EUROGRAPHICS 2005 / M. Alexa and J. Marks
(Guest Editors)

Volume 24 (2005), Number 3

Pinchmaps: textures with customizable discontinuities

Submission id: 1397

Abstract

We introduce a new texture representation that combines standard bi-linearly interpolated samples, for smoothly varying regions, with customizable discontinuities for sharp boundaries between these regions. It consists in a standard signal texture, plus a second texture we call pinchmap, which encodes discontinuities along generally curved lines; this structure is stored in texture memory as a pair of images and is efficiently interpreted on commodity graphic hardware in the fragment shader. We also present a fully automatic way to compute a pinchmap and signal texture pair from an much higher resolution image. We show that the final effect on the screen is a comparable visual quality, for a fraction of the texture storage cost and a very small impact on performance.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Line and Curve Generation

1. Introduction

In general terms, a 2D texture T_s stores a signal function s (e.g. color, normal, alpha value, or other attributes) that is to be applied over a 2D surface. It consists in a regular set of samples (texels) of that signal, that are typically interpolated at rendering times. Naturally, the quality of the result is strictly dependent to the resolution of the texture. High resolution textures are so determinant to achieve better visual result that texture memory is always a resource in short supply, notwithstanding the continuous increase of its availability that we experienced thanks to chipset improvements. Hence the need to increase perceived texture quality by other means than just increasing the number of texels.

One promising direction is to resort to mixed 2D image representations that are more expressive, for typical images, than a direct sampling. In facts, the combination of the infinite precision of vectorial elements (e.g. lines, sharp boundaries) together with the flexibility of sampled texels potentially leads to tremendous decrease of texture memory usage for a comparable visual quality.

However, to be useful in most applications, such a scheme must be efficiently interpreted in graphic hardware. Recently we assisted to been important advancements in this direction, but there is currently no solution that is really feasible to run on commodity hardware at an acceptable price in term of consumed resources. In this paper we present such a solution.

After a focussed, brief analysis of related word, we devote the next four sections to the description the new texture representation (showing its basic underlaying concept, the structure itself, its actual implementation on graphic hardware, and additional effects that can be added, in sections 2, 3, 4 and 5 respectively). Later in section 6 we sketch a fully automatic method to construct an instance of that representation.

1.1. Previous Work

In this section we will address only the few previous research results that, to our knowledge, most closely share our objectives: to embed discontinuities into textures to improve their visual quality - especially when magnified - while keeping texture memory usage low. We refer the reader to the basic and advanced literature for other, conceptually related but technically distant problems (like those involving automatic feature detection, image segmentation, on-the-fly texture synthesis, procedural textures and texture compression).

The approaches presented in [TC04, RBW04, Sen04], and ours too, all share the idea of adopting an image (or texture) representation capable of encoding and displaying features (sharp discontinuities) over images that would otherwise describe smoothly varying values. Another shared characteristic is that the extra information is distributed across the image and stored over a regular grid (of “Bixels”, “feature-based-texture” pixels, “silmap” texel, or the "pinchmap" texels that we are about to propose), so that only a limited number of accesses near the current region will be necessary to locally interpret the image (a necessity, if the algorithm is to be implemented in the GPU, where the number of per fragment texture accesses is severely limited).

In the schema proposed in [TC04], pixels are enhanced to embed sharp boundaries (so becoming "bixels"), and pixel values are not interpolated across such boundaries. Boundaries are defines with linear or quadratic formulas, and can therefore be curved. This work is not designed in its details to be implemented in a programmable fragment shader. Its structure would allow for such an adaptation, but this has not been investigated and many problems would probably arise (the same that we will see shortly).

In [RBW04] a similar schema is proposed. Boundaries are defined as a set of splines. This results in a very expressive representation, but also leads to very complex worst cases that would rule out an implementation on programmable graphic hardware. Authors suggests that the simpler segments should be adopted instead of splines in that scenario, which probably would lead to a solution similar to [Sen04].

[Sen04] (a derivative work of [SCH03]) represents a breakthrough because for the first time this sort of algorithms is really implemented and tested on a programmable fragment shader. To achieve this, complexity of boundaries is kept to a minimum (straight segments). Results are impressive. However, the cost of the technique turns out to be still prohibitive for most applications, requiring a total of eight texture accesses per fragment (five to the texture encoding discontinuities, three to the final signal texture) only to obtain a single texture value (the equivalent of a single texture fetch for a standard texture). This figure can possibly be reduced by some form of texel packing or other similar optimizations, but probably not drastically.

This is a consequence of the approach followed by all the above proposals. To implement boundaries, final texture values are computed by averaging, near such boundaries, not the usual four but three, two or a single texel. Unreachable texels are weighted by zero, and remaining weights are renormalized. This makes the computation of the final texel value very heavy in term of number of accesses. It also creates many different cases (even for very simple boundary primitives), that are difficulty dealt in the fragment shader (extremely ill-equipped for densely branched code).

Additionally, that approach has an important shortfall in terms of quality, because only few pixels are interpolated near boundaries. This means that signal gradient in directions perpendicular to the boundaries will be void, and, worse still, at corners, where only a single texel is "weighted", the color is bound to be constant.

Another common trait is that discontinuities are always defined piecewise, region by region, and within each region, in a way independent from neighbors. As we will discuss later, this ultimately results in the necessity to perform more accesses to the discontinuity encoding texture (as in facts turns out to be the case).

As we will see shortly, we approach the problem from a totally different direction, leading to a solution that uses only an extra texture access (other than the one to the final texture), simplifies the per-fragment computations, improves the visual quality, and naturally supports a number of additional features - including curved boundaries, inexpensive anti-aliasing, smoothly starting discontinuities, optional solid lines, and others.

2. Concepts and Preliminaries

Assume the original texture signal s is defined over a 2D squared domain $\mathbb{T}^2 = 0,N \times 0,N$ for some $N = 2^n, n \in \mathbb{N}$. In T_s, suppose texels are located at each integer position in $\mathbb{T}^2$.

The process of scan converting a textured polygon on screen will produce a set of fragments each with its corresponding region of $\mathbb{T}^2$. When that covers multiple samples of T_s, the problem to combine them into a single value can be efficiently solved by various forms of pre-filtering (MIP-mapping). When, on the contrary, the produced fragment corresponds to a region of $\mathbb{T}^2$ with an area smaller than one, then a magnifying filter f_M is needed. The function f_M, defined over $\mathbb{T}^2$, returns for any given position a value that is some combination of the samples of T_s.

