Dynamic Solid Textures for Real-Time Coherent Stylization

Pierre Bénard, Adrien Bousseau, Joëlle Thollot

▶ **To cite this version:**

Pierre Bénard, Adrien Bousseau, Joëlle Thollot. Dynamic Solid Textures for Real-Time Coherent Stylization. I3D 2009 - Symposium on Interactive 3D graphics and games, Feb 2009, Boston, MA, United States. pp.121-127, 10.1145/1507149.1507169 . inria-00345835v4

HAL Id: inria-00345835

https://inria.hal.science/inria-00345835v4

Submitted on 6 Aug 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Author manuscript, published in "Symposium on Interactive 3D Graphics and Games (I3D) (2009) 121-127"
DOI : 10.1145/1507149.1507169

Dynamic Solid Textures for Real-Time Coherent Stylization

Pierre Bénard* Adrien Bousseau† Joëlle Thollot‡

Grenoble Universities and CNRS, LJK INRIA Grenoble Rhône-Alpes

Figure 1: *Dynamic solid textures (on the left) serve as a basis to create a large range of temporally coherent styles. From left to right: a complex scene (538k tris) is rendered in real-time in a watercolor style, a collage style and a binary style.*

inria-00345835, version 3 - 20 Jul 2011

Abstract

Stylized rendering methods, which aim at depicting 3D scenes with 2D marks such as pigments or strokes, are often faced with *temporal coherence* issues when applied to dynamic scenes. These issues arise from the difficulty of having to satisfy two contrary goals: ensuring that the style marks follow 3D motions while preserving their 2D appearance. In this paper we describe a new texture based method for real-time temporally coherent stylization called *dynamic textures*. A dynamic texture is a standard texture mapped on the object and enriched with an infinite zoom mechanism. This simple and fast mechanism maintains quasi-constant size and density of texture elements in screen space for any distance from the camera. We show that these dynamic textures can be used in many stylization techniques, enforcing the 2D appearance of the style marks while preserving the accurate 3D motion of the depicted objects.

Although our infinite zoom technique can be used with both 2D or 3D textures, we focus in this paper on the 3D case (dynamic solid textures) which avoids the need for complex parameterizations of 3D surfaces. This makes dynamic textures easy to integrate in existing rendering pipelines with almost no loss in performance, as demonstrated by our implementation in a game rendering engine.

CR Categories: I.3.7 [Computing Methodologies]: Computer Graphics—Three-Dimensional Graphics and Realism;

Keywords: Non-photorealistic rendering, temporal coherence, real-time rendering, stylization, infinite zoom

*e-mail: pierreg.benard@inrialpes.fr
†e-mail: adrien.bousseau@imag.fr
‡e-mail: joelle.thollot@imag.fr

1 Introduction

Many non-photorealistic rendering approaches aim at depicting 3D scenes with styles that are traditionally produced on 2D media like paper. The main difficulty suffered by these methods is *temporal coherence* when stylizing dynamic scenes. This problem arises from the contrary goals of depicting a 3D motion (induced by the camera or the 3D objects of the scene) while preserving the 2D characteristics inherent to any style marks (pigments, strokes, etc). Achieving these goals without introducing visual artifacts implies the concurrent fulfilment of three constraints. First, the style marks should have a constant size and density in the image in order to preserve the 2D appearance of the medium. Second, the style marks should follow the motion of the 3D objects they depict to avoid the sliding of the style features over the 3D scene (*shower door effect*) [Meier 1996]. Finally, a sufficient temporal continuity between adjacent frames is required to avoid popping and flickering. Many solutions have been proposed to provide a trade-off between these three constraints [Gooch and Gooch 2001; Strothotte and Schlechtweg 2002]. Unfortunately, the integration of these methods in existing rendering pipelines (such as game engines) remains challenging due to their computation cost and specialized data structures.

In this paper we describe *dynamic textures*, a method that facilitates the integration of temporally coherent stylization in real-time rendering pipelines. Dynamic textures address the specific problem of temporal coherence for color region stylization, leaving contour stylization for future work. Our method uses textures as simple data structures to represent style marks. This makes our approach especially well suited to media with characteristic textures (eg. watercolor, charcoal, stippling), while ensuring real-time performances due to the optimized texture management of modern graphic cards.

Central to our technique is an object space *infinite zoom* mechanism that guarantees a quasi-constant size and density of the texture elements in screen space for any distance from the camera. This simple mechanism preserves most of the 2D appearance of the medium supported by the texture while maintaining a strong temporal coherence during animation. The infinite zoom illusion can be applied to both 2D and 3D textures. However, we present our approach for 3D textures (the dynamic solid textures) which alleviate the need for complex computation or manual definition of 2D parameterizations over the 3D surfaces. These solid textures can be produced either procedurally or by synthesis from a 2D exemplar. In order to

inria-00345835, version 3 - 20 Jul 2011

demonstrate the effectiveness of our approach, we integrated it into the OGRE[1] game rendering engine. The performances measured in this environment indicate the low impact of the proposed method on framerate, even for complex 3D scenes. The different styles implemented in this prototype illustrate the simplicity and efficiency of this approach for real-time non photorealistic rendering.

