Eurographics Symposium on Rendering (2004)
A. Keller, H. W. Jensen (Editors)

Bixels: Picture Samples with Sharp Embedded Boundaries

Jack Tumblin and Prasun Choudhury

Department of Computer Science, Northwestern University, Evanston, IL, USA.

Abstract

Pixels store a digital image as a grid of point samples that can reconstruct a limited-bandwidth continuous 2-D source image. Although convenient for anti-aliased display, these bandwidth limits irreversibly discard important visual boundary information that is difficult or impossible to accurately recover from pixels alone. We propose bixels instead: they also store a digital image as a grid of point samples, but each sample keeps 8 *extra bits to set embedded geometric boundaries that are infinitely sharp, more accurately placed, and directly machine-readable. Bixels represent images as piecewise-continuous, with discontinuous intensities and gradients at boundaries that form planar graphs. They reversibly combine vector and raster image features, decouple boundary sharpness from the number of samples used to store them, and do not mix unrelated but adjacent image contents, e.g blue sky and green leaf.*

Bixels are meant to be compatible with pixels. A bixel is a image sample point with an 8 *bit code for local boundaries. We describe a boundary-switched bilinear filter kernel for bixel reconstruction and pre-filtering to find bixel samples, a bixels-to-pixels conversion method for display, and an iterative method to combine pixels and given boundaries to make bixels. We discuss applications in texture synthesis, matting and compositing. We demonstrate sharpness-preserving enlargement, warping and bixels-to-pixels conversion with example images.*

1. Introduction

There is a shortfall between the data we store and the features we see in digital pictures. Most commonly, a digital picture is a grid of pixels that follow classical sampling-and-reconstruction methods [OS75] to describe a smooth, continuous-valued 2-D intensity map. Display devices approximately reproduce this intensity map, and the number of pixels strictly limits both the image complexity and its perceived "sharpness". But visible features are rarely affected by these limits; instead we see both intensity changes and a critically important set of discontinuities at boundaries caused by shadows, silhouettes, cracks, glints, outlines and occlusions in the depicted scene. The human visual system actively seeks out estimates of these boundaries and their causes. To better understand a viewed image it will even invent boundaries [Pal99] in regions that seem to contain unseen occlusions, (e.g. Kanizsa, Poggendorf illusions), even when obscured by blurring, noise, or missing pixels.

Although humans find visual boundaries in a just few glances, no completely accurate method yet exists to compute them from pixels alone (see [Eld99] for an insightful summary). Pixel storage complicates computed boundary extraction, because anti-aliased pixels hold a weighted average of local scene values. Complex, ad-hoc corrections such as "font hinting" help us see visual boundaries better because pixels ignore boundaries, and can mix together unrelated scene values that are visually distinct. Reversing this process is difficult at best, and may be impossible if the boundary shapes or intensity changes are too complex, as we might find in an image of evening stars and clouds at sunset, occluded by the leaves of a distant tree. Such irreversible mixing often complicates processes that rely on boundaries, such as image compositing, texture synthesis, and texture mapping, image inpainting, or merging depth images recorded from different viewpoints. Good visual boundary estimates are also critical to many novel non-photorealistic rendering (NPR) methods [DFRS03], and important for image indexing and retrieval. Given their importance to perception and graphical computing tasks, we believe that visual boundaries should be an explicit part of digital images, stored in machine-readable form to allow computers to more easily assist us in boundary-related tasks.

Bixels describe sampled digital images with embedded planar geometric boundaries between samples that *do not*

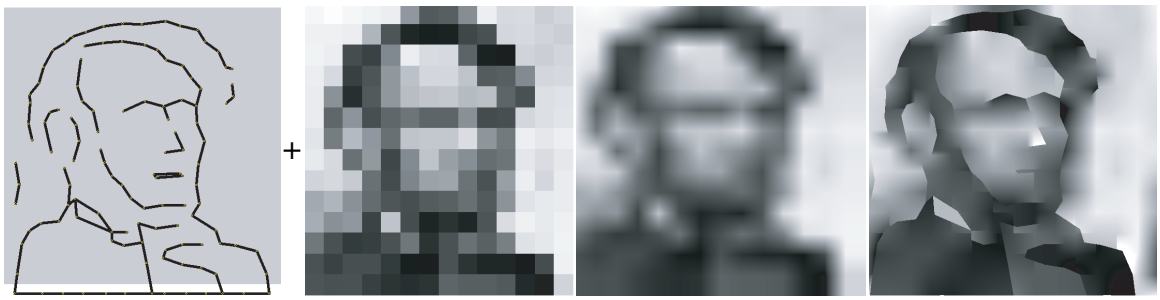

Figure 1: *Pixels degrade or discard visual boundaries, but bixels preserve them in machine readable form for enlargement, compositing, morphing and image-based shape description. Just* 14×14 *bixels can hold many important boundaries, even if only approximated. Boundary positions and source pixels (a and b); 40X bilinear enlargement of* ***pixels*** *(c) and* ***bixels*** *(d). (Image: after Harmon&Julesz,1973)*

mix together unrelated scene intensities. Like pixels, bixels are arranged in a uniform grid, and each bixel has two parts: a point sample of a smoothed 2-D image that is only piecewise-continuous, and a fixed number of additional bits (usually 8) that specify nearby visual boundaries with sub-pixel precision. These boundaries connect to form planar graphs whose nodes and arcs thread between the grid of bixel sample points, and define infinitely sharp discontinuities in image intensities I and gradients ∇I. Note that bixel samples are not pixel values; they differ from pixels because they do not mix unrelated colors separated by visual boundaries, such as the silhouette of a leaf against the sky. The green leaf bixels contain no blue and the adjacent blue sky bixels contains no green; thus bixels will need anti-aliasing for best results when converted to pixels for display, but boundary sharpness will not depend on the density of sample points as it might with pixels. Instead, we can choose bixel sampling rates that best match the complexity of boundaries and intensity variations.

While bixels can sometimes store an image more efficiently than pixels, e.g. a simple corporate logo, they are not intended for image compression; block-transform and wavelet methods (e.g. JPEG, MPEG) are far more efficient. Instead, bixels are an attempt to describe important visual boundaries in a machine-readable form, boundaries that are ignored in pixel-only formats. As shown in Figure 1, bixel boundaries remain sharp for any amount of enlargement, yet convert to anti-aliased pixels for display.