Fetching the closest texel of T_s leads to severe aliasing artifacts. A common solution to alleviate them is to use

$$f_M(u,v) = f_b(u,v), \quad (u,v) \in \mathbb{T}^2$$

where f_b is bilinear (first order) interpolation between the four closest samples of T_s. This solution is so widely applied that is hard-wired on any modern GPU. Bilinear interpolation is ideal when the signal s to be represented is smooth; on the contrary when s presents 0-order discontinuities it leads to exceedingly blurred visual results.

The bilinear interpolation f_b is a continuous function defined piecewise as follows: $\mathbb{T}^2$ is implicitly subdivided into $N \times N$ unit squares, and inside each square f_b is defined as the 1st order bilinear interpolation of the four texels at the corner of that square. We will refer to these squares as *fexel*s (from "four texels"). Each fexel has four corner texels, and adjacent fexels share two corner texels.

2.1. Main Idea

Wherever the signal to be represented presents sharp (0-order) discontinuities, there are some fexels which contains an unwanted smooth transition between the texels that were sampled on either side of that discontinuity. We call those fexels *undesired* fexels. Fexels that are not undesired are *visible* .

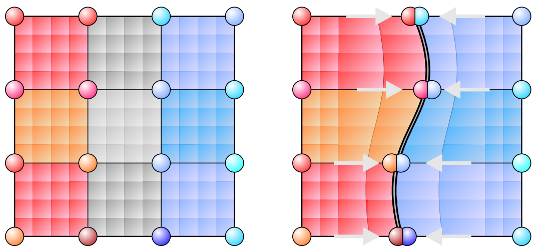

Figure 1: *The concept behind a pinching operation. On the left: a 4×4 closeup of a standard texture (in this case, a color texture). The 16 texels are shown as colored spheres. A* fexel *is the space where 4 texels are bilinear interpolated. Here fexels are separated by a solid line and shown with a superimposed regular grid just to show how the fexel space is warped during the pinching. Fexels that interpolate between similar texels are color-coded with shades of blue or red.* Undesired *fexels, i.e. those that interpolate across very different texels, are color-coded with grays. Right, the undesired fexels have been "pinched" creating a sharp discontinuity between the red and the blue regions.*

The idea is to perturb the locations at which bilinear interpolation is computed so that, intuitively, undesired fexels are shrunk to a line creating a discontinuity in the final result (see Figure 1); in practice, we ensure that area inside undesired fexels will never be accessed. We call this process *pinching*. As magnification filter f_M we therefore use the function:

$$f_M(u,v) = f_b((u,v) + p(u,v)),\ (u,v) \in T_s$$

where p is called the *pinch* function. In locations away from discontinuities, p is valued $(0,0)$ and the magnification filter becomes a standard bilinear interpolation. At discontinuities, p will present a step discontinuity as well.

In all cases, a single bi-linearly interpolated final texture access from the signal texture T_s is performed per fragment. The advantages of this basic choice are manyfold:

- efficiency: bi-linearly interpolated texture accesses are hardware optimized, making best use of on-chip texture RAM bandwidth;
- non-branched code: we always have a single texture texture access to T_s, and that texture access is performed in a non-branched part of the fragment shader;
- visual quality: all visible textured area will still be a part of a legal fexel, i.e. every fetched texture value is interpolated between four texels; this means that the gradient of f_M in any direction is never bounded to be zero;
- additional effects: manipulating the above formula, it is easy to obtain several additional effects, including antialiasing (see sec. 5.1), smooth beginning of sharp boundaries (see sec. 5.2), and others.

Note also that no texel of T_s is wasted, as every texel will still affect an area of T_s; rather we conceal some fexels, that is, regions *between* the texels.

The vector function p implicitly determines: which fexels are pinched, the pinching directions (gray arrows in Fig. 1), and the lines into which pinched fexels are collapsed.

The function p is encoded in an auxiliary texture T_p, the *pinchmap*, that is pre-computed and paired with the main signal texture T_s. Before rendering T_p is loaded on the graphic card as an additional texture. To limit use of GPU RAM T_p should be as compact as possible; also, to limit consumption of on-card texture bandwidth, $p(u,v)$ should be computed using the least number of accesses to T_p, ideally a single one.

One natural choice would be to define p piecewise, by subdividing $\mathbb{T}^2$ into as many pieces as there are texels in T_p and then separately storing each piece in a corresponding texel of T_p, encoding it with a configuration index and a limited number of parameters. However, in order to enforce continuity of p across adjacent texels, one would need to either perform additional accesses to neighbor texels in T_p, or at least replicate some data from neighbor texels inside each texel of T_p. It the former case we go against our objective to limit the number of texture accesses; in the latter case it would become exceedingly difficult to pack all the required information in a single 4-channel texel.

To bypass this problem we designed a scheme where p is computed using a single bi-linearly interpolated texture access to the pinchmap T_p, and $p(u,v)$ is computed starting from the four recovered values (one per channel).

It could be argued that a bilinearly interpolated texture access has a cost equivalent to four direct texture accesses. However, as we noticed, bilinear texture access is so optimized and hard-wired that in practice it affect performance and limits bandwidth just as much as a single closest-value texture access. For example, in all programmable fragment shader GPU architectures the limit of texture accesses is independent on whether the accesses are performed with or without bilinear interpolation.

The scheme is described in detail later in Section 3, but here we briefly list some of its properties:

- the scheme is capable of encoding discontinuities along lines that are, in general, curved;
- the size in texel of the pinchmap T_p can be flexibly chosen to be equal or smaller to the one of s (see Section 5.4).

The resulting per-fragment algorithm to process a fragment with associated texture coordinates (u,v) is conceptually as follows (see Section 4.2 later for a more detailed description):

1. fetch texel t_p at pos. (u,v) from pinchmap T_p;
2. compute pinch-function $p(u,v)$ using t_p (and u and v);
3. fetch final texel t_s at pos. $(u,v) + p(u,v)$ from signal texture T_s;

4. process t_s normally.

Both texture accesses are bi-linearly interpolated. The last step is the same as any other 2D texture mapping application; for example it can consist in a verbatim copy of t_s to the current pixel (if T_s stored a pre-shaded color), or in a shading of T_s (if T_s stored normal values) and so on. The first two steps are detailed in section 3.

2.2. Pinchmaps and mipmapping

Although the pinchmap perturbation is designed for magnification filters, it produces final texture coordinates that are valid for all MIP-map levels. This means that the same algorithm sketched above works regardless of the magnification level (differently from other approaches, we do not need to identify it in the fragment shader). Simply, MIP-map levels can be pre-computed for T_s when loading it and stored on the card as usual; during rendering the texture access in step 3 can be performed with trilinear interpolation.

Clearly, the first texture access (the one to the pinchmap) needs to be performed with simple bilinear interpolation and without MIP-mapping.

