruction can also be used for displacement mapping as discussed in Section 3.5.

3.2 Curve Warping and Diffusion

In order to provide higher texture resolution for visible scene parts close to the observer, we employ a parameter space warping function that can be evaluated in real time and is easily invertible. The function is computed per triangle and resizes texture space according to triangle visibility in the view frustum, whether a triangle is back facing, proximity to the observer, and curve containment (i.e., only triangles containing curves can get more texture space, similar to Sloan et al. [1998] and Lloyd [2007]). Rather than optimizing the individual texture coordinates of all triangles in the mesh, we perform a global optimization separately for each texture coordinate axis, similar to Dachsbacher and Stamminger [2004]. The resulting function is easily invertible.

Given a warping function for each texture coordinate axis, we use the mesh as a "cage" to warp the curves: during rendering we first warp the texture coordinates of the mesh vertices and use the barycentric coordinates of the curve vertices within each mesh triangle to warp the curves. To do this, each linear curve segment has to be registered to the triangle it is placed in. Segments that span multiple triangles are cut and the subsegments are registered separately. The registration process should be fast to allow for animated curves. We accelerate registration with a regular grid structure defined in texture space where each grid cell stores all triangles that at least partly overlap it. This way, during registration of a curve we do not have to check each curve segment against each triangle. With this acceleration structure the curve registration process runs fast enough to allow for interactive updates, even when rendering hundreds of curves.

The texture warping distorts the color gradients in a diffused image, but surprisingly, we have found that it introduces virtually no noticeable artifacts in animations, as can be seen in the video. We believe that this is the result of the fact that the human visual system is not sensitive to smooth color changes.

For the diffusion curve rendering itself we use the variable stencil size diffusion solver presented in [Jeschke et al. 2009]. As a byproduct, the solver outputs a map containing (for each pixel) the closest point on a curve, the color of the *opposite* side of the curve for the same point and a corresponding distance map. These will become important for the feature reconstruction described below.

The diffusion curve definition allows "curve reblurring" in a post-processing step to support unsharp features. We apply the fast separable approximation presented in [Jeschke et al. 2009], but with the blur kernel size being warped in the same manner as the texture. Fortunately, the separable nature of the texture warping function makes this step straightforward.

In our current implementation the whole curve registration process runs entirely on the CPU. If curves are animated, this has to be performed in every frame, otherwise only once as a preprocess. The CPU also computes the warping function and warps the texture coordinates of the mesh vertices as well as the piecewise linear curve segments in every frame. Afterwards, the 3D object and 2D curve segments are transferred from the CPU to the GPU. For higher rendering performance, some of these computations might be performed on the GPU. The curve segment rasterization, diffusion, post processing and final 3D object rendering with feature reconstruction (Section 3.3) is entirely performed in the GPU with the required textures remaining in graphics memory.

3.3 Feature Reconstruction

When sharp features should be provided also for closeup views, simply creating the texture at a larger size is not a viable option be-

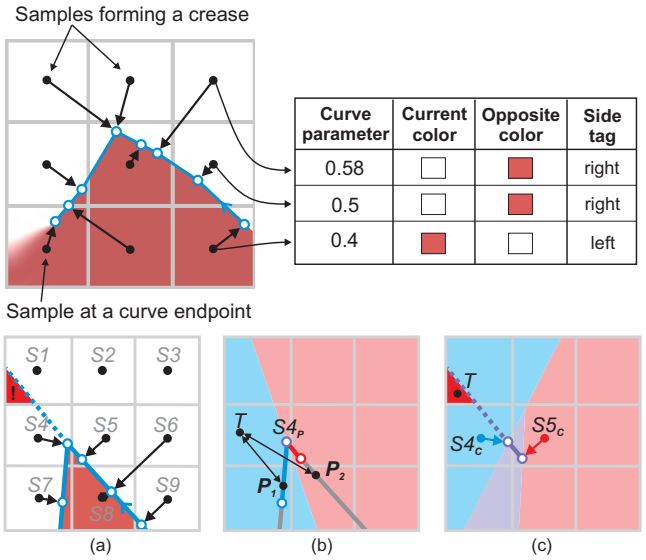

Figure 4: *Top: Local curve reconstruction based on per sample information. The hollow blue dots show the closest curve points for nine samples (black points). The curve is reconstructed by connecting the curve points in order by parameter value. (a): A hard reconstruction case. (b): Extended Voronoi regions of two line segments. (c): Per sample regions for the segment from $S5_P$ to $S4_P$.*