Bixels try to straddle the 'vector/raster' dichotomy in computer graphics [Dur02] and now in web documents [DJ03]. A bixel image can preserve font outlines of letters that were embedded in images, keep the sharp tips and corners of arrowheads and boxes machine-readable, and precisely describe depth discontinuities for image-based modeling and rendering. Bixels can also describe more complex boundaries in real images such as object silhouettes, self-occlusion boundaries, shadow edges and shadow maps, and even express perceived but illusory boundaries in machine-readable form. Because accurate boundary information is routinely available during rendering, bixels seem especially well suited for re-using computer graphics images. Gathering accurate boundaries from photographed scenes remains an important research challenge, but new work by Raskar *et. al.* [RR04], Kwatra *et al.* [KSE*03] and others offer promising approaches to the problem.

2. Related Work

Raster image descriptions of boundaries have appeared before in the computer graphics and vision literature, but none we found provided all the features of bixels. Unconnected line segment samples known as *edgels* are commonly used in image processing for feature detection, but lack connectivity and sub-pixel accuracy. Marching cubes and related methods cleverly create closed polygonal meshes to follow iso-surfaces in sample grids of scalar and vector values [LC87] without non-manifold boundaries. We found several edge-directed image interpolation methods (e.g. [SKS97]) that improve sharpness by making sub-pixel boundary estimates, but none that can embed boundaries for further use.

The literature for boundary finding from pixels is vast and diverse, but bixels do not address this problem; they store boundaries as an intrinsic part of the image they describe. Early and diverse derivative-estimate methods were made optimal by Canny [Can86], and led to refined directional filter bank methods such as the versatile steerable filters [FA91]. Extensive explorations of scale space enlivened robust work in diffusion [PM90] and PDE-based methods, now being refined and greatly expanded by novel level-set approaches [OF02]. A forward-looking paper by Elder [Eld99] asked if sharpness-controlled boundaries alone can express all the important visual content of an image, and influenced our thinking on bixels. Elder [Eld99] represented images made solely of oriented edges described by

their smoothness and their amplitude, and reconstructed display images using a Poisson-solver like method. We agree with Elder's larger ideas, but attempted a much smaller step. Our bixel framework retains the sample grid of a pixel-only image, but adds infinitely-sharp discontinuities.

Even the earliest NPR papers [SALS96] sought visual boundaries to guide rendering, and the topic remains central and active [DFRS03]. Many varieties of stroke-based methods resolved line-like primitives into anti-aliased pixels rather than boundaries, and others labelled mesh edges in 3D rather than image space for specialized rendering: see [GG01] for a good overview. But in an early and very insightful paper, Salisbury *et al.* [SALS96] also saw the fundamental importance of accurate boundary primitives embedded within images, and our work echoes and expands on several of their ideas.

Visual boundaries received sustained attention in computer graphics for rendering silhouettes and shadows, from discontinuity meshing for global illumination [Hec92, LTG92] and piecewise interpolants [SLD92], to silhouette clipping by Sander *et al.* [SGG*00] and an innovative solution by Lokovic *et al.* [LV00] for rendering extremely numerous complex boundaries such as hair and fur. Last year Sen *et al.* [SCH03] greatly improved shadow maps by storing closed, piecewise linear boundaries with sub-pixel accuracy. Bixels store similar boundaries, but also permit non-closed boundaries and continuous-valued intensity interpolation of samples between them. In a concurrent paper published soon [Sen04], Sen uses programmable shaders to embed boundaries in hardware texture maps in a manner very similar to bixels.

Last year, Bala *et al.* [BWG03] improved efficiency for anti-aliased real-time ray-tracing by using bixel-like geometric boundaries and sparse eye rays to accurately estimate pixel colors. In another concurrent paper [RBW04], they extend these ideas to texture maps as well. Adapting Bala *et al.*'s [BWG03] system to produce bixel outputs appears straightforward, and would enable re-sizeable display and use of their work as input for image-based modeling, improved methods for texture synthesis [KSE*03, ZZV*03] and novel displacement maps [WWT*03].

Explicit embedded boundaries may also hold promise for light maps; work by Ng *et al.* [NRH03] on "all frequency lighting" describes sources with arbitrary sharpness by wavelet coefficient selection; simpler but similar results might be achieved by controllably smoothing the infinitely sharp boundaries offered by bixels.

3. Bixel Images Defined

Both bixels and pixels approximate an ideal signal that we call the "scene" (see Figure 2). The scene is boundlessly detailed and perfect, such as the true radiances that a camera tries to measure, or the true depths sampled by a z-buffer.

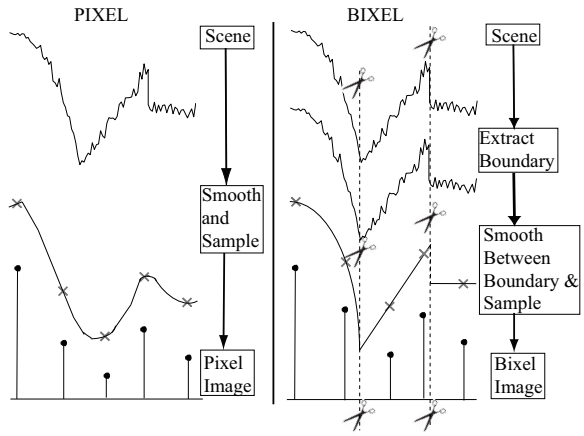

Figure 2: *Pixels mix scene values from both sides of intensity and gradient discontinuities; bixels store the boundary locations (scissor cuts) and store separately smoothed scene values from each side.*

The visual boundaries stored by bixels approximate ideal boundaries that separate adjacent but unrelated portions of the ideal scene, as if the scene signal were an intensity height-field surface cut by scissors, but otherwise left unchanged.

The process of making anti-aliased pixels ignores these scissor cuts. First, it smoothes (pre-filters) the scene to avoid aliasing, and then finds values at integer (x,y) sample positions in the smoothed scene to store as pixels. (note: even the A-buffer [Car84] and other super-sampling schemes are approximations of such smoothing) Due to this smoothing, each pixel is a weighted average of nearby scene values, even if some values were separated by a visually important boundary or "scissor cut". As shown in Figure 2, pixel image creation degrades ridge-like and step-like features from the scene, producing a set of blunted sample values that can differ noticeably from bixels.

"Scissor cuts" strongly affect bixel-making. Unlike pixels, a bixel image stores planar graphs of boundaries that follow "scissor cuts" with sub-pixel precision. Like pixels, bixels also use integer (x,y) sample positions to take sample values from a smoothed scene, but the scene smoothing method is different. Bixel pre-filtering smoothes the surface on each side of a cut separately, and preserves the separate scene intensities and local gradients as much as possible, as described in Section 4.1. Ideally, bixel making smoothes each part of the scene intensities by just enough to allow alias-free reconstruction. Bixel samples within each area can then recreate the smoothed area without significant aliasing or loss of visual detail.