3. Pinchmaps

In this section we detail how a proper pinch function p can be stored in and recovered from a pinchmap texture T_p. For illustration purposes, we first tackle a 1D case.

3.1. One dimensional case

In a one-dimension analogue, a fexel is the segment between two consecutive texels in a one-dimensional set of samples. Fexels delimited by texels that should not be interpolated are undesired , just like before. Note that two consecutive fexels cannot both be undesired , otherwise the texel they share would not be part of any visible fexel.

Undesired fexels will be collapsed into a point γ located somewhere inside it, by expanding the two adjacent fexels on its left and right side (which are both visible ones). In particular, we choose to expand only the closest halves of the two adjacent fexels. In this way, the two furthest halves of these fexel are left unchanged, and can be expanded over the possibly undesired fexel on the opposite side, if needed.

We define a local one dimensional pinch function p^{1D}_{γ} : $\mathbb{R} \rightarrow [-1..+1]$ that is parameterized with γ and is used to perturb the texture location in order to pinch away a single undesired fexel. The final texture access for a texel with texture coordinate k will be $k + p^{1D}_{\gamma}(k)$.

Note that, in order to avoid visual artifacts, we must make sure that $k + p^{1D}_{\gamma}(k)$ is strictly monotonically increasing with k. This translates in the constraint $\partial p^{1D}_{\gamma}(k)/\partial k > -1$.

Let us describe p^{1D}_{γ} in a reference system where the origin is in the middle of the 1D fexel to be pinched away (see figure 2). Consequently the locations of the two texel delimiting the fexel will be at -0.5 and at 0.5. The function p^{1D}_{γ} is non-zero only inside the interval $(-1..+1)$ (as we want to affect only the two halves of the adjacent fexels), and moreover it must be zero in ± 1 to ensure continuity. The parameter γ, which is the position where the undesired fexel is collapsed, typically ranges in $[-0.5..+0.5]$.

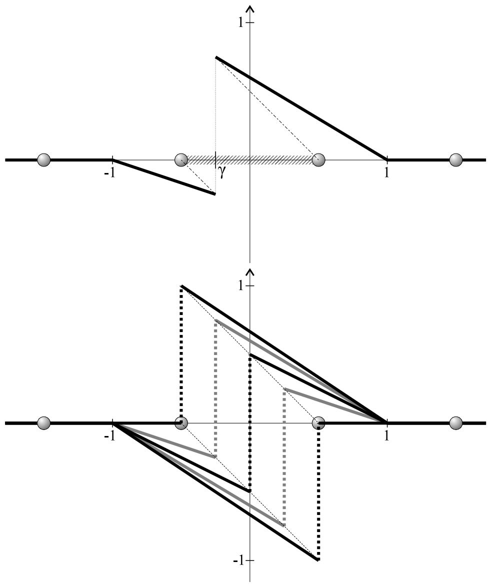

Figure 2: *Above: the one dimensional pinch function p^{1D}_{γ} for a given γ. The horizontal axis is centered in the middle of the fexel to be pinched (hatched with diagonal lines). Texels are shown as gray balls. Below: the function for $\gamma = -0.5, -0.25, 0, +0.25, +0.5$.*

Since we want to “pinch” both delimiter texels into position γ, we need that $p(\gamma^-) = h - \gamma$ and $p(\gamma^+) = h + \gamma$, where h is the value 0.5 (we are using h as a parameter because later we will need to change its value, see Sec. 5.4). By interpolating linearly between these fixed values at -1, $+1$, γ^- and γ^+ we get

$$p^{1D}_{\gamma}(k) = \begin{cases} 0 & k \leq -1 \\ \frac{(1+k)(h+\gamma)}{\gamma+1} & -1 < k \leq \gamma \\ \frac{(1-k)(h-\gamma)}{\gamma-1} & \gamma < k \leq 1 \\ 0 & k > 1 \end{cases} \tag{1}$$

Note that the constraint on the derivative of p^{1D}_{γ} is satisfied for any $\gamma \in (-1..+1)$.

In summary, the function p^{1D}_{γ} is such that $k + p^{1D}_{\gamma}(k)$, with $k \in \mathbb{R}$ is never in $(-0.5..+0.5)$, but will assume any other value for some k. The interval $(-0.5..+0.5)$, which corresponds to an undesired fexel, is effectively “pinched away”,

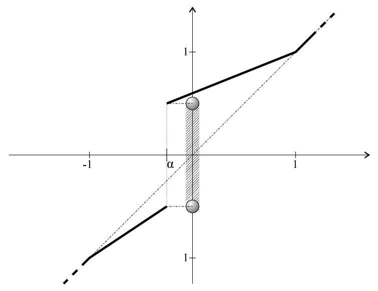

Figure 3: *The plot of the function $p^{1D}_\gamma(k)+k$, for the same parameter γ as shown in Picture 2, above.*

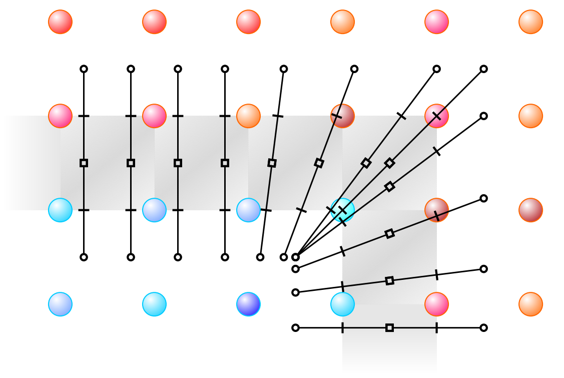

Figure 4: *An discrete subset of the segments along which the function p^{1D}_γ (see Figure 2) is to be applied. In each shown segment, the midpoint ($k=0$) is identified by a square, the extreme points ($k=\pm1$) by a circle, and the points at position $k=\pm0.5$ by a small crossing line. Undesired fexels are grayed (note that they correspond to the points at positions $-0.5<k<+0.5$). For clarity we do not show the pinching positions γ, defined for each segment and forming a discontinuity line.*

meaning that that area of the one-dimensional texture will never be accesses (see Figure 3).

3.2. Two dimensions

Getting back to the two dimensional case, we will define a proper continuous set of 2D segments over the texture space over which p^{1D}_γ is applied. Each segment is identified by its midpoint (u_m,v_m) and its direction $\|(u_d,v_d)\|$, with $\|(u_d,v_d)\|_\infty=1$, and it is parameterized as

$$(u_m,v_m)+k\cdot(u_d,v_d)\ k\in[-1,+1]$$

. We also assign to each segment a position γ that identifies where, over that segment, the discontinuity is to appear. Over each segment, the function p is defined as

$$p((u_m,v_m)+k\cdot(u_d,v_d))=p^{1D}_\gamma(k)\cdot(u_d,v_d) \qquad (2)$$

The midpoint of the segments are in the middle of undesired fexels, and their direction are such that the segment position for $k=\pm0.5$ is on the frontier of between visible and undesired fexels (see figure 4). This way, the image of $(u,v)+p(u,v)$ corresponds exactly to the union of visible fexel; in other words, all and only the undesired fexels will be “pinched away”, and the final texture will never be accessed at point inside them.