cause of the required memory and time for diffusion, as shown in figure 2. Here, locally reconstructing curves is the key for crisp, anti-aliased curve rendering while maintaining a modest texture size. For our curve reconstruction algorithm we use per texel information (from now on called a *sample* S) (see figure 4, top). Aside from the texel center S_C we need the following data which is created during curve rasterization [Jeschke et al. 2009]:

- the closest curve point S_P to the sample (shown as hollow blue dots in figure 4),
- the curve parametrization value S_t at point S_P,
- the *current side texel color* (after diffusion) and the *opposite side color* (stored during rasterization),
- a *side tag* $S_{side} = \{left, right, end\}$ indicating if the texel is on the left side, right side, or near an endpoint of the curve.

Reconstructed curves are defined as line segments that connect curve points. We classify samples as *valid* and *invalid*. A sample S is valid if its curve point S_P lies in the box enclosing the 8 adjacent texel centers S_C (i.e. S_p is within one texel width of S_c in both x and y). For example, valid samples are $S4$ to $S9$ in figure 4(a)-(c). A linear curve segment is defined between consecutive *valid* curve points. That is, Si_P and Sj_P form a segment if there is no other sample with a parameter value between Si_t and Sj_t.

When rendering a fragment at texture coordinates T we first determine if we are far enough away from a curve so that no feature reconstruction needs to be done. To do this we simply test if any of the four samples closest to T is invalid. If so, we can safely apply bilinear color interpolation as no curve can exist between the samples. If T is too close to a curve, we locally reconstruct the line segments from a 5×5 neighborhood around T. To do this we sort the 25 samples according to parameter value S_t and construct up to 24 consecutive line segments from them. From this set, we determine the segment that is closest to T. It may happen that two segments are equally close to T, when the closest point S_p is a shared vertex

Figure 5: *(a): Standard bilinear interpolation for a 60×60 texel image. (b): The same image with reconstructed features. (c): Closeup in which all texels with valid samples are outlined.*

of two segments. Figure 4(b) shows this situation with $S4_p$ being the closest point. To resolve the tie, we construct two points P_1 and P_2 by walking a fixed distance away from S_P along each segment. The distances to these two new points are then used to break the tie, thus forming a sort of extended Voronoi region as shown in the figure. Qin et al. [2006] present a different solution with the same result to break this tie. Having found the closest line segment, we must determine on which side of the curve T lies (left or right). The side with respect to the segment can easily be computed because it follows the direction of the original curve (see figure 4(top)).

The algorithm above always computes the correct side because the 5×5 neighborhood by definition contains all valid samples needed to locally reconstruct a curve. However, it is rather slow due to comparing and sorting 25 samples. We developed an alternate algorithm that uses only a 3×3 neighborhood and is about 3 times faster. The main problem with this smaller neighborhood is that some important curve segments might be missed. Figure 4(a) and (c) show a particular case (a cusp) where $S7$ lies not in the 3×3 neighborhood around T so that the segment from $S4_P$ to $S7_P$ is missing. However, the closest segment from $S5_P$ to $S4_P$ indicates the wrong side: the red region in sample $S1$ is falsely classified as being "left" of that segment (assuming counter clockwise curve parametrization in the example). Interestingly, $S4_{side} = right$ provides the correct side, which also seems more reliable in this case since it is closer to T than the segment. Obviously, between $S4_C$ and $S4_P$, $S4_{side}$ can safely "overrule" the wrong information. We take this idea and refine the side computation: given a line segment defined by samples Si_P and Sj_P ($S5_P$ and $S4_P$ in figure 4(c)), construct two line segments from Si_C to Si_P and from Sj_C to Sj_P in addition to our current segment. Now compute the closest one of these three segments to T, again using the extended distance definition as described above. This in principle adds "per sample Voronoi regions" as figure 4(c) shows. Finally, the closest of the three segments determines the current side of T. Intuitively this always gives preference to the side information closest to T. While no formal proof can be given here, this faster algorithm always worked correctly during extensive practical tests.