2 Previous Work

In order to solve the contradiction between a 2D style and a 3D motion, Meier [1996] proposes in her seminal work on painterly rendering to decorrelate the appearance of style marks from their motion. In her approach, style marks (paint strokes in her case) are drawn with constant size billboards which preserve their 2D appearance. Each mark is then attached to a 3D anchor point on the 3D object that it depicts. Although this approach has been extended to numerous styles (painterly [Daniels 1999; Vanderhaeghe et al. 2007], stippling [Pastor et al. 2003], watercolor [Bousseau et al. 2006]), the data structure required to manage the anchor points and the expensive rendering of each individual style element makes this family of methods not well-suited for real-time rendering engines.

The individual management of style marks can be avoided by grouping the style marks in textures. The *Dynamic Canvas* method [Cunzi et al. 2003] applies a 2D paper texture over the screen in order to stylize a 3D environment during a real-time walkthrough. This approach reproduces 3D transformations of the camera with 2D transformations of the texture. Translations in depth of the camera are mimicked with an infinite zoom mechanism that preserves a quasi-constant size of the texture in screen space. Although this approach provides a convincing trade-off between the 3D motion of the camera and the 2D appearance of the paper, it is limited to navigation of static scenes with a restricted set of camera motions. Moreover, the image space mechanisms proposed by Cunzi et al. models the scene as a single plane which leads to sliding artifacts for strong parallax. Coconu et al. [2006] and Breslav et al. [2007] adopt an approach similar to Dynamic Canvas by applying a stylization texture in screen space on each object of the scene. In these methods, the projected 3D transformations of the objects are approximated by their closest 2D transformations. Because the textures are only transformed in screen space, these methods well-preserve the 2D appearance of style marks. On the other hand, the approximation of a 3D transformation by a 2D transformation can lead to sliding effects for extreme 3D motions. A novel approach proposed recently by Han et al. [2008] introduces an infinite zoom mechanism. This 2D multiscale texture synthesis algorithm generates texture elements on the fly during the zoom. The zoom illusion produced by this method is more accurate than the one of Cunzi et al. but its computing cost limits its integration in real-time rendering pipelines. In this paper we extend the Dynamic Canvas infinite zoom mechanism to object space textures, which allows the real-time stylization of dynamic objects without any sliding effects.

The above approaches tend to sacrifice accuracy in 3D motion to preserve the 2D appearance of the style. However, the converse can also lead to reasonable solutions. This is the approach adopted by *art maps* [Klein et al. 2000] and *tonal art maps* [Praun et al. 2001] that map a stylization texture directly on the 3D objects of the scene. Using object space textures, the style marks are perfectly attached to the object but are severely distorted by the perspective projection. The art maps solution relies on the *mipmapping* mechanism to adapt the scale of the texture according to the distance to the camera. This approach corrects the texture compression induced by depth and can be extended to the correction of perspective deformation using the more complex *ripmaps* mechanism [Klein et al. 2000]. As noted by Praun et al. [2001], higher temporal coherence can be obtained for binary styles by including the binary marks of one tonal art map level into the next level. Freudenberg et al. [2001] prove the performance of these methods by integrating an art map based non photorealistic rendering in the Fly3D game engine.

The infinite zoom mechanism described in this paper extends the art maps approaches in several ways. First, while the mipmap mechanism addresses the texture compression due to minification (zooming out), it produces simple linear blending for texture magnification (zooming in). Our mechanism in opposition produces quasi-constant size texture elements for any distance to the camera. Then, by combining several scales of the texture at a time, our approach is less specialized to binary styles than tonal art maps. Finally, mapping art maps on 3D objects requires the definition of a 2D parameterization of the 3D surfaces. The automatic definition of parameterization that avoid texture distortions or visible seams is still an active research topic. In the case of tonal art maps, Praun et al. [2001] propose to automatically compute the parameterization using *lapped textures* but this approach requires additional data structures. Because our approach is based on solid textures, it does not require this definition of additional surface parameterization.

3 Dynamic Solid Textures

We present in this section the object-space infinite zoom mechanism central to the dynamic solid textures. Various applications of our approach for the real-time coherent stylization of 3D animations will be described in section 4.