3.3. Constraints on 2D

Just as in one dimension we could not have two consecutive pinched fexels, we have similar consistency constrains in two dimensions.

First of all, the combination where a group of 2×2 adjacent fexels are all pinched does not make sense. If that was the case, the signal texel shared by the four fexels would belong to no visible fexel.

Another combination to be ruled out is the one where a texel is at the same time a corner of two opposite undesired fexels, and two opposite visible fexels. If that was the case, the texel in question would be involved in two different incompatible pinch operations pushing it in opposite directions.

Any other combination is legal (see Figure 5 for some examples).

4. Implementation

4.1. Pinchmap structure

In order to apply equation (2) to the texture position of each fragment, we need to use the values u_d, v_d, k and γ of that formula. We store these quantities through the pinchmap T_p. A different channel of T_p is used for each of the four parameters: the texel values $(u^T_d,v^T_d,k^T,\gamma^T)$ are stored in the (r,g,b,α) channels of T_p respectively.

The value of each channel of T_p ranges from -1 to $+1$, and since (r,g,b,α) values natively range in $[0..1]$, a remapping $M(x)=2x-1$ is applied in the fragment shader to all channels of any texel of T_p just after fetching (some care must be taken to be able to represent the value 0 precisely. For example, if the textures has an 8-bit precision per component, then the range $[-1,+1]$ must be represented in the texture with the interval $[0..254]$, not $[0..255]$, so that the value 0 can be correctly represented by the value 127).

As we mentioned, the pinchmap T_p will be accessed in the fragment shader with bilinear interpolation (therefore in this section it will be useful refer both to fexels of the pinchmap T_p and to fexels of signal-texture T_s. When we refer to the former will be always specify *pinchmap* fexel). A single texture access to the pinchmap will return the quadruple of interpolated values $(u^T_d,v^T_d,k^T,\gamma^T)$.

We adopt a schema where we set a pinchmap texel in

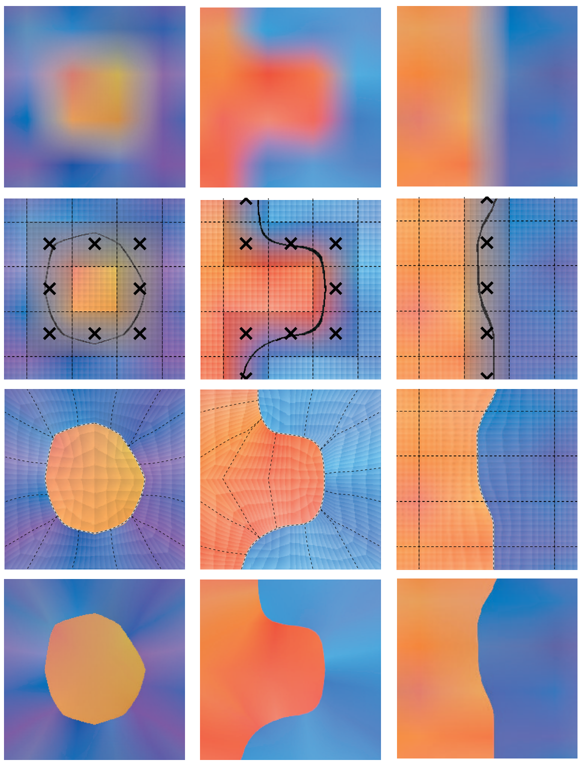

Figure 5: *Examples of basic pinching operations, over a minimal* 4×4 *signal texture texture. Each column shows a different combination. Top row: standard bilinear interpolation, with no pinching. Second row: fexels are shown before pinching. A* 8×8 *subgrid is superimposed, and undesired fexels are darkened and identified with a black cross. The discontinuity line (randomly chosen) is also shown in black. Third row: undesired fexels are collapsed to that line, and neighbors fexels are expanded over them. Bottom row: final visual result (actual snapshot).*

T_p in the center of each fexel of the signal texture T_s. In other words, the two textures are reciprocally displaced by 0.5 in both directions (see Figure 6). In this way, it is easier to obtain the 1-order discontinuities that the channel k must present in $k = \pm 1$ (compare Figure 4), as they now naturally appear at the border between pinchmap fexels.

We refer to the pinchmap texels corresponding to undesired fexels of T_s (that will be pinched) with the term *active* texels.

The main channel of the pinchmap T_p is the channel k^T, which records the position inside the belonging segment. It is set to zero in active texels, and ± 1 elsewhere.

Since $p_\gamma^{1D}(k)$ is valued 0 for $k = \pm 1$, no pinch will be performed in zones where k is constantly ± 1. Therefore the texels of the pinchmap T_p are subdivided into a number of zones separated by a line of active texels, and in each zone

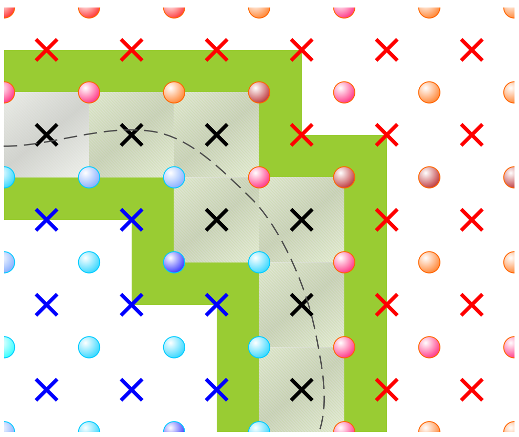

Figure 6: *A detail of a signal- (in this case, color-) texture* T_s *with its pinchmap texture* T_p *superimposed. Texels of* T_s *are shown as colored balls, while texels of* T_p *are sown as crosses. There is one texels of* T_p *for each fexel of* T_s*, so the two grids are displaced by half a texel size. Undesired fexels of* T_s *(grayed areas) correspond to active texels of* T_p*, shown as black crosses, where the k channel is zero. The other texels of* T_p *have a k channel with a value of either minus one (red crosses) or one (blue crosses). The pinch-function is nonzero only in zones colored green or gray, so the pinch affects only these regions. In particular, the green region will expand over the gray region, covering it (in this way the gray region will be pinched into a discontinuity line - here dotted).*

we have a constant $k = \pm 1$. As these zones are necessarily closed, this may at first seem to severely limit the the number of possible pinchmap configurations: more precisely, we need that any pinched fexel belongs to a closed chains of pinched fexels, meaning that the discontinuity line in the final result will need to be always closed. However this is not a problem, as pinching can be locally disabled even in active texels by setting their u_d and v_d channels to zero (see Section 5.3 later).