Having the correct curve side, we can apply a color. Colors are bilinearly interpolated between the four samples closest to T in order to keep connectivity to texels further away from a curve. At the same time, the interpolation has to respect the curve boundary. We solve this in the following way: if a sample is on the opposite side from T (i.e., $S_{side} \neq T_{side}$), the *opposite side color* (stored when the texture was created) is substituted for it during interpolation. Finally, anti-aliasing can be applied as in Section 3.4. The color for the opposite curve side is computed by inverting the color side decisions made for all four texels and interpolating them.

Endpoints: The handling of endpoints is simple: if any sample defining the current line segment is an endpoint $S_{side} = end$, we apply bilinear color interpolation of all *current* colors, since near

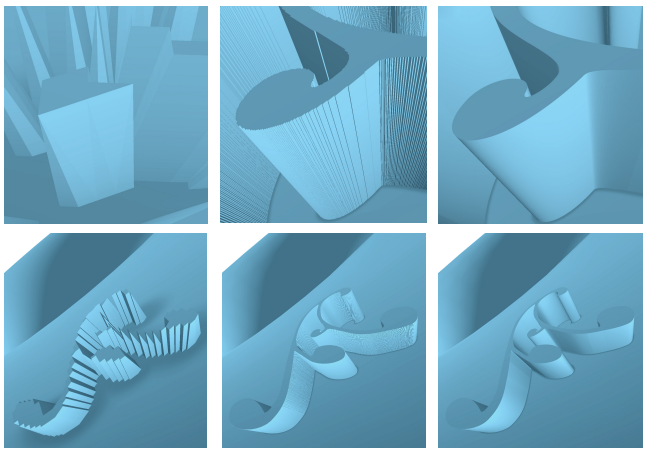

Figure 6: *(Left) The straightforward application of an* 800×800 *diffusion curve displacement map on the surface of a torus only provides a resolution of* 38^2 *displacement values for the decal. Middle: view dependent texture warping raises this resolution to about* 530^2*, but the illumination still shows stair-step artifacts on sharp boundaries. (Right) Adding local feature reconstruction leads to a high quality rendering, even in extreme closeups.*

endpoints colors diffuse from both sides of a curve.

Multiple curves: Line segments must not be formed between different curves. We exclude them by adding a number for each parameter S_t that enumerates the curves $(2, 4, 6, .., 2n$ for n curves). This way, a parameter difference larger than 1 tells us that two samples point to different curves. Similarly, during color interpolation, we always take the *current* color of a sample if it points to a different curve than the one closest to T.

Figure 5 shows an example for curve reconstruction on an image of only about 60×60 pixels. Notice the sharp and accurate color boundaries despite the low image resolution and bilinear interpolation artifacts near the corners.

3.4 Filtering and Anti-aliasing

Filtering and anti-aliasing are essential for high quality minified and magnified viewing. For minified viewing we resort to mip-mapping the final image. Anti-aliasing is performed using the distance map and the current pixel footprint size provided by the graphics hardware: if the distance d is smaller than half a pixel ($0 \leq d < \frac{1}{2}$), a curve passes through the current pixel, which needs to be anti-aliased. In this case we blend the color at the current pixel c_{pixel} with the color on the other side of the curve c_{other} (as was stored during curve rasterization). The final color c_{final} is computed as

$$c_{final} = (1 - 2 * d) * (\frac{c_{pixel} + c_{other}}{2}) + 2 * d * c_{pixel} \qquad (1)$$

One can see that if $d = 0$ (i.e. the pixel is exactly on a curve), the average of the two colors is taken, and at $d = \frac{1}{2}$ the current pixel color is retained.