But without further constraints, bixels can define pathological images that are difficult or impossible to reconstruct.

For example, a closed loop of boundaries that does not contain at least one integer (x,y) sample position will enclose an area with no defined intensities. Overly dense or complex boundaries are troublesome too, such as a many-turn spiral packed within a square of 4 adjacent pixels. We avoid these difficulties by defining our terms and imposing a set of rules:

Bixel Sample Values: The bixel image intensity at a single sample point. Bixel sample points form a unit grid at integer *x*,*y* positions.

Tiles: Bixel images are easier to describe by dividing the (x,y) plane into addressable "tiles". Tiles are unit squares whose corners meet at the integer (x,y) locations of bixel sample points. We identify tile corners by letter, in counterclockwise (CCW) order: *A* for the lower left corner, the *B* corner is lower right, and *C* and *D* ar at the upper right and left respectively. The integer (x,y) location of its *A* corner at the lower left also gives each tile its unique address.

Boundary Points and Boundary Segments: All the boundaries described in a bixel image form a set of planar undirected graphs; boundaries can form loops, but are not required to have form closed shapes. Graph nodes are called *boundary points* and are stored within the tiles that contain them, and the graph arcs are *boundary segments* that selectively connect together adjacent boundary points. To avoid ambiguity, three rules limit the complexity and position of the boundary graphs:

Rule 1: Each tile can contain either one boundary point or none at all. Each tile's lower-left corner sample (A) stores the boundary point position (x_p, y_p) with fixed $M \times N$-bit precision, and typically $M = N = 3$ bits, providing an 8×8 grid of allowable boundary positions within each tile. To avoid placing boundary points at tile corners, all grid positions include a half-grid-cell offset of $(2^{-(1+M)}, 2^{-(1+N)})$.

Rule 2: No more than one boundary segment may cross the side of a tile, as shown in Figure 3. This rule ensures that both sides of all boundary segments have at least one nearby bixel sample value to determine its color.

Rule 3: No boundary segment may cross or intersect a tile corner, because boundaries denote discontinuous intensities or gradients, and would be ambiguous at a sample point. Boundary segments cross over the tile sides, but never tile corners.

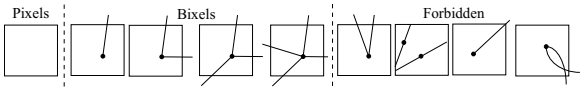

Figure 3: *A variety of tiles and boundary segments: (a) pixels have tiles without boundaries, (b) bixels have 0 to 4 boundary segments that meet at one boundary point, and each segment must cross a different tile side; (c) all other tile configurations are invalid.*

In our implementation, all boundary segments are straight lines for simplicity. Curved segments might be useful for fonts or extreme enlargements (such as Figure 11) but require more complex rules to avoid extra intersections, as shown in the last tile in Figure 3. Straight boundary segments and the three rules above also limit graph connectivity; segments leaving a boundary point can only connect to boundary points in the tile above, below, or to the right or left. As shown in Figure 3, these rules forbid several seemingly useful boundary configurations. These "complex" tiles are quite rare in our test images, as they were for Bala *et al.* [BWG03] who used slightly different boundary rules. In early discussions with them that predated both of our papers, we agreed that tile subdivision to form multi-resolution tiles can handle arbitrary boundary complexity, but we did not need them for any test images we explored.

Just two bits per tile can encode all boundary segments in a bixel image, because adjacent tiles share sides. In each tile's lower left (A) corner we store a 2-bit '*lb*' (for left, bottom) value whose most significant bit is FALSE if the tile's left side is cut apart by a boundary segment, and the LSB applies to bottom side. Combined with three more bits each for boundary point position (x_p, y_p), each bixel's storage cost is typically 8 bits greater than the same image stored as pixels alone.

Bilinear basis: Although much better filters exist for prefiltering and reconstruction (e.g. [MN88]), we use the bilinear basis both for pre-filtering to create bixels from scenes and for the interpolation needed to evaluate a bixel image at any point between its sample points at integer (*x*,*y*) locations. Bilinear interpolation is also widely available in commodity texture-mapping hardware, but bixel bilinear bases are organized into patch functions that depend on nearby boundaries as well as sample values at tile corners as shown in Figure 4 and explained in Section 4.

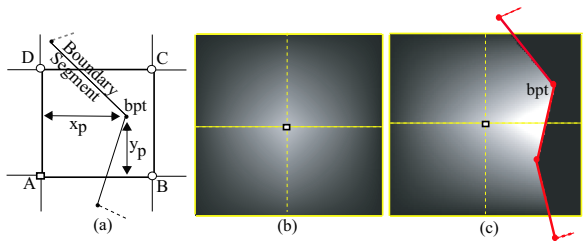

Figure 4: *Tile Details: (a) A tile is a* 1×1 *area with bixel sample points at its corners. It holds only one boundary point, and may be connected to others in adjacent tiles by straight line boundary segments. (b) Reconstruction of a* 4*-tile portion of a bixel image without boundaries: all* 9 *bixel samples are zero except center sample at* 1.0*. (c) With boundaries: patch functions bilinearly interpolate or extrapolate intensities only within their tile regions.*

Tile Regions and Patch Functions: The bixel image reconstruction functions described in Section 4 let us evaluate bixel image intensity values at any point (dx, dy) within any tile ($0 \leq dx, \quad dy \leq 1$). As we will see, boundaries always complicate this task because they split tiles into separate *tile regions*, as shown in Figure 5. Each tile region always includes at least one tile corner (by Rule 2), and we use these corner letters to name each kind of tile region. Each kind of region has a *patch function* $P(dx, dy)$ that specifies the bixel intensity at any position (dx, dy). Symmetry permits us to compute all tile regions from 10 unique patch functions, given in Appendix 10 and explained in Section 4.1.

Complexity: The seemingly arbitrary limits on boundary complexity within a tile were chosen to match the aliasing limits imposed by sample points. Without boundaries, an $M \times N$-tile bixel image holds a maximum of $M \times N$ intensity changes, as it might from sampling the Nyquist frequency sinusoid $cos(\pi x)cos(\pi y)$ shown in Figure 6. If we add a complete, 4-connected boundary point to every tile in this image, it becomes a checkerboard with adjustable boundaries; and every bixel sample point is enclosed in its own region. No allowable boundary point adjustment can create new regions, no more bixel boundary segments can be adopted, and any boundary segment deletions will merge existing regions.

