Scented Sliders for Procedural Textures

Anass Lasram[1] Sylvain Lefebvre[1] Cyrille Damez[2]

[1]ALICE/INRIA [2]ALLEGORITHMIC

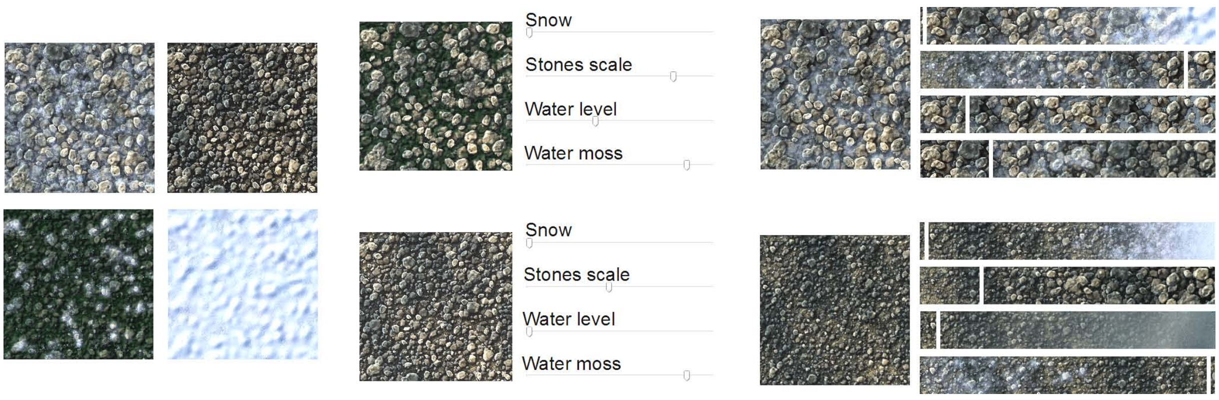

Figure 1: *Left: Four random variations of the same procedural texture. Middle: Two textures with standard sliders controlling their appearances. Right: Two textures with our visual slider previews controlling their appearances.*

Abstract

Procedural textures often expose a set of parameters controlling their final appearance. This lets end users tune the final look and feel, typically through a set of sliders. However, it is difficult to predict the changes introduced by a given slider, especially as sliders interact in non–trivial ways.
We augment the sliders controlling parameters with visual previews revealing the changes that will be introduced upon manipulation. These previews are constantly refreshed to reflect changes with respect to the current settings. The main challenge is to generate the visual sliders in a very limited pixel space and at an interactive rate. This is done by synthesizing the visual slider from a small set of patches ordered in accordance with the slider. These patches are chosen so as to reveal as much as possible the visual variations induced by the slider. The selection and ordering are achieved by using the seam–carving algorithm to carve patches with low visual impact. The obtained patches are then stitched together using patch-based texture synthesis to form the final visual slider.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

Procedural texture design software, such as *Substance* from *Allegorithmic*, *Genetica* from *Spiral Graphics* or *Filter Forge* let technical artists carefully design complex procedures generating a large variety of textures. These procedures are controlled by a set of parameters exposed to the user. The most widely spread interface for this task is a set of sliders with the name of the parameters next to them. Such interfaces have proven efficient for setting parameters in a computer graphics context [KP10]. However, they are difficult to use for someone discovering a new procedural texture: Contrary to other applications, the set of parameters and their meaning varies strongly from one texture to another (see Figure 1 or Figure 2). Since there is no visual clue – besides the parameter name – of what each slider exactly does, it is hard to predict and understand the influence of each parameter without trying a large number of different settings. In addition, dozens of parameters are exposed and many of them change the effect of each other. This quickly becomes a bottleneck for users exploring the possibilities of-

DOI: 10.2312/conf/EG2012/short/045-048

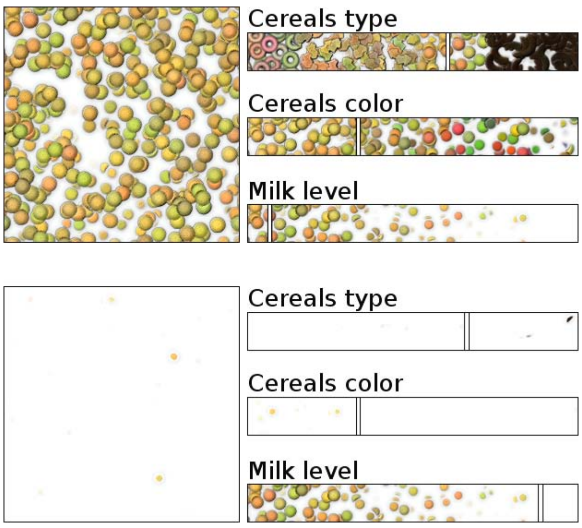

Figure 2: *Two settings of the 'Cereals' texture with visual sliders control.* Top: *The 'Milk level' parameter has a low value.* Bottom: *The 'Milk level' parameter has a high value, masking the effect of other parameters.*