3.5 Diffusion Curve Displacement Mapping

Because of their sharp features, diffusion curves have a major advantage over raster images for representing certain types of displacement maps [Cook 1984]. Figure 6 brings this advantage to light. In a surface displaced by a raster image, if sharp features do not align with pixel boundaries, jagged artifacts appear in the geometry. This is only partially remedied by increasing the resolution. Using a diffusion curve to define the displacement offers the chance to get rid of the artifacts through feature embedding.

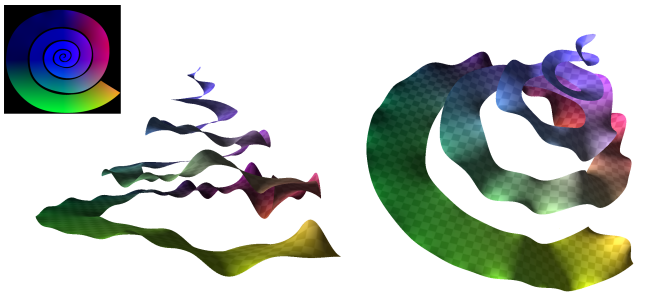

Figure 7: *A wavy spiral defined as a DCI geometry image.*

There are two general approaches to render a displaced surface: ray casting or displacing vertices of a fine mesh. We opt for the second approach since accounting for the distorted texture space and finding local features seem very costly for GPU ray casting methods [Policarpo et al. 2005]. To render displacement mapped diffusion curves, we reuse all steps introduced in Section 3, but we must add an additional step to make sure that curve boundaries remain sharp in the reconstruction. We first sample a geometry image [Gu et al. 2002] from the object that should be displaced. This is done by rasterizing each object polygon into a texture of the same size as the DCI using its (warped) parametrization. Every texel stores the corresponding 3D position and normal, both transformed to camera space. Next, we render a fine mesh that uses texels of the geometry image as vertices. The stored position, normal, and displacement value (stored in the DCI) define the 3D position of every such vertex. The result is a displaced object with higher mesh resolution in visible areas close to the observer, thanks to the warping method presented in Section 3.2.

To retain sharp features, we move vertices placed at *valid* samples (the ones defining the curve, see Section 3.3) to the respective position S_C directly *on* the curve and reread the position and normal at those coordinates. This simple method retains sharp feature boundaries in most cases. However, we have to note that this method does not work for thin features like sharp long cusps as the number of vertices is not sufficient to reconstruct all curves. In practice this means that one side of the cusp might not appear perfectly smooth. Another issue is that the texture warping function only takes the base surface into account, but not the actual displacement, like methods specifically designed for displacement mapping [Moule and McCool 2002; Doggett and Hirche 2000; Gumhold and Huettner 1999]. Because of this, artifacts are sometimes introduced at sharp object corners and large displacements when the displaced surfaces should be visible, but the base surface is not.

3.6 Diffusion Curve Geometry Images

We have tried a number of experiments with representing geometry images [Gu et al. 2002] directly as diffusion curves. In this representation, a 3D object is defined by a set of curves in space with minimal surfaces between them. The diffusion process remains the same as in the image and displacement cases. Figure 7 gives one example of a diffusion curve geometry image. One appealing characteristic of this representation is that the curves can be moved in texture space without changing the shape of the object. This points to the possibility of editing tools that open up parts of texture space where more detail is needed, for example. One area for future work would be to define a set of GUI tools for creating and editing diffu-