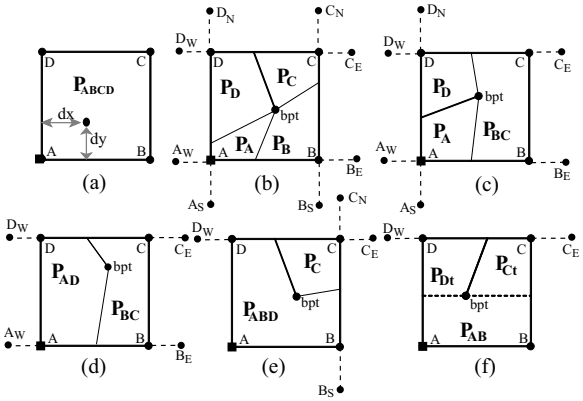

Figure 5: *Six tile configurations that define all patch functions $P()$ needed to interpolate bixel intensities within tile regions. Tile corners are A, B, C and D; neighbors are denoted by a direction subscript for east, west, north and south neighbors, e.g. A_S is the south neighbor of A.*

4. Bixel Reconstruction

Bilinear re-sampling and reconstruction methods are straightforward for pixels, and we can extend them to interpolate bixel images as well if we proceed tile-by-tile and region-by-region. As Figure 5 shows, boundary segments connect boundary points in adjacent tiles, and split each tile into regions. We named each kind of tile region by the letters of the tile corners they contained, and each tile-region has its own "patch function" $P()$, which computes the bixel image intensities at position (dx, dy) within its tile. To evaluate the bixel image at any desired point ($x.dx$, $y.dy$), we must first find the tile (at x, y) and the tile region that contains that point, and then evaluate that tile region's patch function $P()$ at (dx,dy). Though the number of permutations of tile regions is large, by patch re-use and 90° rotations we can reduce all of them to just ten unique patch functions held in the 6 example tiles shown in Figure 5. Each of the 10 patch functions perform bilinear interpolation between a selected combination of neighboring bixel sample points, chosen by the presence or absence of nearby boundary segments. Section 4.1 and Appendix 10 hold further details for each patch function, and we provide C++ source code for bixels at *www.cs.northwestern.edu/ jet/publications.html*.

Goals for this boundary-switched bilinear scheme are to ensure that: (1) bixel intensities interpolate all bixel samples; (2) intensities I match along adjacent tile sides; and (3) gradients ∇I match along adjacent tiles if connectivity constraints allow it. Our library meets these goals for all patch functions except P_{ABD} (Figure 5e) that must average gradients at the D and B corners to cover the region with a single patch function. If we split P_{ABD} into two patch functions that meet along the line between corner A and the tiles' boundary point, then it is possible to satisfy all patch goals, but this two-part patch is more complicated to evaluate.

Patch functions automatically provide reasonable behavior for small closed loops of boundary segments. If the loop encloses only one sample point, then the patch functions for tile regions within the loop are held constant at that one sample point value. Loops that enclose either two or three sample points are linearly shaded by the gradients defined by those samples, and larger loops are shaded bilinearly.

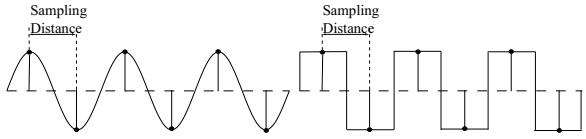

Figure 6: *(a) Reconstructed scan-line from 2D a boundary-free bixel image (e.g. pixels) with the maximum number of intensity reversals; (b) Bixel reconstruction with the maximum possible boundary complexity.*

4.1. Patch Function Details

This section explains how to construct patch functions for tile regions and can be skipped on first reading. We begin with the simplest; with no boundary segments at all, $P_{ABCD}(dx, dy)$ (see Figure 5(a)) describes bixel intensities within a tile region that covers the entire tile, free of any interference from boundaries. Its patch function is a simple bilinear interpolation between tile corner sample values A,

B, C, and D, at position $(0 \le dx, dy \le 1)$ within the tile:

$$P_{ABCD}(x,y) = A(1-dx)(1-dy) + B(dx)(1-dy) + C(dx)(dy) + D(1-dx)(dy). \quad (1)$$

We can rewrite P_{ABCD} in matrix form if we define a row vector V to hold sample values from the tile corners: $V = [A \quad B \quad C \quad D]$; define a column vector X to hold the 2-D polynomial basis: $X = [1 \quad dx \quad dy \quad dxdy]^T$; and define a bilinear coefficient matrix W:

$$W = \begin{bmatrix} 1 & -1 & -1 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix}.$$

Then Equation 1 becomes:

$$P_{ABCD}(dx,dy) = V \cdot W \cdot X. \quad (2)$$

To derive the W matrix, make a matrix X_{corner} whose columns are the X vector evaluated at the tile corner positions for A, B, C, D, use it to replace X in Equation 2, and then the left-hand side must equal the column vector V. Solving for W yields

$$W = X_{corner}^{-1}. \quad (3)$$

Nearly the same method can define patch functions for less complete tile regions. Figure 5(b) shows $P_A(dx,dy)$ for a tile region with only one corner (A). Boundaries separate tile corner A from the other three, forcing us to use other nearby bixel samples to determine the bilinear patch function. Though we will only evaluate the patch function within its tile region, we could easily evaluate $P_A()$ anywhere, including the corners that are unreachable due to intervening boundaries. We call these unreachable corners the *pseudo-corners* for the patch, and their values are marked with an asterisk: they are B^*, C^* and D^* for patch function $P_A(dx,dy)$.

Pseudo-corners make patch finding simpler, because any combination of tile corners and pseudo-corners applied to Equations 2 and 3 will determine the patch. First, we write corner expressions that best satisfy the three bixel interpolating goals listed at the start of this section. Next, we convert them to matrix form and apply Equations 2 and 3 to set the patch. For patch function $P_A(dx,dy)$, the expressions are simple: we make pseudo-corners from A's south and west neighbors (A_S and A_W in Figure 5(b)). If the path from corner A along the straight dotted line to A_W is not cut by a boundary segment, then we use backward-differencing to set the patch's x-derivative; otherwise it is zero. Similar conditions for neighbor A_S sets the y-derivative, and produces these pseudo-corner expressions for $P_A()$:

$$\begin{aligned} B^* &= A + aa_w(A - A_W), \\ D^* &= A + aa_s(A - A_S), \\ C^* &= B^* + D^* - A; \end{aligned} \quad (4)$$

where path bit $aa_s = 1$ if the (A,A_S) path is not cut by a boundary segment, and zero otherwise; path bit $aa_w = 1$ if the (A,A_W) path is intact, and zero otherwise.