Each channel of the pinchmap require some implementation consideration.

Channel k**:** As we have seen in Section 3.2, we need the value of k to be linear along all the the segments (and consequently to be equal to ± 0.5 correpsonding to the border between undesired and visible fexels). Even if the value of k^T is set to k in all pinchmap texels, the bi-linearly interpolated value inside a pinchmap fexel is in some case not linear. This happens in a pinchmap fexel that has exactly one or exactly three active pinchmap texels as its corners (see figure 7).

Lucky it is easy to correct the interpolated value k^T and recover linear k, knowing the current texture position over T_p and the signs of u_d^T, v_d^T.

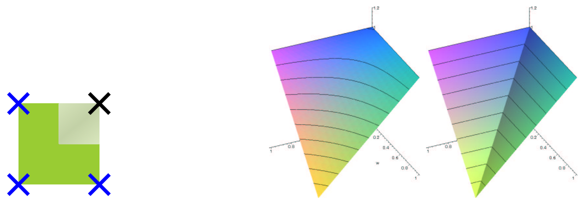

Figure 7: *Left: a single pinchmap fexel from the pinchmap shown in Figure 6. This particular fexel has the k channel set to 0 in one (active) corner texel (in black), and the other three corner texels with a k set to +1 (in blue). This is one case where the result of the bilinear interpolation of the k^T channel (plotted in the middle left) is not equal to the linear signal k that we need (plotted in the right). The latter is linear along the segments defined in Section 3.2, the former is not.*

Channels u_d and v_d: We store a zero value for u_d^T and v_d^T for all non-active in texels, and we store the proper segment orientation only to active pinchmap texels. In this way all non-active pinchmap texels within the same zone delimited by active texels share the same values in the u_d^T, v_d^T and k^T (namely, 0, 0 and ± 1 respectively), independently from neighbors.

This makes it trivial to deal with the case when a non-active pinchmap texel is adjacent to multiple active texels, but poses a problem: the value of (u_d^T, v_d^T) will be the interpolation of the needed values (u_d, v_d) and the value $(0,0)$ that we defined at inactive texels.

However, (u_d, v_d) can be recovered from (u_d^T, v_d^T) simply normalizing the latter (with respect to the infinity norm, that is, dividing it by $\max(|u_d^T|, |v_d^T|)$).

Better yet, given the structure of the channels u_d^T, v_d^T and k^T of our pinchmap, we can predict $\max(|u_d^T|, |v_d^T|) = (1 - |k|)$. We prefer the latter formulation for reasons that will be explained in Section 5.2.

Channel γ: The channel γ is the sole responsible to determine the exact location of discontinuity line -where the undesired fexels will be collapsed- within the zones affected by the pinch (colored in green and gray in In figure 6).

In facts, that discontinuity appears in points where $k = \gamma$ (see Figure 2). We define $\gamma^T \leftarrow \gamma + (k^T - k)$, so that the above equality is equivalent to the easier to control $k^T = \gamma^T$.

Note that both the member of the last equality are simply bilinear interpolated values that we control by setting the texels of T_p. Note also that at exact texel positions of T_p, since $k^T = k$, we have that $\gamma^T = \gamma$.

All we need to do is to set to γ^T channel of each texels appropriate values so that the intersection of that channel with the channel k^T (which is determined by the combination of active and inactive texels) produces the required discontinuity lines (see Section 6 for an illustration of how to do this in an automatic way). Note that such intersections are in general along curved lines.

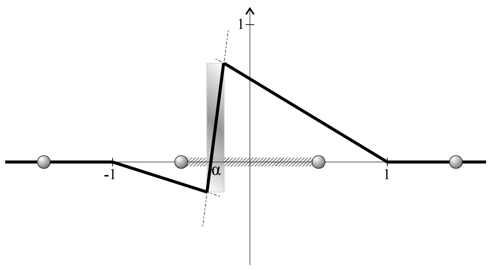

Figure 8: *The plot of the 1 Dimensional Anti-Aliased pinch function $p_\gamma^{1D,AA}(k)$, for $\gamma = -0.25$. In the horizontal axis the anti-aliased part i is grayed.*

4.2. Fragment Program

The complete fragment program, for a fragment with initial texture coordinates (u, v), is therefore:

1. Fetch bilinearly interpolated values (r, g, b, α) from Pinchmap Tp at $(u + h, v + h)$ (with bilinear interpolation);
2. Remap $(u_d^T, v_d^T, k^T, \gamma^T) \leftarrow M(r, g, b, \alpha)$;
3. Compute the "rectified" value k form k^T;
4. Normalize direction: $(u_d, v_d) \leftarrow (u_d^T, v_d^T)/(1 - |k|)$;
5. Compute $\gamma \leftarrow \gamma^T + (k - k^T)$;
6. Pinch the texture coordinate:
   $(u', v') \leftarrow (u, v) + p_\gamma^{1D}(k) \cdot (u_d, v_d)$
7. Fetch final signal value from (u', v')
   (with bilinear interpolation and mip-mapping).
8. Process fetched signal value as usual.

The entire program, including all the extensions that we are going to illustrate in the next section, has been implemented in less than 50 ARB fragment program instructions (the most expensive parts currently being Steps 3 and 5).

Numerical analysis shows that the algorithm is stable even when the divisor in step 4 is arbitrarily close to zero (in which case, also the final displacement will be close to zero). Of course, care must be taken to avoid a direct division by zero - for example, by adding a small constant.

5. Additional Features

In this section we show some additional feature that can be added, with a small or null impact of efficiency, to the pinchmap algorithm.

5.1. Anti-aliasing

It is easy to add an anti-aliasing on screen over the 0th order discontinuities that we introduce in the texture. Conceptually, the idea is simply to perform only incomplete pinch

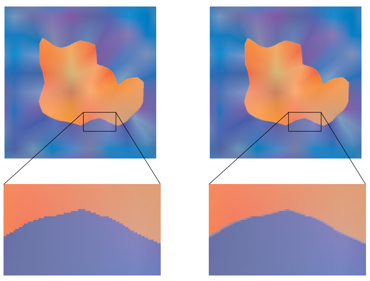

Figure 9: *Actual screenshots featuring a simple textured quad, showing a random shape defined by a very small (8x8) pinchmap and a rgb-colormap (as the signal texture) of the same size. Below: a post-processed close-up to show individual screen pixels. Left and right: results with and without anti-aliasing.*

operation, so that undesired fexels are shrunk to a very small but positive area rather than to a line. This effectively results in an anti-aliasing, as undesired fexels are exactly the region where the values from the two sides of the 0-order discontinuity are interpolated together.