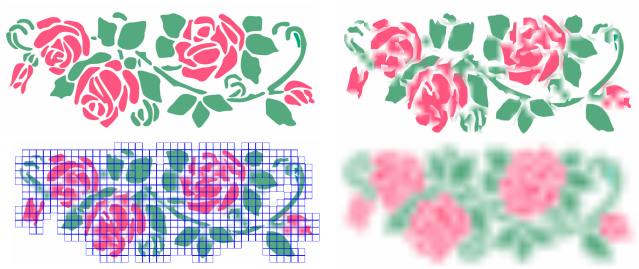

Figure 8: *A failure case for curve reconstruction. Top left: high resolution image. Top right: Curve reconstruction inaccuracies due to multiple curves crossing single texels. Lower left: the pixel grid used for reconstruction. Lower right: bilinear interpolation at the same resolution.*

sion curve geometry images efficiently.

3.7 Discussion

One might ask why we do not store the algebraic curve descriptions for the feature reconstruction to provide theoretically infinite curve resolution [Qin et al. 2008; Nehab and Hoppe 2008]. One reason is that any practical diffusion process works on sampled data, so it seems natural to provide approximately the same amount of detail along the curves. Another reason is consistency: since the piecewise linear curve approximations used during rasterization do not exactly match the algebraic curves, some texels would lie on opposite curve sides, resulting in false colors during feature reconstruction.

The very high local resolution provided by the texture warping, combined with the feature reconstruction, produced smooth curves in all our examples even at very close inspection. Of course, details can only be correctly reconstructed if enough information has been stored in the texture, but even with texture warping the resolution may locally not be high enough in some situations. Figure 8 shows such a case where we chose an extremely low resolution. In the top right, the undersampled details do not provide a consistent reconstruction. For comparison, the lower right shows the same image bilinearly interpolated. In this case, feature reconstruction still seems preferable to bilinear filtering as it retains more structural information. Our representation cannot correctly handle curve intersections, as is the case for all approaches based on real-time per-pixel feature reconstruction [Parilov and Zorin 2008]. Fortunately, as texture resolution is typically high due to the warping, such artifacts are small on screen which makes them less noticeable. In addition, since we use a small neighborhood around the curves to gather curve information, we achieve slightly higher curve resolution compared to previous methods based on individual pixels.

In many practical applications, textures are tiled or used for multiple objects. Computing the warping and diffusion for each instance provides high quality but is potentially costly. The warping and diffusion can also be computed for the closest instance and the resulting DCI reused for more distant ones, for example, if a texture is tiled along a wall. In general, studying efficient strategies for such cases is an interesting avenue for future work.

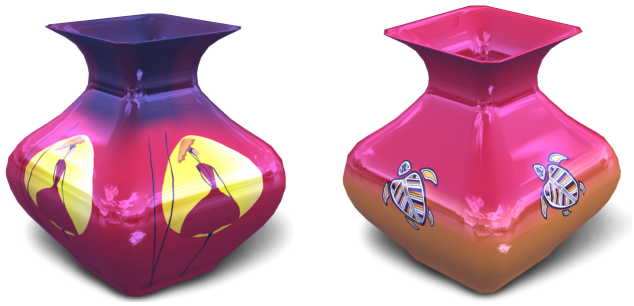

Figure 9: *A vase rendered with different diffusion curve textures.*

4 Results

4.1 Diffusion Curve Textures

Figure 1 (left) shows the image quality possible with DCI textures, highlighting the sharp curve boundaries provided by our texture warping and feature embedding. Note the sharp and anti-aliased curves on the vase and bowl, and in particular, the closeup view of the bowl with detailed inlays. This quality is achieved with a base texture of only 800^2 texels, which keeps memory requirements and the computational effort for the diffusion fairly low. Unless otherwise indicated, all examples have been created with this DCI resolution. In the closeup view the image warping provides a virtual resolution of 2300×6600, with even sharper feature reconstruction.