Next, put these expressions in matrix form. Define a row vector V' that holds values of the nearby bixel samples that patch function $P_A()$ will use: $V' = [A \quad A_W \quad A_S]$. Also, define a matrix T that transforms V' into a more general V vector that holds the tile's corner or pseudo-corner values: for our $P_A()$ patch function, $V'T = [A \quad B^* \quad C^* \quad D^*]$ and T is determined by Equation 4 above. Then the patch function is:

$$P_A(dx,dy) = V' \cdot T \cdot W \cdot X \quad (5)$$

We can apply this same method with complementary neighbors to derive the $P_B()$, $P_C()$ and $P_D()$ patch functions given in Appendix 10 and shown in Figure 5(b). In each case, the W matrix is fixed, and the T matrix has a simple dependence on connectivity; patch function evaluation is a weighted sum of stored bixel sample values.

Each of the patch functions in Figure 5(b) had no more than two neighboring sample points, limiting us to a planar solution. Tile regions with more corners have more neighboring sample points available, and permit higher-order surface fitting if desired. But for the P_{BC} patch in Figure 5(c) we chose to use the two east neighbors B_E and C_E to set the patch function's two pseudo-corners. We used backwards differencing to estimate the x-derivatives at B and C, and path bits bb_e and cc_e mark intact dotted-line paths from B to B_E and from C to C_E respectively. Patch P_{BC} pseudo-corners are:

$$\begin{aligned} A^* &= B + bb_e(B - B_E), \\ D^* &= C + cc_e(C - C_E); \end{aligned} \quad (6)$$

As before, we build a V' matrix from A^*, B, C and D^*, construct the T matrix from Equation 6, and then matrix multiplications produce the patch function $P_{BC}(dx,dy)$ given in Appendix 10. Repeating this scheme for the remaining three adjacent pairs of tile corners will produce patch functions $P_{AB}()$, $P_{CD}()$ and $P_{AD}()$.

Tile regions that include three corner functions such as $P_{ABD}()$ in Figure 5(e) have proved the most troublesome, because the bilinear function can be over-constrained by tile side derivatives. It is these patch functions that need the most improvement, perhaps by a new scheme that splits them into two or more bilinear patches, or perhaps by applying higher-order surfaces such as quadratics or cubics. To find our patch function, we compute a single pseudo-corner, but assign its value from the average of two forward-differenced estimates:

$$C^* = B + D + A\frac{(dd_w + bb_s - 2)}{2} - D_W\frac{dd_w}{2} - B_S\frac{bb_s}{2}; \quad (7)$$

As before, we only use forward difference estimates taken from intact paths between neighboring sample values. Path bits dd_w and bb_s give connectivity for the D–D_W and B–B_S paths respectively. Note that our V' vector contains five sample values rather than four. This enlarges the T matrix

to size 5×4 in Equation 7 and may complicate hardware implementations.

Finally, a tile with just one boundary segment as seen in Figure 5(f) requires a slightly different approach, because its tile regions are not disjoint. Our solution is simple; we find the tile side cut by the single boundary segment, and construct a parallel joint-line that passes through the boundary point (x_p, y_p). We then construct three separate patch functions. The patch that does not share a tile side with a boundary ($P_{AB}()$ in Figure 5(f)) is unaffected by the boundary segment; thus $P_{AB}() = P_{ABCD}()$. In the other two patch functions, the 't' suffix denotes a single boundary segment cuts the top of the tile. We require P_{Dt} and P_{Ct} to match the $P_{AB}()$ tile value along the joint line.

Unlike all other tiles, we can compute a value bpt at the boundary point location (x_p, y_p):

$$\begin{aligned} bpt = & A(1-x_p)(1-y_p) + B(x_p)(1-y_p) + \\ & C(x_p)(y_p) \qquad\quad + D(1-x_p)(y_p). \end{aligned} \tag{8}$$

We use this boundary-point value along with the usual corners, pseudo-corners, and path bits to specify the patch functions on either side of the boundary segment. For P_{Ct}, we find the pseudo-corner

$$D^* = C + cc_e(C - C_E), \tag{9}$$

and then construct a T matrix that will transform the bixel sample values $V' = [A \quad B \quad C \quad D \quad C_E]$ into $V = [D^* \quad B \quad bpt \quad C]$. A mirror image of this procedure produces the corresponding $P_{Dt}()$ patch function.

4.2. Bixel Resizing and Resampling

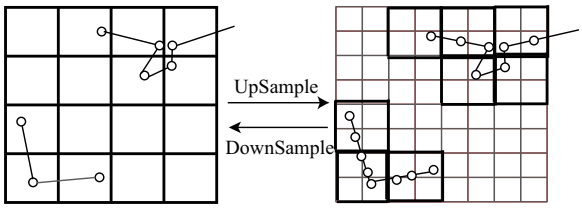

Figure 7: *Upsampling and downsampling of bixels*

Enlarging one bixel image to make another requires resampling of both the image and the boundary geometry, as shown in Figure 7, but is otherwise straightforward. To find the value of each output sample point we find the corresponding position in the input bixel image, find its tile, find the position within that tile, and then evaluate its patch function at the desired position. To make new boundaries, we copy the source boundary point positions to their corresponding positions in the output bixel image, and then replace source image boundary segments by a line-drawing-like process that creates new connected boundary points in any new tiles. Note that small amounts of enlargement (e.g. 1.1) can sometimes require new boundaries that can only be approximated due to bixel boundary placement rules as in Figure 7, lower left.

Reducing the size of a bixel images reveals several important shortcomings. Straightforward minification methods work well for the bixel samples, but as the reduced-size boundaries become too dense to represent by 4-connected boundary segments, we must confront a strangely constrained 2D mesh simplification problem. How should we modify the shrunken boundaries to best fit bixel constraints? What simplifications are most visually acceptable? Even more fundamentally, when should shrunken boundaries that exceed bixel complexity limits be retained or replaced with boundary-free tiles of the same overall intensity and gradient?

5. Antialiasing: Bixel to Pixel Conversion

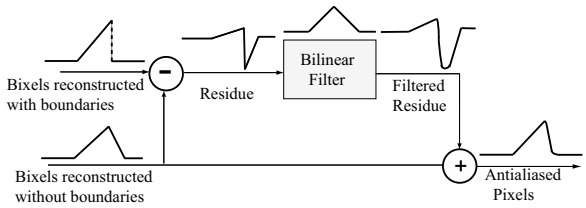

Figure 8: *Anti-aliasing for bixel display*

If displayed as pixels, bixel samples will show aliasing, because their embedded boundaries are discontinuous and infinitely sharp. To remove this aliasing, we need to smooth away only the extra, excessively high-frequency components that these sharp boundaries provide. If a bixel image had no boundaries, its sample values become those of ordinary bilinearly interpolated pixels. We must take care to restrict the smoothing to apply only to the portions of the signal that disappear when the boundaries are removed.