fered by different textures. For instance, the parameters 'Cereals type' and 'Cereals color' in Figures 2 would not have any effect if the parameter 'Milk level' is set to its highest value. Our slider previews solve this problem by illustrating the effect of each parameter *for the current settings*. In particular, our sliders dynamically adapt to user interactions by refreshing all the slider previews if one parameter changes. This way, our visual slider previews make the user aware that the 'Milk level' is the only parameter that could affect the settings of Figures 2, bottom.

Designing such visual slider previews presents a number of challenges:

- The slider previews should indicate in an obvious manner all the *changes* that will occur in the texture when the slider is manipulated.
- The pixel space is very limited: We cannot afford to display a large image below each slider, or the navigation from one slider to the next would quickly become tedious. This is especially problematic when the slider impacts small features in the texture.
- The continuous refresh of slider previews imposes a fast synthesis algorithm.

We address these three points in Section 3.

2. Previous work

In a human–computer–interaction context techniques have been developed to improve sliders efficiency by augmenting the sliders with visual clues such as histograms or color bars [Eic94, WHA07]. Video tapestry [BGSF10] replaces the slider used to navigate in a video by a single image strip that summarizes the content of the video. In a recent work, Kerr and Pellacini [KP10] compare different approaches for the task of selecting material shader parameters. The study concludes that, under interactive feedback, sliders perform best in particular when precise adjustments are required. Image summarization techniques [AS07, SCSI08] seek to summarize the content of a large image in a smaller pixel-space. These techniques, mainly targeted at photographs, change the layout of features in the image in order to preserve the most important elements. We exploit these approaches to create our visual sliders.

3. Slider previews

Overview To synthesize visual slider previews following the requirements set in Section 1, we need to concentrate in a small pixel area the important information: in this case the parts of the image that are effectively changed by the slider. We understand this as an importance–driven image–relayout problem: Starting from an image showing all changes we would like to discard non–essential information that remains unchanged when moving the slider. This is done by the following steps:

- A large image is formed by tiling the texture a number of times horizontally. The large image is then divided into a grid of patches (Figure 3, ***(b)***). Each column of patches reflects the appearances of one setting of parameters. The parameter corresponding to the slider is increasing from left to right while others keep their current values. This means that the leftmost column of patches reflects the appearances produced if the parameter is at its lowest value. The rightmost column reflects the appearances produced if the parameter is at its highest value.
- An importance map is computed (Figure 3, ***(c)***) by giving a score to each patch depending on its variance during slider manipulation. Patches that vary very often will get high scores.
- The set of patches is reduced by removing patches with low scores (Figure 3, ***(d)***). To limit distortions and preserve the layout of patches, seam–carving is applied on the importance map to remove patches rather than pixels.
- The remaining patches (Figure 3, ***(e)***) are stitched together using patch–based texture synthesis (Figure 3, ***(f)***).

Generating the grid of patches We note $V \in [0,1]^N$ the parameter vector of size N describing the current state of the texture. There are N sliders and we are interested in synthesizing the slider s, $s \in \{1..N\}$. The values in V will be fixed with the exception of $V[s]$ that varies linearly from 0(left) to 1(right). We construct a grid of patches G of size $W_G \times H_G$ in which every element $G[x,y]$, $(x,y) \in \{1..W_G\} \times \{1..H_G\}$, is a patch of size $W_P \times H_P$. The patch at $G[x,y]$ is contained in the texture T_x that has a size of $W_T \times H_T$ and parameters v such as $v[i] = V[i]\ if\ i \neq s$ and $v[s] = \frac{x-1}{W_G-1}$ otherwise. The upper–left corner of this patch is located at coordinates