The only thing that we need to change is the function p^{1D}_{γ}, which is to be substituted in equation (2) with its anti-aliased version $p^{1D,AA}_{\gamma}$. The function $p^{1D,AA}_{\gamma}$, which is continuous, is defined as:

$$p^{1D,AA}_{\gamma}(k) = \begin{cases} \beta(k-\gamma) & \text{if } (k<\gamma) \otimes (\beta(k-\gamma) < p^{1D}_{\gamma}(k)) \\ p^{1D}_{\gamma}(k) & \text{otherwise} \end{cases} \tag{3}$$

where the $\otimes$ symbol denotes the logical exclusive-or, and β, with $\beta \gg 1$, is the anti-aliasing parameter, intuitively denoting the speed of texture coordinate over the undesired fexels. The meaning of the formula is that the value of p^{1D}_{γ} is overridden in proximity if its discontinuity by a steep, but not instantaneous, connecting function (see Figure 8).

The anti-aliasing parameter β determines the width of the region where the undesired fexel will be squeezed. It is easy to see that that region will be $1/\beta$ texel sizes long. Ideally, for a perfect anti-aliasing, should be equal to one screen pixel. One possibility is to actually use the per-fragment texture-speed to adequately set β for every texel. Alternatively, that can be approximated with a global parameter β for each textured object.

The function $p^{1D,AA}_{\gamma}$ is just an approximation of the ideal anti-aliasing function (in facts, contrarily to the ideal case, the anti-aliased regions on the left and on the right of γ are in general different in size). However, the function is very simple to compute and it is good enough for all practical purposes. For an example, see figure 9.

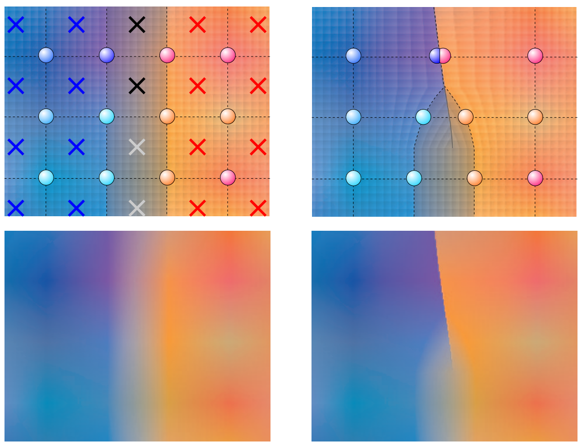

Figure 10: *Pinching can be locally turned off, and discontinuity lines can start smoothly. Top left: an image covered by a 4×3 subset of a signal texture T_s (circles) with a pinchmap T_p (crosses). Each fexel is shown with a superimposed 8x8 grid, and undesired fexels are darkened. Active pinch texels are shown as black or white crosses: white ones have a pinch strength p_{str} of zero, black ones of one. As a result (top right) only the top undesired fexel is pinched away, while the bottom one is not. The two middle ones show a smooth transition between these two states. Bottom: actual screenshots of the application without (left) and with (right) pinching: the discontinuity line starts smoothly.*

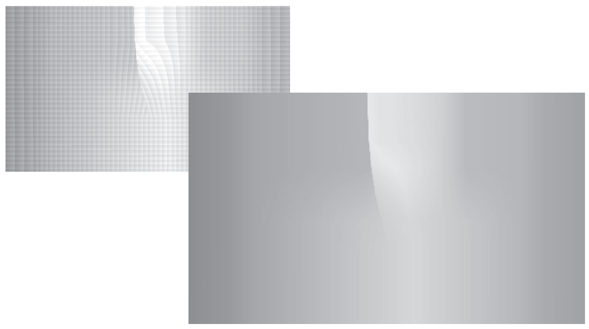

Figure 11: *Here, the signal texture is a normal-map encoding a curved surface with a crease. The pinch strength is modulated from zero to one over the space of three fexels in order to make a crease start smoothly. Left: fexels are shown with a superimposed 8x8 grid, right: final result (actual screenshot).*

5.2. Smoothly starting discontinuities

An important feature of our system is the ability to have a smooth spatial transition between pinched and not pinched regions. Intuitively the effect is obtained by only "half-pinching" a given undesired fexel, leaving one of its side un-pinched while pinching the other (see figure 10).

The concept is straightforward. We assign to each seg-

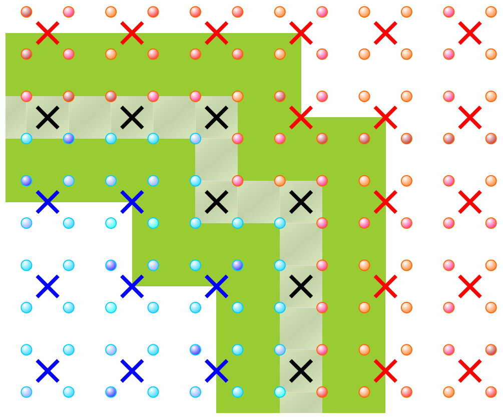

Figure 12: *A detail of a signal-texture T_s and paired with a pinchmap T_p in case that the latter has a resolution that is an half of the one of the former. Notice the disposition of the texels of the two textures. For a legend of symbols, see the caption of Figure 6.*

ment *pinch strength* $p_{str} \in [0,1]$, and we use it to weight the pinch function p_γ^{1D} in equation 2, which becomes:

$$p((u_m, v_m) + k \cdot (u_d, v_d)) = p_\gamma^{1D}(k) \cdot p_{str} \cdot (u_d, v_d) \qquad (4)$$

This is implemented in a simple way, by encoding p_{str} in the (u_d^T, v_d^T) channels of each texels. In active texels we store, as (u_d^T, v_d^T), the values $(p_{str} \cdot u_d, p_{str} \cdot v_d)$. Step 4 of the fragment program in Section 4.2 will now rightly compute $p_{str} \cdot (u_d, v_d)$ rather than (u_d, v_d)

The ability to make boundaries smoothly start over several texels is especially useful in cases when the signal texture T_s represents a normal field: the 0th order discontinuity lines are in this cases creases, and in many cases (like, for example, in an automobile chassis) it is required that they start very gently (see Fig. 11).