Our method for anti-aliasing bixels for display relies on super-sampling to approximate continuous signals, and is outlined in Figure 8. We first collect a uniform grid of samples from the bixel image. We use multiple samples per tile (typically 4×4), to approximate the piecewise continuous signal that the bixel represent. We then remove (or ignore) all boundaries from the same bixel image and repeat the process. The difference between these two grids of samples represents the 'residue' signal, which holds only the image components that the boundaries introduce into the bixel image. We smooth this residue by bilinear filtering and downsampling, then add it to the original bixel samples to produce anti-aliased display pixels from bixels.

6. Making Bixels from Pixels and Boundaries

Computer graphics renderers can directly construct bixels as their rendered output, but other data sources usually offer

pixels and boundary estimates separately. We cannot simply append boundary segments to pixels to form bixels, because bixel values differ from pixels near boundaries. We need to find the smoothed but unrelated scene values that the boundaries keep separate. For the given boundaries, we want to find the bixel image that, when converted back to pixels, matches the source image.

An iterative approach yielded very good results, as seen in Section 7. We initialize the bixel sample values to match the input pixel values, and set its boundary points and segments from given boundary data. We then convert this tentative bixel image back to pixels, as described in Section 5, and subtract it from the source pixel image to find a bixel error image. We subtract a small fraction of the error image from the bixel sample values, and repeat the process until the error converges to zero.

7. Results

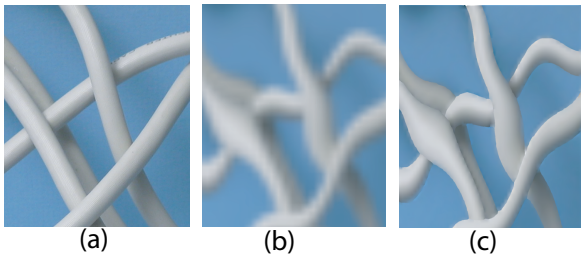

Figure 9: *(a) Pixel-only cable image; (b) pixel-only warping of a) increases blur in enlarged regions; (c) warped bixels preserves sharpness*

As shown in Figures 1, 9, 10, 11 and 12, bixels are applicable to both photographic and synthetically rendered images. For the images in Figure 1 and 10, bixels perform well in preserving most of the sharp features from an extremely low resolution input image. As evident in the bone image of Figure 10, bilinear interpolation for pixel images noticeably blurs details near the silhouettes and boundaries. Similar degradation of visual boundaries is also evident in the Lincoln image (Figure 1) where bilinear interpolation for pixels obscures the nose, eyes and other important features (Note: boundaries shown were hand-selected). Figure 9 demonstrates boundary preservation in bixels for arbitrary nonlinear warps. Figure 11 illustrates nonlinear and quadratically curved boundaries. The bixel interpolation scheme also preserves discontinuities well for medium to high resolution input images. In Figure 12, the bixel interpolation scheme keeps the sharp corners sharp and avoids the blurring visible in the pixel image.

8. Discussion

As Section 2 shows, others have placed discontinuities within sampled images before, but each differs from bixels in important ways. Though Salisbury *et al.* [SALS96] showed excellent NPR results, the reported their method was problematic for general purpose images, because it reduced or distorted gradients near boundaries and used exhaustive connected-path searching for image sampling and reconstruction. Bixels use switched linear filters implemented as patches to avoid these problems. Sen [SCH03] used 2-bit boundary codes similar to bixels, and his upcoming

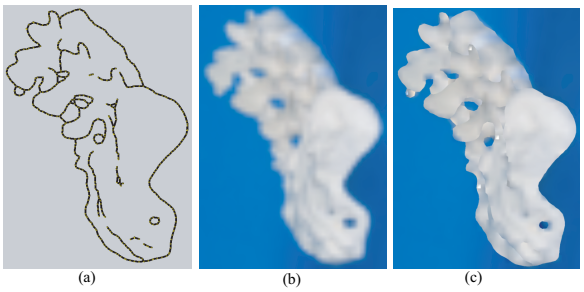

Figure 10: *Bixels preserve sharp boundaries for low resolution input images. (a) Silhouette shapes in just* 65×50 *bixels; (b) Pixels ignore boundaries and blend background and foreground; (c) Bixels' sharp features reveal bone structure. Source image courtesy Ramesh Raskar, MERL.*

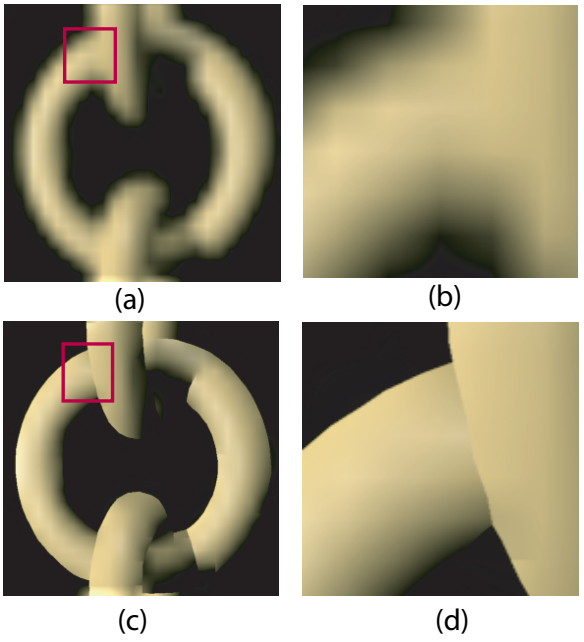

Figure 11: *(a)* 25×25 *pixel ray-traced torus (bilinear); (b) Enlarged red box contents from (a) (bilinear); (c)* 50×50 *bixels (bilinear) preserve contours well. Note: our slight shadow boundary misplacement caused gradient errors; (d) Enlargement preserves sharpness. Source image courtesy Kavita Bala and Bruce Walter, Cornell University.*

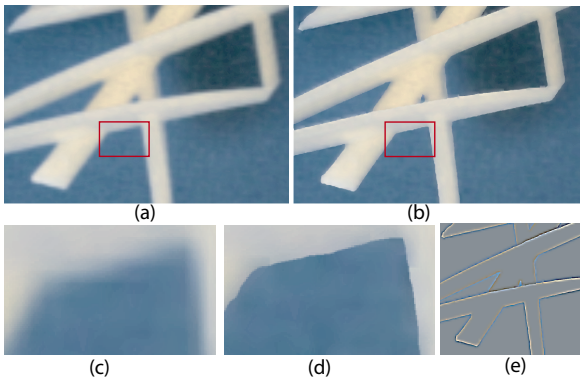

Figure 12: *(a)* 100×100 *pixel plastic part photo (bilinear) (b)* 100×100 *bixels (bilinear) hold both gentle curves and acute boundary angles well (c) pixels destroy small corners, (d)but bixels do not (e) Difference (a)-(b) shows boundary-applied signal.*

work [Sen04] defines nearly the same connectivity rules. His reconstruction filter is simple, fast, and hardware-based, but also reduces intensity gradients near boundaries, a distortion shared with [SALS96, BWG03] and [RBW04]. More centrally, these boundaries are viewed only as an aid to rendering, rather than visually meaningful scene features available in machine-readable form.