Figure 9 shows two other examples of diffusion curves used as texture maps. Note the expressive gradients on the vase tops that would be difficult to obtain with standard vector graphics techniques. As demonstrated by the environment map reflections, we are using diffusion curves exactly as we would common surface textures.

The video shows the quality that our renderer can achieve for real-time viewing. We are able to reproduce crisp anti-aliased curve edges at high zoom scales, and the texture appearance is completely stable, despite warping that occurs in the underlying texture.

Table 1 gives frame rates for different DCI resolutions. (All 3D renderings in this paper have been created at a resolution of 800^2.) Note that even with a DCI resolution of 1024^2, the renderer consistently maintains almost 40 fps. Paradoxically, the render speed is lower for a 128^2 DCI texture than for a 256^2 texture. This is because the lower resolution causes more pixels to perform feature reconstruction, invoking a slower path in the pixel shader.

Table 2 varies the number of curve segments, adding multiple copies of the "Ona" decal from figure 2. As the table shows, there is a fairly linear drop off in performance as more curve segments are added, with each new segment increasing the render time by about 0.04 ms. We believe this value to be rather consistent since after the first few curves are rasterized, the distance map rendering for new curves will mostly be culled by hierarchical Z culling.

Table 3 toggles texture warping, feature reconstruction and reblurring, while keeping the other two effects on. Similar to lowering the DCI resolution, turning off texture warping increases the number of pixels that must perform feature reconstruction, and the frame rate actually decreases when we turn off the warping. Feature reconstruction itself is fairly time intensive, as shown in the table, but it is essential for zoomed views. On the other hand, reblurring takes so little time that it does not even register in our test.

4.2 Displacement Mapping

Figure 1 (right) shows displaced hieroglyphs on an obelisk and a stone tablet. Note how precise the displacements are, even in the zoomed inset. It is practically impossible to achieve this quality with raster displacements, as demonstrated in figure 6. Despite the large number of polygons, the rendering still runs at about 20 fps for the stone tablet and obelisk, thanks to the high polygon throughput of current graphics hardware.

The variety of displacement map applications achievable by our framework ranges from small scale features like the hieroglyphs to medium sized features such as the picture frame shown in figure 10(top) to more global features like the golf course relief in figure 10(bottom), in the spirit of Bruneton and Neyret [2008]. The representation of these examples is remarkably simple: the picture frame only consists of 4 curves while the golf course requires only 12.

Table 4 compares render times for different sized DCI displacement maps. The results shown in the paper were all rendered with 800^2 displacement maps. Table 5 gives the performance of our displacement map renderer for different numbers of curve segments, with a DCI resolution of 800^2. The frame rate achieved here is about half that of the texture renderer, but is still quite interactive for up to a thousand curve segments. In table 6 we can see the cost of animating the curves. The minimal extra overhead ($\sim 15\%$) occurs because the renderer must re-register all of the curve segments against the triangles of the input mesh for each frame.

4.3 Geometry Images

DCI geometry images achieve similar performance to displacements, with similar dependencies on the DCI resolution and the number of curves rasterized, as shown in tables 7 and 8. The intriguing aspect of the diffusion curve geometry images is that we have a completely new curve-based object representation. Figure 7 shows an example of a DCI geometry image, and another is shown in the video. Notice how precisely the curve contours are captured by our feature embedding. This would not be possible by just sampling the surface at higher resolution on a rectangular grid.

5 Conclusions and Future Work

This paper makes several contributions to enable high quality diffusion curve rendering on surfaces. The combination of a view dependent texture space warping and dynamic feature reconstruction allows sharp, high quality renderings of diffusion curve textures with a low base resolution. The amount of displayed curve detail is automatically adapted to the viewing distance. We also demonstrate the application of dynamic feature reconstruction to displacement mapping. These contributions enable several applications as demonstrated in the paper: diffusion curves on surfaces, displacement mapped diffusion curves, and diffusion curve geometry images. These applications all run interactively, and allow interactive creation, editing and curve manipulation.