3.1 Object Space Infinite Zoom Mechanism

The contradictory goals of the infinite zoom illusion are to maintain the size of the texture elements as constant as possible in screen space while preserving the enlarging or shrinking of the texture elements required for a convincing feeling of zooming in or out. More generally, the infinite zoom on a signal can be seen as the infinite growth of its frequency. For an 1d signal (i.e. a sound) it corresponds to an endlessly increasing pitch, as demonstrated by Shepard [1964]. For 2D images, Cunzi et al. [2003] and Han et al. [2008] consider this process as the infinite generation of new visible details.

Drawing inspiration from the procedural noise function of Perlin [1985] and the infinite zoom mechanism of Dynamic Canvas [Cunzi et al. 2003], our method relies on the *fractalization* of a texture. In Dynamic Canvas, a self-similar image is obtained by linearly blending n octaves (doubled frequency) of a texture. When the observer moves forward or backward, the frequencies of the octaves are continuously shifted to produce an illusion of zoom.

We similarly define a *dynamic solid texture* as the weighted sum of n octaves Ω_i of the original solid texture. Each 3D object is then carved in such a solid texture, which naturally ensures a convincing feeling of zooming in and out: the texture will appear twice bigger when the object is twice closer to the camera. But care must be taken to keep the texture elements at a quasi-constant size in screen space. To this purpose, we introduce the notion of *zoom cycle* that occurs every time the appearant size of the texture doubles. In that case, each octave is replaced by the following one and a new high frequency octave is created, as illustrated in Fig. 2. Note that the fractalization process introduces new frequencies in the texture, along with a loss of contrast, as discussed in section 5. While reducing the number of octaves limits these artifacts, it also makes the apparition and disparition of texture elements more visible. Empirically we observed that $n = 4$ octaves is enough to deceive human perception.

[1] http://www.ogre3d.org

inria-00345835, version 3 - 20 Jul 2011

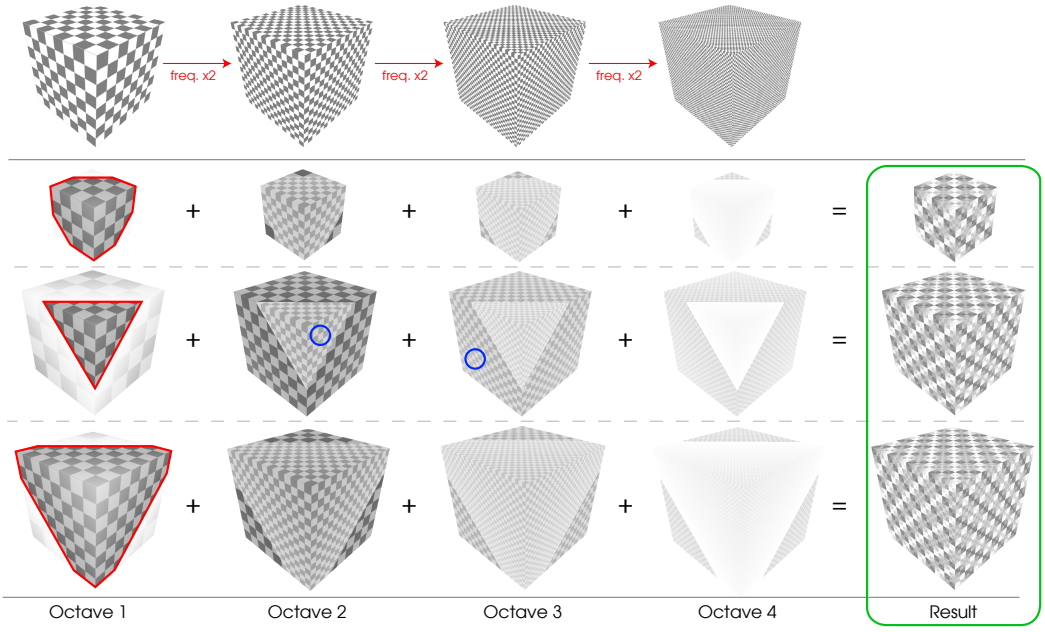


Figure 2: *Tridimensional infinite zoom mechanism (using a solid checkerboard texture for illustration purpose). Observe the globally invariant frequency of the solid texture produced by our algorithm (last column). The red line delineates the boundary between two consecutive zoom cycles. Note the frequency similarity between two adjacent octaves for two consecutive zoom cycles (blue circles).*

3.2 Proposed Algorithm

In practice, an object is embedded in its dynamic solid texture by deriving texture coordinates from vertex coordinates in the local mesh frame. This object space texturing solves the Dynamic Canvas limitation of approximating an entire scene with a plane. For the sake of simplicity, we use only one cube of solid texture that we sample at different rates to retrieve all n octaves. For a vertex $p(x, y, z)$ at distance z_{cam} from the camera, the 3D texture coordinates (u, v, w) for the octave i are given by:

$$(u, v, w)_i = 2^{i-1}(x, y, z)/2^{\lfloor \log_2(z_{cam}) \rfloor}$$

In this equation, the 2^{i-1} term scales the sampling rate so that the texture retrieved for one octave is twice smaller than the texture for the previous octave. The $\lfloor \log_2(z_{cam}) \rfloor$ term accounts for the refreshing of the octaves at each zoom cycle: the distance z_{cam} can be decomposed in $log_2(z_{cam})$ zoom cycles, making $\lfloor \log_2(z_{cam}) \rfloor$ the indicator of how many times each octave has been doubled during the zoom.