We think bixels might apply to a broader range of research topics in computer graphics. For example, image morphing and warping schemes (see [GDCV98]) routinely rely on artificially placed boundaries to adjust images: bixels might permit control based on intrinsic image content. For image compositing, bixels can remove the ambiguity of the 'alpha' channel, which conflates the sub-pixel position of the outline of an object with its transparency. Many authors have noted that remaining difficulties with texture synthesis are often related to poor statistical descriptions of visual boundaries; bixel-based methods might offer better results. Similarly, visual boundaries are fundamental tools to guide non-photorealistic rendering methods; perhaps bixels could couple sketch-like renditions to boundaries linked to 3D shapes.

Bixels have several serious shortcomings. Perhaps the most obvious is the need to both reduce as well as enlarge bixel images, as briefly discussed in Section 4.2. Also, capturing bixel data photographically is currently difficult because visual boundaries are not always marked by well defined intensity changes. Also our conversion method to bixels from pixels and boundaries (Section 6) could be improved; we adjust only bixel sample values and do not refine the boundary positions because it presents a difficult nonlinear optimization problem endangered with local minima. A better solution might assist bixel reconstruction from lower quality source data. Local control of bixel sharpness is also sorely needed; for example Lincoln's nose-tip in Figure 1 should not be infinitely sharp, and we believe that Elder's work [Eld99] offers a good approach to this problem.

The importance of readable visual boundaries in images leading to new approaches to capture them photographically. For example, recent work by Raskar *et. al* [RR04] finds silhouette and self-occlusion boundaries using multiple photographic flash units, and they kindly assisted us by supplying source data for Figure 10 and by processing our photos into the boundary precursors we needed to the create bixel image for Figure 12.

9. Conclusion and Future Work

Bixels offer raster images with embedded visual boundaries that are infinitely sharp, positioned well, and machine readable. They reversibly merge both 2-D image geometry and sampled image data, and unlock the previously fixed relationship between sharpness and the number of image samples. They can describe discontinuities that form arbitrary planar graphs without restriction to closed loops, and can separate arbitrary and variable intensities and gradients. The seem suitable for a wide variety of graphical data sets including images, depth, surface normals, and more. Bixels use bilinear patches for each tile region, and with care they can be rendered as texture-mapped polygonal meshes in OpenGL. Bixels are a good fit to computer graphics because the lighting changes and geometric shapes responsible for most visual boundaries are readily available with high precision. However, improved methods to capture such boundaries photographically may broaden its range. We think bixel images are but one step towards digital images that more directly encode all of the visually-available contents of a scene.

10. Appendix

We list the expressions for vector V' and matrix $T \cdot W$ for the tiles in Equation 5.

$$P_A\colon V' = (A, A_W, A_S);\ T \cdot W = \begin{bmatrix} 1 & aa_w & aa_s & 0 \\ 0 & -aa_w & 0 & 0 \\ 0 & 0 & -aa_s & 0 \end{bmatrix}.$$

$aa_w = 1$ if pixels A and A_W are connected and 0 otherwise; $aa_s = 1$ if pixels A and A_S are connected and 0 otherwise.

$$P_B\colon V' = (B_S, B, B_E);\ T \cdot W = \begin{bmatrix} 0 & 0 & -bb_s & 0 \\ 1+bb_e & -bb_e & bb_s & 0 \\ -bb_e & bb_e & 0 & 0 \end{bmatrix}.$$

$bb_e = 1$ if pixels B and B_E are connected and 0 otherwise; $bb_s = 1$ if pixels B and B_S are connected and 0 otherwise.

$$P_C\colon V' = (C_N, C, C_E);\ T \cdot W = \begin{bmatrix} -cc_n & 0 & cc_n & 0 \\ 1+cc_n+cc_e & -cc_e & -cc_n & 0 \\ -cc_e & cc_e & 0 & 0 \end{bmatrix}.$$

$cc_e = 1$ if pixels C and C_E are connected and 0 otherwise; $cc_n = 1$ if pixels C and C_N are connected and 0 otherwise.

$$P_D\colon V' = (D_W, D_N, D);\ T \cdot W = \begin{bmatrix} 0 & -dd_n & 0 & 0 \\ -dd_n & 0 & dd_n & 0 \\ 1+dd_n & dd_w & -dd_n & 0 \end{bmatrix}.$$

$dd_w = 1$ if pixels D and D_W are connected and 0 otherwise; $dd_n = 1$ if pixels D and D_N are connected and 0 otherwise.

P_{AD}: $V' = (A, A_W, D_W, D); T \cdot W = \begin{bmatrix} 1 & aa_w & -1 & -aa_w \\ 0 & -aa_w & 0 & aa_w \\ 0 & 0 & 0 & -d_w \\ 0 & 0 & 1 & d_w \end{bmatrix}$.

aa_w = 1 if pixels A and A_W are connected and 0 otherwise; dd_w = 1 if pixels D and D_W are connected and 0 otherwise.

P_{BC}: $V' = (B, B_E, C_E, C); T \cdot W = \begin{bmatrix} 1+bb_e & -bb_e & -1-bbe & -bb_e \\ -bb_e & bb_e & bb_e & -bb_e \\ 0 & 0 & -cc_e & cc_e \\ 0 & 0 & 1+cc_e & -cc_e \end{bmatrix}$.

bb_e = 1 if pixels B and B_E are connected and 0 otherwise; cc_e = 1 if pixels C and C_E are connected and 0 otherwise.

P_{ABD}: $V' = (A, B, B_S, D, D_W); T \cdot W = \begin{bmatrix} 1 & -1 & -1 & \frac{dd_w+bb_s}{2} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{bb_s}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -\frac{dd_w}{2} \end{bmatrix}$.

bb_s = 1 if pixels B and B_S are connected and 0 otherwise; dd_w = 1 if pixels D and D_W are connected and 0 otherwise.