We believe that our rendering solutions provide several opportunities for future work. While we demonstrated our approach for the specific case of diffusion curves, it seems interesting to study its applicability to conventional vector graphics. Another idea is to use diffusion curves to model general three-dimensional surfaces. Our initial experiments with geometry images have offered some encouraging results, but we would like to develop efficient tools to create and edit diffusion curve geometry images.

Figure 10: *Top: A picture frame, modeled with only four curves that are colored red in the displacement map image at the top left. Bottom: The relief of a golf course modeled using only 12 curves.*

Acknowledgements

We thank Orzan et al. [2008] for making their curve drawing application online. This work was supported by the NSF and the Austrian Science Fund, Project FWF P20768-N13.

References

BRUNETON, E., AND NEYRET, F. 2008. Real-time rendering and editing of vector-based terrains. In *Proceedings of Eurographics '08*, vol. 27, 311–320.

CARR, N. A., AND HART, J. C. 2004. Painting detail. In *ACM Trans. Graph. (SIGGRAPH 2004)*, vol. 23, 845–852.

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

DACHSBACHER, C., AND STAMMINGER, M. 2004. Rendering procedural terrain by geometry image warping. In *Proceedings of Eurographics Symposium on Rendering*, 103–110.

DOGGETT, M., AND HIRCHE, J. 2000. Adaptive view dependent tessellation of displacement maps. In *Proceedings of the conference on Graphics hardware*, 59–66.

ELDER, J. H., AND GOLDBERG, R. M. 2001. Image editing in the contour domain. In *PAMI '01*, 291–296.

GU, X., GORTLER, S. J., AND HOPPE, H. 2002. Geometry images. In *ACM Trans. Graph. (SIGGRAPH 2002)*, vol. 21, 355–361.

GUMHOLD, S., AND HUETTNER, T. 1999. Multiresolution rendering with displacement mapping. In *Proceedings of the conference on Graphics hardware*, 55–66.

Resolution	# Segments	FPS (min / max)
128 x 128	159	48 / 49
256 x 256	159	62 / 62
512 x 512	159	48 / 50
800 x 800	159	41 / 43
1024 x 1024	159	39 / 41

Table 1: *DCI textures, varying resolution.*

Repetitions	# Segments	FPS (min / max)
1 x	159	48 / 49
2 x	318	39 / 40
4 x	636	27 / 27
8 x	1272	17 / 18
16 x	2544	9 / 10
32 x	5088	5 / 6

Table 2: *DCI textures, varying # of curve segments.*

Figure	# Segments	FPS (off / on)
texture warping	159	39 / 42
feature reconstruction	159	52 / 42
blurring off / on	159	42 / 42

Table 3: *DCI textures, turning features on and off.*

Resolution	# Segments	FPS (min / max)
128 x 128	35	221 / 222
256 x 256	35	154 / 155
512 x 512	35	74 / 74
800 x 800	35	38 / 39
1024 x 1024	35	25 / 25

Table 4: *DCI displacements, varying resolution.*

Figure	# Segments	FPS (min / max)
fig. 3 torus decal	35	38 / 39
fig. 11 picture frame	44	39 / 40
2 scrolls	70	30 / 30
4 scrolls	140	28 / 28
fig. 12 golf course	200	21 / 22
fig. 1 stone plate	468	20 / 20
fig. 13 clock tower	478	17 / 18
fig. 1 obelisk	579	20 / 21
32 scrolls	1120	10 / 10

Table 5: *DCI displacements, varying # of curve segments.*

Repetitions	# Segments	FPS (off / on)
1 scroll	35	38 / 35
2 scrolls	70	30 / 28
4 scrolls	140	28 / 25
8 scrolls	280	22 / 19
16 scrolls	560	16 / 14
32 scrolls	1120	10 / 9