During a zoom cycle, we modulate each octave with a weight $\alpha_{i=1\ldots n}(s)$. These weights are dependent on s, the interpolation factor between the beginning and the end of the cycle:

$$s = \log_2(z_{cam}) - \lfloor \log_2(z_{cam}) \rfloor \ \in [0, 1]$$

We must impose three constraints on the weights in order to ensure a smooth transition during the frequency shift (Fig. 3). First, to avoid the sharp appearance/disappearance of texture elements, the first octave should appear at the beginning of the cycle while the last should disappear at its end:

$$\alpha_1(0) = 0 \ \text{ and } \ \alpha_n(1) = 0$$

Second, the weight of the intermediate octaves at the end of the cycle should be equal to the weight of the following octaves at the beginning of the next cycle:

$$\alpha_i(1) = \alpha_{i+1}(0) \ \ \forall i \in \{1, \ldots, n-1\}$$

Finally, the weights should sum to 1 to preserve a constant intensity. In our implementation, we use a linear blending which is fast to compute and coherent with the linearity of the zoom. We choose the following weights:

$$\begin{array}{lllllll} \alpha_1(s) & = & s/2 & & \alpha_2(s) & = & 1/2 - s/6 \\ \alpha_3(s) & = & 1/3 - s/6 & & \alpha_4(s) & = & 1/6 - s/6 \end{array}$$

The full process is illustrated in Fig. 2. The red line highlights the frequency shift. It corresponds to the distance after which the zoom cycle restarts. Note the frequency similarity between the octaves i and $i + 1$ for two consecutive zoom cycles. This correspondence ensures the texture continuity after the blending (last column).

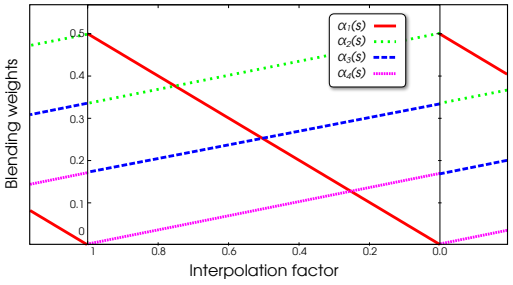


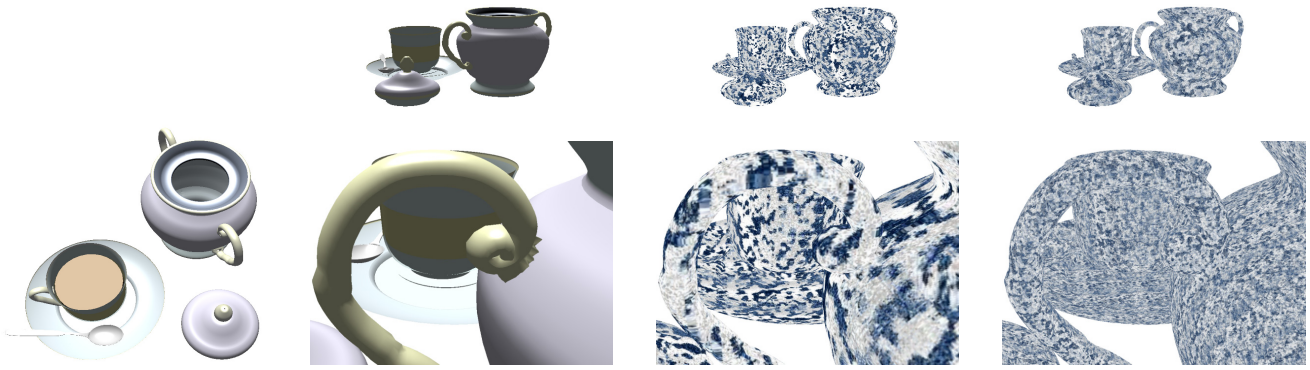
Figure 3: *Evolution of the blending weights during a zoom cycle. Note the continuity at the beginning and the end of the zoom cycle, which ensures an infinite number of cycles.*

3.3 Implementation details

We have implemented this infinite zoom algorithm in GLSL[2] and integrated it in the rendering engine OGRE (see shaders source code in appendix A). The *vertex shader* assigns the 3D texture coordinates of the vertex which can be the position of the vertices in the