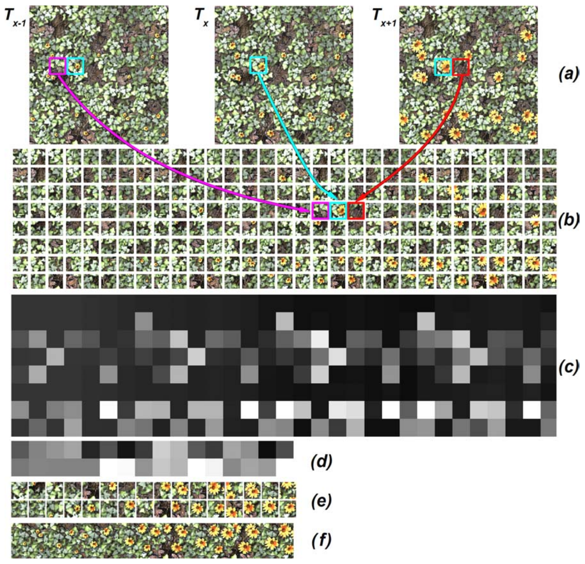

Figure 3: *Algorithm overview: **(a)** Texture samples from which patches are extracted. **(b)** Initial grid G of patches. **(c)** Per-patch importance map I. This map will be carved to remove patches with low scores. **(d)** Carved map I. Bright pixels correspond to patches with high scores. **(e)** The remaining patches. Notice how the flowers, which are the most impacted, dominate the patches. **(f)** Resulting slider after the synthesis step.*

$((x \times W_P)\%W_T\ ,\ (y \times H_P)\%H_T))$ in T_x. Figure 3 ***(a)*** and ***(b)*** highlights the grid creation.

Importance map The importance map I gives a score to every patch in G (Figure 3, ***(c)***). This score is high for patches with high visual impact. We note $I[x,y]$ the score of patch $p = G[x,y]$. We note (x_0, y_0) the upper–left coordinates of patch $p = G[x,y]$ and stored in texture T_x. The score $I[x,y]$ of patch p is computed as follows:

$$I[x,y] = \sum_{t=1}^{W_G} \frac{e^{-\frac{||x-t||^2}{2\sigma^2}}}{1+\sum_{i=x_0}^{x_0+W_P}\sum_{j=y_0}^{y_0+H_P} ||T_x[i,j]-T_t[i,j]||^2}$$

The denominator corresponds to a per–pixel difference between the patch p contained in texture T_x and the patch having the same location as p but contained in another texture T_t. As an example, in Figure 3, ***(a)***, the blue patch in texture T_x is compared to the two blue patches in T_{x-1} and T_{x+1}. The numerator controls the contribution of each texture T_t in the score. Textures with very different parameters will contribute less. These contributions can be further customized using the scalar $\sigma \neq 0$. A small σ^2 emphasizes the possible variations when little changes occur to the slider cursor. If σ^2 goes to infinity then all textures T_t will have a same contribution making the score independent from the cursor position. We typically set $\sigma = 1$ and, in order to speed up the algorithm, we ignore textures with very low contribution (the numerator is very low).

Patch carving We shrink the grid of patches G with seam–carving [AS07], removing vertical/horizontal seams of patches. Carving at the patch–level rather than pixel–level avoids texture distortion but produces discontinuities along the boundaries of the remaining patches. The carving is done simultaneously on G and I according to the scores contained in I. The shrunk version of the map I is shown Figure 3, ***(d)***. It mostly contains patches with high scores.
In principle, non–linear relayout may occur due to the carving. However in practice we never noticed a bias strong enough to justify remapping the parameters.

Synthesizing After carving, we hide abrupt and discontinuous changes of appearance between patches by optimizing their boundaries. Starting from the shrunk version of G (Figure 3, ***(e)***), we find an optimal boundary with graph–cut [KSE*03]. To further reduce potential visible seams, we allow for small changes in the patch positioning and repeat graph–cut optimization a few times (typically four times). The result is a continuous preview strip, with an emphasis on the part of the texture impacted by the slider Figure 3, ***(f)***.

4. Results

Figures 4 shows the benefit of our patch summarization method. In this figure, the most varying elements of the texture (the green stars) represent small features that are far from each other. Sliders (a) and (c) are generated with a naive algorithm that starts from a large slider preview having the same width as the final slider and the same height as the procedural texture. It then crops the horizontal region that maximizes the overall variation. Sliders (b) and (d) are generated using our method. These sliders reveal more variation and give a better insight to the user.

Figure 5 shows an example containing texture and color parameters. User interactions are shown in the accompanying video. Generating one slider for this texture takes on average 151 milliseconds. This timing is dominated by the generation of the 16 textures required to build the slider.

Figure 6 shows another example of sliders for a more complex texture. In this case, an average of 355 milliseconds is required to generate one slider.

5. Limitations

Photographs or textures exhibiting global structural changes are not handled well by our approach. The synthesizer fails to preserve the structure as shown Figure 7, top. One possible work–around is to show these particular parameters as thumbnails. We included this possibility in our sliders by selecting high variance regions as thumbnails and ordering them on the slider Figure 7, bottom.