P_{Ct}: $V' = (A, B, C, D, C_E)$;

$T \cdot W = \begin{bmatrix} 1 & -1 & -1 & 1 \\ 0 & 1 & 0 & -1 \\ \frac{(1+cc_e)\cdot y_p}{y_p-1} & \frac{(1+cc_e)\cdot y_p}{1-y_p} & \frac{(1+cc_e)}{1-y_p} & \frac{(y_p+cc_e)}{y_p-1} \\ \frac{y_p}{1-y_p} & \frac{y_p}{y_p-1} & \frac{y_p}{y_p-1} & \frac{y_p}{1-y_p} \\ \frac{y_p\cdot cc_e}{1-y_p} & \frac{y_p\cdot cc_e}{y_p-1} & \frac{cc_e}{y_p-1} & \frac{cc_e}{1-y_p} \end{bmatrix}$.

cc_e = 1 if pixels C and C_E are connected and 0 otherwise and the boundary point within a tile is located at (x_p, y_p).

P_{Dt}: $V' = (A, B, C, D, D_W); T \cdot W = \begin{bmatrix} 1 & -1 & -1 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & \frac{y_p}{1-y_p} & 0 & \frac{y_p}{y_p-1} \\ 0 & \frac{(1+dd_w)\cdot y_p}{y_p-1} & 1 & \frac{dd_w+y_p}{1-y_p} \\ 0 & \frac{-y_p\cdot dd_w}{y_p-1} & 0 & \frac{dd_w}{y_p-1} \end{bmatrix}$.

dd_w = 1 if pixels D and D_W are connected and 0 otherwise and the boundary point within a tile is located at (x_p, y_p).

References

[BWG03] BALA K., WALTER B., GREENBERG D. P.: Combining edges and points for interactive high-quality rendering. *ACM Trans. Graph. 22*, 3 (2003), 631–640.

[Can86] CANNY J.: A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell. 8*, 6 (1986), 679–698.

[Car84] CARPENTER L.: The a -buffer, an antialiased hidden surface method. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (1984), ACM Press, pp. 103–108.

[DFRS03] DECARLO D., FINKELSTEIN A., RUSINKIEWICZ S., SANTELLA A.: Suggestive contours for conveying shape. *ACM Transactions on Graphics, Special issue: Proceedings of ACM SIGGRAPH 2003 22*, 3 (July 2003), 848–855.

[DJ03] DEAN JACKSON E.: Scalable vector graphics (svg)1.2: W3c working draft. web document, Nov 2003.

[Dur02] DURAND F.: An invitation to discuss computer depiction. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering* (2002), ACM Press, pp. 111–124.

[Eld99] ELDER J.: Are edges incomplete? *International Journal on Computer Vision* (1999).

[FA91] FREEMAN W. T., ADELSON E. H.: The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence 13*, 9 (1991), 891–906.

[GDCV98] GOMES J., DARSA L., COSTA B., VELHO L.: *Warping and Morphing of Graphical Objects*. Morgan Kaufmann Publishers, 1998.

[GG01] GOOCH B., GOOCH A.: *Non-Photorealistic Rendering*. A. K. Peters, Ltd., 2001.

[Hec92] HECKBERT P.: Discontinuity meshing for radiosity. In *Eurographics Rendering Workshop* (May 1992), pp. 203–216.

[KSE*03] KWATRA V., SCHöDL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph. 22*, 3 (2003), 277–286.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987), ACM Press, pp. 163–169.

[LTG92] LISCHINSKI D., TAMPIERI F., GREENBERG D. P.: Discontinuity meshing for accurate radiosity. *IEEE Comput. Graph. Appl. 12*, 6 (1992), 25–39.

[LV00] LOKOVIC T., VEACH E.: Deep shadow maps. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 385–392.

[MN88] MITCHELL D. P., NETRAVALI A. N.: Reconstruction filters in computergraphics. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (1988), ACM Press, pp. 221–228.

[NRH03] NG R., RAMAMOORTHI R., HANRAHAN P.: All-frequency shadows using non-linear wavelet lighting approximation. *ACM Trans. Graph. 22*, 3 (2003), 376–381.

[OF02] OSHER S. J., FEDKIW R. P.: *Level set methods and dynamic implicit surfaces*. Springer-Verlag, 2002.

[OS75] OPPENHEIM A. V., SCHAFER R. W.: *Digital Signal Processing*. Prentice Hall, Englewood Cliffs, NJ, USA, 1975.

[Pal99] PALMER S. E.: *Vision Science: Photons to Phenomenology*. The MIT Press, Cambridge, Massachusetts, 1999.

[PM90] PERONA P., MALIK J.: Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell. 12*, 7 (1990), 629–639.

[RBW04] RAMANARAYANAN G., BALA K., WALTER B.: Feature-based textures. In *Proceedings of Eurographics Symposium on Rendering* (June 2004), Jensen H. W., Keller A., (Eds.), vol. (to appear).

[RR04] RAMESH RASKAR KAR-HAN TAN R. F. J. Y. M. T.: Non-photorealistic camera: Depth edge detection and stylized rendering using multi-flash imaging. *ACM Transactions on Graphics, special issue on Proceedings of ACM SIGGRAPH Annual Conference 23*, 3 (2004), (to appear).

[SALS96] SALISBURY M., ANDERSON C., LISCHINSKI D., SALESIN D. H.: Scale-dependent reproduction of pen-and-ink illustrations. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), ACM Press, pp. 461–468.

[SCH03] SEN P., CAMMARANO M., HANRAHAN P.: Shadow silhouette maps. *ACM Trans. Graph. 22*, 3 (2003), 521–526.

[Sen04] SEN P.: Silhouette maps for improved texture magnification. In *Proceedings of Eurographics/SIGGRAPH Graphics Hardware* (August 2004), McCool M., Akenine-Moller T., (Eds.), vol. (to appear).

[SGG*00] SANDER P. V., GU X., GORTLER S. J., HOPPE H., SNYDER J.: Silhouette clipping. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 327–334.

[SKS97] SIDDIQI K., KIMIA B. B., SHU C. W.: Geometric shock-capturing ENO schemes for subpixel interpolation, computation, and curve evolution. *Graphical Models and Image Processing 59*, 5 (1997), 278–301.

[SLD92] SALESIN D., LISCHINSKI D., DEROSE T.: Reconstructing illumination functions with selected discontinuities. *Third Eurographics Workshop on Rendering* (May 1992), 99–112.

[WWT*03] 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 (2003), 334–339.

[ZZV*03] ZHANG J., ZHOU K., VELHO L., GUO B., SHUM H.-Y.: Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Trans. Graph. 22*, 3 (2003), 295–302.