[2]OpenGL Shading Language : http://www.opengl.org/documentation/glsl/

inria-00345835, version 3 - 20 Jul 2011



(a) *Top view* **(b)** *Phong shading* **(c)** *Traditional 3D texture mapping* **(d)** *Dynamic solid texture*

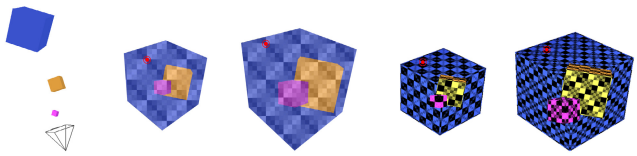
Figure 4: *Comparison of our dynamic solid textures (d) with standard texture mapping (c). Regardless of the distance from the camera (top and bottom row), our method provides quasi-constant size of texture elements.*

local object coordinate system (for deformable objects, this position in an undeformed pose of the object). The user can also specify an additional scaling factor to define the global size of the texture with regards to the depicted object. We compute the distance between a vertex and the camera as its z coordinate in the camera frame. Then the *fragment shader* blends linearly the four octaves of the solid texture for each pixel, according to the formula detailed in the previous section.

We produced the solid textures presented in this paper following two different approaches. We generated some of them procedurally (Perlin noise [Perlin 1985; Olano 2005], for example) using shaders. The more natural and complex textures have been synthesised from 2D exemplars using the algorithm proposed by Kopf et al. [2007]. In this case, the texture cube is stored in the graphic card memory (typically $128 \times 128 \times 128$ RGB pixels, *i.e.* 6 MB without compression in *DirectDraw Surface* format).

3.4 Results

We compare in Fig. 4 our dynamic solid textures with traditional texture mapping. With a standard (meaning fixed scale) mapping, the size of texture elements varies with depth due to perspective projection. On the contrary, with our infinite zoom approach, texture elements keep a globally constant size in image space independent of the zoom factor.



(a) *Top view* **(b)** *Dynamic Canvas* **(c)** *Dynamic Solid Textures*

Figure 5: *Comparison with Dynamic Canvas: note the sliding of the texture with the Dynamic Canvas method, highlighted by the red dot.*

Compared with Dynamic Canvas in Fig. 5, our approach suffers from perspective deformations when the surface is almost tangential to the viewing direction and from discontinuities at occlusion boundaries. However, these artifacts are limited by the infinite zoom mechanism that effectively reduces the apparent depth of the scene. On the other hand, as highlighted by the red dot on Fig. 5, sliding effects occur with the Dynamic Canvas approach. In our case, the texture elements follow perfectly the 3D motion of the scene. We refer the reader to the accompanying video for a better illustration of these sliding effects compared to the accurate motion and convincing infinite zoom produced by our approach. Note that the approaches of Coconu et al. [2006] and Breslav et al. [2007] would suffer from the same sliding problem.

The main advantages of our method are its simplicity of integration in existing rendering engines and its real-time performance. Our implementation in OGRE induces a small additional cost on the order of 10% in comparison with a traditional gouraud shading (computed in a shader). For the complex scene of Fig. 6 (135k tris) rendered at a resolution of 1280×1024 pixels, the framerate decrease from 70 fps to 65 fps with a 2.4GHz Core 2 Duo 6600, 4Go memory and a Geforce 8800 GT. This makes dynamic solid textures perfectly suited for real-time applications such as video games.