5.3. Open boundaries

Another important application of the above is when the pinch strength p_{str} is uniformly set to zero in a larger area, in which case no pinching at all occurs in that area (see Figures 11 and 10). The ability to locally turn off pinching let us have non closed discontinuity lines across the texture. A non closed discontinuity line is (correctly) bound to start smoothly.

5.4. Different resolutions for the pinchmap and signal textures

Until now we assumed the pinchmap T_p and the signal texture T_s had the same resolution. However, in general, the resolution of the pinchmap can be 2^K (with $K \geq 0$) times smaller than the signal texture. When $K > 0$, the texels of T_s must be grouped in cluster sized $2^K \times 2^K$, that will be placed on one same side of the discontinuity. The displacement of T_p with respect to T_s is, in the general case, $2^{(-K-1)}$ (see Figure 12).

In the fragment shader, the only difference is that the illegal fexel to be pinched are now smaller with respect to the size of one fexel of T_p. Therefore the function p_γ^{1D} need to be smaller in magnitude. The value of the parameter h (in equation (1) and in Step 1 of the fragment code in Section 4.2) is in the general case $2^{(-K-1)}$ (see Figure 12).

On the contrary, it is impossible to have a signal textures smaller than the pinchmap, as that would make the function p_γ^{1D} too steep (with the derivative smaller than -1).

Within this limit, the two resolutions can be chosen independently. Intuitively, the resolution of the pinchmap determines the complexity of the discontinuities - the higher it is, the more complex and dense the discontinuity lines can be; the resolution of the signal texture determines instead the possible complexity of the color-map in the smoothly varying zones.

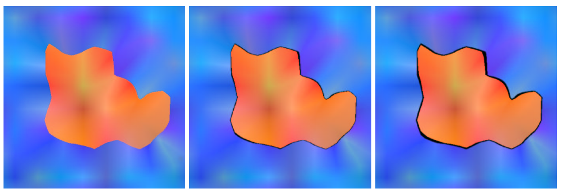

Figure 13: *Actual screenshots featuring the same data as figure 9. Left: pinched result, without solid lines. Middle and right: solid black lines of different thickness are added in the way described in Section 5.5.*

5.5. Solid lines.

Another easy-to-obtain effect consists in adding thick lines of a constant color in correspondence of the discontinuities lines, an effect that can be useful, for example, when the signal texture contains color information and we are targeting a non-realistic rendering (see Figure 13). This is applicable only when all discontinuity lines are closed.

To get this effect, it is sufficient to apply the anti-aliased pinch function $p_\gamma^{1D,AA}$, as defined in 3, and detect when the 1st of its cases is applied. In this anti-aliased would normally take effect. Instead, we simply overwrite the current fragment color with a globally defined constant color. The global parameter β of $p_\gamma^{1D,AA}$ determines lines thickness (which however will not be perfectly constant, due to the variation of the length of diagonal segments).

In some application it could be appropriate to store line thickness and color in each vertex of the model, or in an additional texture, in order to have them varying over the surface.

6. Automatic pinchmap creation tool

So far we described the way pinchmap-enanched textures work.

In this section we detail how one can build a pinchmap and a signal texture to represent a given signal s in a fully automatic way.

6.1. Inputs and outputs

The **input** of this process is a description of the original signal s, for example a vector-based representation of an image, or a procedurally defined image, or a very high-resolution raster image I_{HiRes}. In our implementation we adopted the latter approach because of its better flexibility (if the signal if originally defined in any other way, in can be sampled into a hi-res raster image in a pre-preprocessing stage, with arbitrary precision).

Moreover, we use I_{HiRes} images with a extra alpha (transparency) channel, that is used by the final tool user to specify where the signal discontinuities must appear: the intended meaning of this channel is as follows: wherever two regions of I_{HiRes} are separated in the alpha channel by a steep jump from zero to one or from one to zero, it means a sharp border between the two corresponding texture regions is required. On the contrary, where the alpha channel present no such jump, the signal in the final result will be blended.

For example, the user can provide as input a $rgb\alpha$ color image consisting of a foreground shape (where alpha is set to one), silhouetted against the background (where alpha is set to zero), to obtain a final result where that shape appears separated from the background by a sharp boundary, and the color varies smoothly both internally and externally of the shape (see for example Figure 14).

Finally, we are assuming that the resolution res_p of the required pinchmap T_p and the resolution res_s of signal texture T_s are also given as input. Clearly the two resolutions are supposed to be much lower than the resolution of I_{HiRes}.

The **output** of the process is a pinchmap and a signal-texture at the asked resolutions that can be used to approximate signal originally stored in I_{HiRes}: in particular, the discontinuity encoded in the pinchmap will mimic the boundaries discretely specified via the alpha channel as described above; the signal texture will store the signal specified in the other channels of I_{HiRes} (whether they represented color, normal, transparency, etc).

Once built, the two output textures can be stored and later used independently from the original image I_{HiRes} that was used to create them.

6.2. Algorithm

The algorithm consist in the following phases:

1. down-sample the alpha channel α of I_{HiRes} into a bit-mask I_{Layer} of resolution res_p;
2. enforce consistency constraints on the bit-mask I_{Layer};
3. build u_d^T v_d^T and k^T channels of T_p from the bitmask;
4. optimize the channel γ^T channels of T_p to match $I_{HiRes}.\alpha$
5. fill T_s from the other channels copying values from the proper locations of I_{HiRes};

In the first four step the pinchmap T_p is constructed.

Step one consists in a brutal down-sampling of α channel of I_{HiRes}. We build a temporary I_{Layer} bit-mask with a bit for each fexel of the final pinchmap. All texels of the corresponding area of I_{HiRes} are checked against the threshold 0.5, and the I_{Layer} bit is snapped to 0 or to 1 according to the majority of the results (we keep track of the magnitude of the rounding for the next step).

In the second step we enforce the constrains described in Section 3.3. Problematic areas are located, and then an iterative local search is performed: in each step we find and apply the cheapest move that diminishes the global number of broken constraints. A basic move consist in assigning a previously mixed 2×2 patch of I_{Layer} to an one or a zero (overwriting their previous values), and its cost is computed as the worsening of the total rounding error).

In step 3 this I_{Layer} will be used to identify active and inactive pinchmap texels, and to consequently build the u_d^T v_d^T and k^T channels of each texel of T_p. This is straightforward: each texel is build in the intersection of 4 pixel of I_{Layer}, and is fully determined by their value.

Next, the γ^T channel is optimized in step 4. Our objective is to assign a value to the γ^T component of each texel (that is either active or around an active texel), so that we maximize the number of matching pixels of I_{HiRes}. A pixel is considered matching when its α value is bigger than 0.5 if and only if in the corresponding point of T_p the bi-linearly interpolated value of γ^T is bigger than the interpolated value of k^T (see Section 4.1). We do this using a randomized simulated annealing approach.