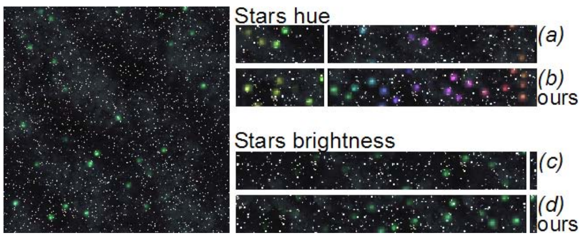

Figure 4: *A procedural texture with two different types of sliders controlling parameters.* (a) and (c): *Sliders generated by cropping the best horizontal strip from a larger initial slider.* (b) and (d): *Sliders generated by our method.*

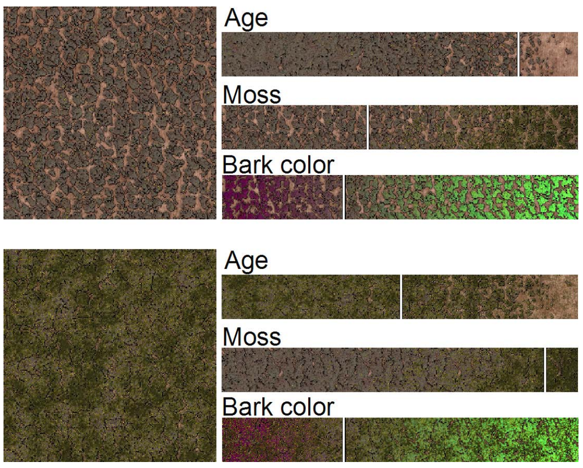

Figure 5: *Two different settings of the 'bark' texture with visual sliders controlling parameters.*

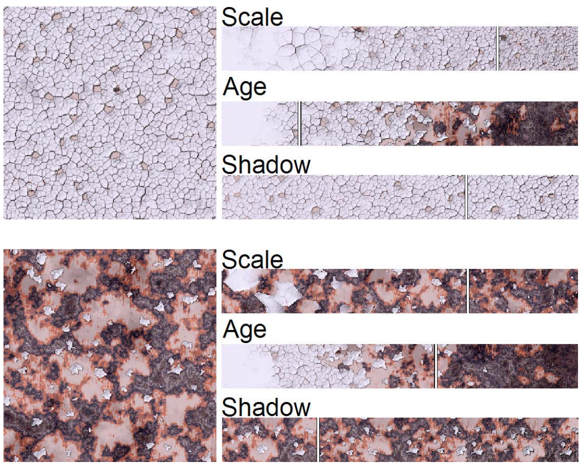

Figure 6: *Two different settings of the 'rotten wall' texture with visual sliders controlling parameters.*

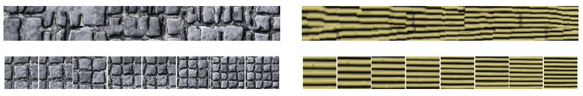

Figure 7: Top: *Failure cases : our approach does not capture global, sudden structural changes in the textures.* Bottom: *Slider thumbnails constitute a possible solution for structural parameters.*

6. Conclusion

Procedural textures tend to have parameters exhibiting unpredictable effects. In this paper a visually driven slider interface for procedural textures has been described to ease parameter selection. Our algorithm generates and analyses several procedural textures at interactive rates. This allowed the creation of dynamic visual sliders that give users more control over the result.

Our sliders are not only useful for final users, but also during the design of the texture. Most procedural textures are obtained by assembling image filters, between tens to hundreds of them. Each filter exposes a number of parameters (e.g. blur intensity, blending alpha value, emboss direction). Our slider previews help quickly revealing the possible effects to the designer.

Acknowledgments
We thank Allegorithmic's texture artists for producing the examples shown in this paper. This work was supported by the ANR SIMILAR-CITIES 2008-COORD-021-01.

References

[AS07] Avidan S., Shamir A.: Seam carving for content-aware image resizing. *Transactions on Graphics* (2007). 2, 3

[BGSF10] Barnes C., Goldman D. B., Shechtman E., Finkelstein A.: Video tapestries with continuous temporal zoom. *Transactions on Graphics* (2010). 2

[Eic94] Eick S.: Data visualization sliders. In *ACM symposium on User interface software and technology* (1994). 2

[KP10] Kerr W. B., Pellacini F.: Toward evaluating material design interface paradigms for novice users. *Transactions on Graphics* (2010). 1, 2

[KSE*03] Kwatra V., Schödl A., Essa I., Turk G., Bobick A.: Graphcut textures: Image and video synthesis using graph cuts. *Transactions on Graphics* (2003). 3

[SCSI08] Simakov D., Caspi Y., Shechtman E., Irani M.: Summarizing visual data using bidirectional similarity. 2

[WHA07] Willett W., Heer J., Agrawala M.: Scented widgets: Improving navigation cues with embedded visualizations. *IEEE Transactions on Visualization and Computer Graphics* (2007). 2