4 Application to coherent stylization

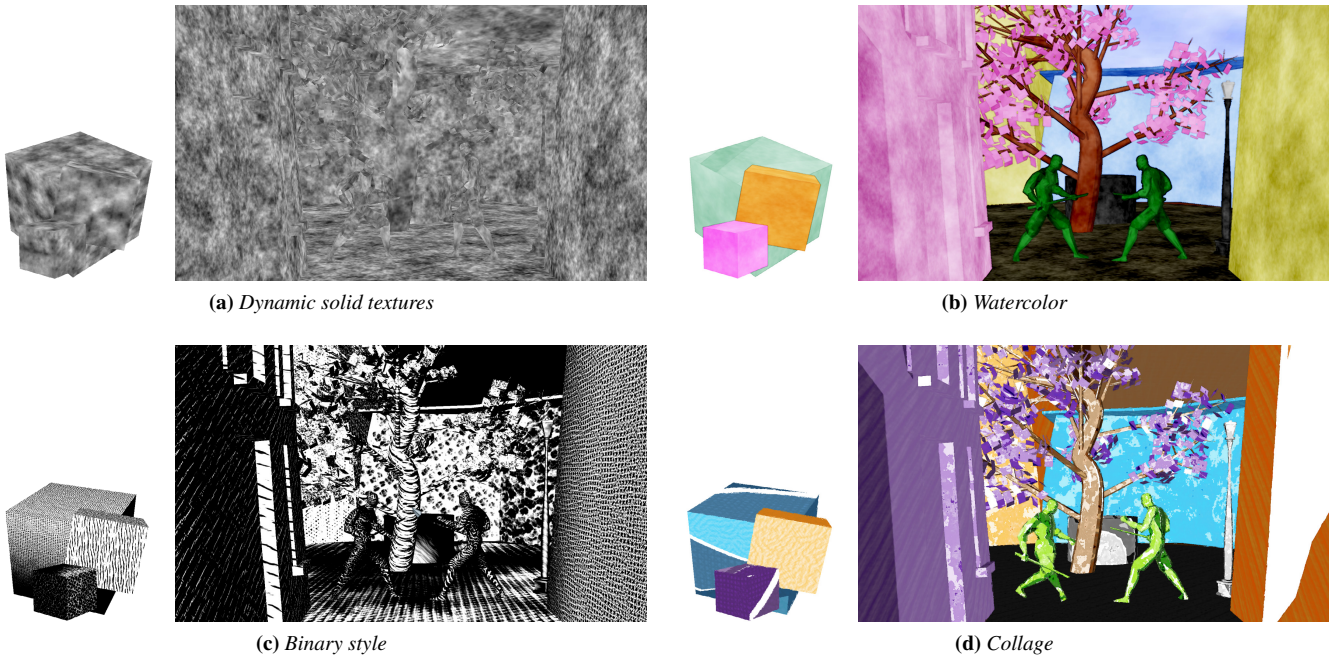
The technique of dynamic solid textures introduced in this paper can be used in a variety of rendering methods simply by replacing standard textures. We illustrate this principle with three real-time stylization algorithms (see Fig. 6 and 7). Two are inspired by existing styles (watercolor and binary style) and one is an original style that benefits from the variety of textures that our method can combine in one image (collage style).

4.1 Watercolor

We propose to extend the static watercolor pipeline of Bousseau et al. [2006] to dynamic scenes. This approach makes use of 2D gray level textures to mimic the variation of watercolor pigments density on paper. We simply replace these textures by our dynamic solid textures. As a result, each of the watercolor effects related to the texture (pigmentation pattern, wobbling) remains totally coherent during the animation. Fig. 6b and the accompanying video illustrate the richness of the medium and the temporal coherence obtained with this approach.

We compared our method with the *texture advection* approach proposed by Bousseau et al. [2007] for video watercolorization (see the accompanying video for the visual comparison). Note that the bidirectional advection requires the knowledge of the entire anima-

inria-00345835, version 3 - 20 Jul 2011



(a) *Dynamic solid textures* **(b)** *Watercolor*

(c) *Binary style* **(d)** *Collage*

Figure 6: *Complex scene (135k tris) rendered with various styles.*

tion sequence, which makes it unsuitable for real-time applications. However it produces animated watercolor with almost no noticeable sliding or deformations and thus gives a very strong 2D appearance on still frames. As can be seen on the video, the motion of the texture elements – perceived and interpreted as tridimensional by the observer – tends to conceal their 2D characteristics during an animation. Consequently we believe that the additional perspective deformation produced by our approach,although noticeable on still frames, is a small degradation compared to the vivid perception of perspective induced by the motion cues, particularly considering the performance gain.

4.2 Binary style

We obtain black-and-white shading styles (pen-and-ink, stippling, charcoal...) using a rendering pipeline very similar to the one of Durand et al. [2001]. In this method, strokes are simulated by truncating a *threshold structure* – a grayscale texture seen as a height-field – at different height with respect to the target tone. We apply a similar threshold on our dynamic solid textures. Fig. 6c illustrates the diversity of binary styles we can combine in one image by simply using a different dynamic solid texture for each object. During the animation, binary strokes appear and disappear progressively thanks to the infinite zoom mechanism.

Note that contrary to existing methods for pen-and-ink stylization [Hertzmann and Zorin 2000; Praun et al. 2001], our solid texture based approach does not orient the binary strokes along the principal directions of the surface. This limitation of the solid textures is discussed in section 5.

4.3 Collage

We finally propose a new stylization process that we call *collage*. This new style takes advantage of the diversity of the texture gallery that one can synthesize. Traditional collage consists in creating an image as a composition of several small strips of paper with various colors and textures. We mimic this style by assigning different dynamic solid textures to each tone of the image (obtained by a discretized shading model similar to toon shading [Lake et al. 2000]). In order to reinforce the paper aspect of the collage, a white border and a wobbling effect is added between the tone strips (Fig. 6d). The accompanying video illustrates the accurate temporal coherence of the paper strips, which would be difficult to obtain without our dynamic solid textures.