Table 6: *DCI displacements, turning on and off animation.*

Resolution	# Segments	FPS (min / max)
128 x 128	35	240 / 246
256 x 256	35	164 / 169
512 x 512	35	70 / 71
800 x 800	35	32 / 32
1024 x 1024	35	19 / 19

Table 7: *DCI geometry image, varying output resolution.*

Repetitions	# Segments	FPS (min / max)
1 x	35	32 / 32
2 x	70	27 / 27
4 x	140	25 / 25
8 x	280	20 / 20
16 x	560	14 / 14
32 x	1120	9 / 10

Table 8: *DCI geometry image, varying # of segments.*

JESCHKE, S., CLINE, D., AND WONKA, P. 2009. A GPU Laplacian solver for diffusion curves and Poisson image editing. In *ACM Trans. Graph. (SIGGRAPH ASIA 2009)*, vol. 28, 1–8.

KRAUS, M., AND ERTL, T. 2002. Adaptive texture maps. In *Proceedings of the conference on Graphics hardware*, 7–15.

LEFEBVRE, S., AND HOPPE, H. 2006. Perfect spatial hashing. In *ACM Trans. Graph. (SIGGRAPH 2006)*, vol. 25, 579–588.

LLOYD, B. 2007. *Logarithmic Perspective Shadow Maps*. PhD thesis, University of North Carolina at Chapel Hill.

MOULE, K., AND MCCOOL, M. D. 2002. Efficient bounded adaptive tessellation of displacement maps. In *Proceedings of Graphics Interface*, 171–180.

NEHAB, D., AND HOPPE, H. 2008. Random-access rendering of general vector graphics. In *ACM Trans. Graph. (SIGGRAPH Asia 2008)*, vol. 27, Article 135.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: A vector representation for smooth-shaded images. In *ACM Trans. Graph. (SIGGRAPH 2008)*, vol. 27, Article 92.

PARILOV, E., AND ZORIN, D. 2008. Real-time rendering of textures with feature curves. *ACM Trans. Graph. 27*, 1, 1–15.

POLICARPO, F., OLIVEIRA, M. M., AND COMBA, J. L. D. 2005. Real-time relief mapping on arbitrary polygonal surfaces. In *Proceedings of I3D '05*, 155–162.

QIN, Z., MCCOOL, M., AND KAPLAN, C. 2006. Real-time texture-mapped vector glyphs. In *Proceedings of I3D '06*, 125–132.

QIN, Z., MCCOOL, M., AND KAPLAN, C. 2008. Precise vector textures for real-time 3D rendering. In *Proceedings of I3D '08*, 199–206.

RAMANARAYANAN, G., BALA, K., AND WALTER, B. 2004. Feature-based textures. In *Proceedings of the Eurographics Symposium on Rendering*, 265–274.

RAY, N., NEIGER, T., CAVIN, X., AND LÉVY, B. 2005. Vector texture maps. Tech. rep., INRIA - ALICE.

SANDER, P. V., GORTLER, S. J., SNYDER, J., AND HOPPE, H. 2002. Signal-specialized parameterization. In *Proceedings of Eurographics Workshop on Rendering*, 87–98.

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

SEN, P. 2004. Silhouette maps for improved texture magnification. In *Proceedings of the conference on Graphics hardware*, 65–73.

SLOAN, P. J., WEINSTEIN, D. M., AND BREDERSON, J. D. 1998. Importance driven texture coordinate optimization. In *Proceedings of Eurographics '98*, vol. 17, 97–104.

STAMMINGER, M., AND DRETTAKIS, G. 2002. Perspective shadow maps. In *ACM Trans. Graph. (SIGGRAPH 2002)*, vol. 21, 557–562.

TARINI, M., AND CIGNONI, P. 2005. Pinchmaps: Textures with customizable discontinuities. *Computer Graphics Forum 24*, 3, 557–568.

TUMBLIN, J., AND CHOUDHURY, P. 2008. Bixels: Picture samples with sharp embedded boundaries. In *Proceedings of Eurographics Symposium on Rendering*, 186–196.