Now the pinchmap T_p is ready, all we have to do is to fill the signal-texture T_s. To do that we first compute the inverse g^{-1} of the function $g(u,v) = (u,v) + p(u,v)$ (by using a discreet approximation, simulating the pinchmap behavior as defined by T_p). Then to each texel of T_s, corresponding to the position (i,j) on T_p, we assign a signal value that is a combination of the texels of I_{HiRes} around $g(u,v)$.

When we do that, we make not mix in T_s texels of I_{HiRes} that belong to different regions. That is, we must be sure to combine only texels that have all $\alpha > 0.5$ (if in position (u,v) of T_p where $\gamma > k$) or $\alpha < 0.5$ (otherwise).

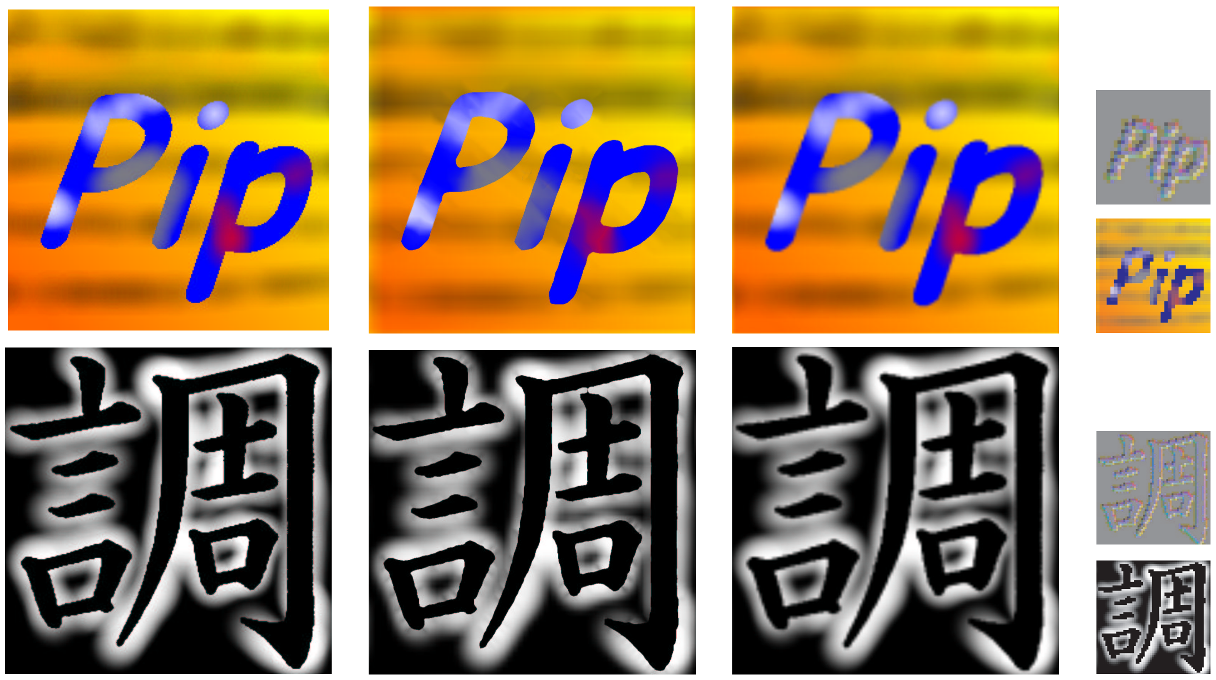

Figure 14: *Some results of the automatic pinchmap creation tool. Left: original starting image (alpha channel, which in both cases separates foreground from background, is not shown). Second column: the result pinchmapped texture (actual screenshot of the HW implementation): above both signal and pinchmap texture are sized* 32×32*, below* 64×64*. Third column: for comparison, a 128x128 anti-aliased down-sampling of the original image. Last column: the signal map and the color-map.*

6.3. Results

We tested our method in a variety of input of different sizes and with different parameters. Some results in visible in Figure 14.

The computation times are contained, keeping in mind that this is just a preprocessing phase. The forth phase, being an iterative search over a very large domain, is the most expensive one and takes proportionally almost all of the time. However, thanks to the optimization, total times are in the order of a few seconds for a 32×32 pinchmap mimicking an original texture of 1024×1024.

More importantly, once built the pinchmap/signal-texture pair can be rendered in real time (as was predictable from the fragment program instruction counts), also leaving many per-fragment resources (texture accesses and ALU instructions) untapped for any further computation.

7. Conclusion

We presented a texture representation that uses an auxiliary pinchmap to describe and embed custom discontinuities along generally curved lines over standard bi-linearly interpolated 2D textures. We showed how the obtained results are visually comparable to those usually obtainable only at a cost of a severely larger texture memory usage.

In this worth mentioning here the two basic ingredient of our approach. First, the very pinching (fexel hiding) approach means that we always resort to a single final bilinearly interpolated to the last texture access, with positive effects both on visual quality (every point interpolation of four points) and of course on efficiency. Second, the preceding texture access to the auxiliary pinchmap is also bilinear interpolated, again improving efficiency (both reducing texture accesses and leading to basically single case algorithms), and also unlocking simple solutions to deal with anti-aliasing, mip-mapping, smoothly starting discontinuities, resolution differences between pinchmaps and signal textures, and so on.

The system is designed for minimal performance impact, maximal quality and features during the rendering phase; the cost is that it becomes difficult to design a pinchmap that delivers the wanted discontinuities at the wanted locations. This is why the work presented here would have been limitedly useful without an automatic procedure to perform that task. Even considering that, some limits remain: pinchmaps only approximate the required image where more than two sharp boundaries generates from a point. Moreover, sharp angles of discontinuity lines are sometimes difficult to achieve.

As a final consideration it should be noted that the resulting system -consisting in a module to create pinchmap/signalmap pairs, and a fragment program to display them- can be used as a black box by final user.

References

[RBW04] RAMANARAYANAN G., BALA K., WALTER B.: Feature-based textures. In *Proc. of the Eurographics Symposium on Rendering* (June 2004), Eurographics Association. 1, 2

[SCH03] SEN P., CAMMARANO M., HANRAHAN P.: Shadow silhouette maps. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003) 22*, 3 (July 2003), 521–526. 2

[Sen04] SEN P.: Silhouette maps for improved texture magnification. In *Proc. of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (August 2004), Eurographics Association. 1, 2

[TC04] TUMBLIN J., CHOUDHURY P.: Bixels: Picture samples with sharp embedded boundaries. In *Proc. of the Eurographics Symposium on Rendering* (June 2004), Eurographics Association. 1, 2