5 Discussion and Future Work

Infinite zoom mechanism The main limitation of our method, shared with Dynamic Canvas [Cunzi et al. 2003], texture advection [Bousseau et al. 2007] and to some extent with mipmaps-based approaches [Klein et al. 2000; Praun et al. 2001; Freudenberg et al. 2001], is the linear blending of multiple octaves that creates new frequencies and induces a global contrast loss compared with the original texture. Self-similar textures, such as Perlin fractal noise, paper or watercolor textures, are not severly affected by this blending. However, more structured textures can be visually altered, as individual features tend to overlap (the checkerboard texture being an extreme case). We plan to address this limitation in future work. One possible solution would be to replace the linear blending by a salience aware blending in the spirit of the work of Grundland et al. [2006]. This blending should better preserve the relevant features of the texture. The required salience data could be easily encoded in solid *feature maps* [Wu and Yu 2004].

Texture mapping We chose to develop the infinite zoom mechanism for solid textures because it avoids the need for an adequate 2D parameterization. However, the counterpart of this choice is that textures are decorrelated from the 3D surfaces. This can be seen as a limitation for styles that benefit from the orientation of texture elements along the surface. An example of such styles is pen-and-ink, for which it has been shown that orienting strokes along the principal directions of the surface emphasizes the shape of the objects [Hertzmann and Zorin 2000; Praun et al. 2001]. Nevertheless, the infinite zoom mechanism described in this paper is independent of the texture dimension and can also be applied on 2D textures if the required parameterization is available.

(a) *Collage* **(b)** *Binary style* **(c)** *Watercolor*

Figure 7: *Simple scene rendered with various styles.*

A limitation shared by all texture-based approaches concerns the texturing of the highly deformable objects. In this case, defining the texture coordinates as the vertices positions of the undeformed object creates additional dilatation/shrinking of the simulated medium, decreasing the perception of a two-dimensional look.

6 Conclusion

We introduced in this paper the dynamic solid textures, that maintain a quasi-constant size of textures in screen space independently of depth. In the context of non-photorealistic rendering, dynamic solid textures preserve the 2D appearance of style marks while ensuring temporal coherence. In addition, the simplicity and efficiency of this approach facilitates its use in real-time applications such as video games.

In the future we plan to extend the range of applications of our method. As an example, any type of feature line that stays fixed on the object surface (*ridges* [Ohtake et al. 2004], *demarcating curves* [Kolomenkin et al. 2008]) could be stylized according to a dynamic texture.

Acknowledgements

Most 3D models used in this research were kindly provided by Laurence Boissieux. The authors would like to thank Kartic Subr, Alexandrina Orzan, Frédo Durand, François Sillion and the anonymous reviewers for their valuable comments and suggestions. Many thanks to Tilke Judd for the voice-over.

References

BOUSSEAU, A., KAPLAN, M., THOLLOT, J., AND SILLION, F. 2006. Interactive watercolor rendering with temporal coherence and abstraction. In *NPAR 2006*, 141–149.

BOUSSEAU, A., NEYRET, F., THOLLOT, J., AND SALESIN, D. 2007. Video watercolorization using bidirectional texture advection. *ACM TOG (proc. of ACM SIGGRAPH 2007) 26*, 3, 104.

BRESLAV, S., SZERSZEN, K., MARKOSIAN, L., BARLA, P., AND THOLLOT, J. 2007. Dynamic 2d patterns for shading 3d scenes. *ACM TOG (proc. of ACM SIGGRAPH 2007) 26*, 3, 20.

COCONU, L., DEUSSEN, O., AND HEGE, H.-C. 2006. Real-time pen-and-ink illustration of landscapes. In *NPAR 2006*, 27–35.

CUNZI, M., THOLLOT, J., PARIS, S., DEBUNNE, G., GASCUEL, J.-D., AND DURAND, F. 2003. Dynamic canvas for immersive non-photorealistic walkthroughs. In *Graphics Interface*.

DANIELS, E. 1999. Deep canvas in disney's tarzan. In *ACM SIGGRAPH 99: Sketches and applications*, 200.

DURAND, F., OSTROMOUKHOV, V., MILLER, M., DURANLEAU, F., AND DORSEY, J. 2001. Decoupling strokes and high-level attributes for interactive traditional drawing. In *EGSR 2001*, 71–82.

FREUDENBERG, B., MASUCH, M., AND STROTHOTTE, T. 2001. Walk-Through Illustrations: Frame-Coherent Pen-and-Ink Style in a Game Engine. *Computer Graphics Forum (Proc. of Eurographics 2001) 20*, 3, 184–191.

GOOCH, B., AND GOOCH, A. 2001. *Non-Photorealistic Rendering*. AK Peters Ltd.

GRUNDLAND, M., VOHRA, R., WILLIAMS, G. P., AND DODGSON, N. A. 2006. Cross dissolve without cross fade: preserving contrast, color and salience in image compositing. In *Computer Graphics Forum : Proc. of Eurographics 2006*, vol. 25.

HAN, C., RISSER, E., RAMAMOORTHI, R., AND GRINSPUN, E. 2008. Multiscale texture synthesis. *ACM TOG (proc. of ACM SIGGRAPH 2008) 27*, 3, 1–8.

HERTZMANN, A., AND ZORIN, D. 2000. Illustrating smooth surfaces. In *ACM SIGGRAPH 2000*, 517–526.

KLEIN, A. W., LI, W. W., KAZHDAN, M. M., CORREA, W. T., FINKELSTEIN, A., AND FUNKHOUSER, T. A. 2000. Non-photorealistic virtual environments. In *ACM SIGGRAPH 2000*, 527–534.

KOLOMENKIN, M., SHIMSHONI, I., , AND TAL, A. 2008. Demarcating curves for shape illustration. *ACM TOG (Proc. of ACM SIGGRAPH ASIA 2008)*.

KOPF, J., FU, C.-W., COHEN-OR, D., DEUSSEN, O., LISCHINSKI, D., AND WONG, T.-T. 2007. Solid texture synthesis from 2d exemplars. *ACM TOG (proc. of ACM SIGGRAPH 2007) 26*, 3, 2.

LAKE, A., MARSHALL, C., HARRIS, M., AND BLACKSTEIN, M. 2000. Stylized rendering techniques for scalable real-time 3d animation. In *NPAR 2000*, 13–20.

MEIER, B. J. 1996. Painterly rendering for animation. In *ACM SIGGRAPH 96*, 477–484.

OHTAKE, Y., BELYAEV, A., AND SEIDEL, H.-P. 2004. Ridge-valley lines on meshes via implicit surface fitting. *ACM TOG (Proc. of ACM SIGGRAPH 2004)*, 609–612.

OLANO, M. 2005. Modified noise for evaluation on graphics hardware. In *Graphics hardware*, 105–110.

PASTOR, O. M., FREUDENBERG, B., AND STROTHOTTE, T. 2003. Real-time animated stippling. *IEEE Computer Graphics and Applications 23*, 4, 62–68.

inria-00345835, version 3 - 20 Jul 2011

inria-00345835, version 3 - 20 Jul 2011

PERLIN, K. 1985. An image synthesizer. *Computer Graphics (Proc. of ACM SIGGRAPH 85) 19*, 3, 287–296.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. In *ACM SIGGRAPH 2001*, 579–584.

SHEPARD, R. N. 1964. Circularity in judgments of relative pitch. *The Journal of the Acoustical Society of America 36*, 12, 2346–2353.

STROTHOTTE, T., AND SCHLECHTWEG, S. 2002. *Non-Photorealistic Computer Graphics: Modeling, Rendering and Animation*. Morgan Kaufmann.

VANDERHAEGHE, D., BARLA, P., THOLLOT, J., AND SILLION, F. 2007. Dynamic point distribution for stroke-based rendering. In *EGSR 2007*, 139–146.

WU, Q., AND YU, Y. 2004. Feature matching and deformation for texture synthesis. *ACM TOG (proc. of ACM SIGGRAPH 2004) 23*, 3, 364–367.

A Infinite Zoom Shaders

Vertex Shader

```
uniform float obj_scale;
varying float dist;

void main(void)
{
        //distance to the camera
        vec4 posTransform = gl_ModelViewMatrix*gl_Vertex;
        dist = abs(posTransform.z);

        //volumetric texture coordinate
        gl_TexCoord[0] = gl_Vertex/obj_scale;

        //projected position
        gl_Position = ftransform();
}
```

Fragment Shader

```
uniform sampler3D solidTex;
varying float dist;

vec4 main(void)
{
        //number of zoom cycles
        float z = log2(dist);
        float s = z-floor(z);

        //scale factor according to fragment distance to
            the camera
        float frag_scale = pow(2.0,floor(z));

        //octave weight according to the interpolation
            factor
        float alpha1 = s/2.0;
        float alpha2 = 1.0/2.0-s/6.0;
        float alpha3 = 1.0/3.0-s/6.0;
        float alpha4 = 1.0/6.0-s/6.0;

        //texture lookup
        vec4 oct1 = alpha1*texture3D(solidTex,
            gl_TexCoord[0].xyz/frag_scale);
        vec4 oct2 = alpha2*texture3D(solidTex, 2.0*
            gl_TexCoord[0].xyz/frag_scale);
        vec4 oct3 = alpha3*texture3D(solidTex, 4.0*
            gl_TexCoord[0].xyz/frag_scale);
        vec4 oct4 = alpha4*texture3D(solidTex, 8.0*
            gl_TexCoord[0].xyz/frag_scale);

        //blending
        vec4 n = oct1+oct2+oct3+oct4;

        gl_FragColor = n;
}
```