The Filtering of 3d textures

John Buchanan*
Imager,
Department of Computer Science,
University of British Columbia,
Vancouver, British Columbia
Canada V6T 1W5
(604) 228-2218
e-mail buchanan@cs.ubc.ca

Abstract

Solid textures or textures defined over 3d have provided computer graphics with a new set of tools. Using these tools we can obtain effects which were difficult, if not impossible, to obtain with 2d texture mapping methods. These textures are simple to implement using either scan line renderers or ray-tracers. Research in this area has focused on the generation and rendering of these textures with little consideration of the sampling and filtering problems which arise when these textures are used. The de-facto answer to the filtering problem has been to clamp the signal so that no aliasing frequencies appear in the texture. Clamping a signal can cause variety of problems including energy loss and cross-spectrum energy leaks, we show how these artifacts can arise in fairly simple examples. We propose a filtering method for filtering 3d textures which uses a box shaped kernel. A variety of filters can then be applied over this box. This method is illustrated by using two kernels: the first aligned with the tangent plane of the surface; the second aligned along the ray from the intersection point to the eye.

Keywords: Solid texture, Aliasing, Clamping, Filtering, Quadrature rules.

Résumé

Les textures définies en 3d procurent de nouveaux outils en infographie. En utilisant ces outils, il est possible de créer des effets difficiles, sinon impossibles à obtenir avec les techniques conventionnelles de mapping de texture 2d . Ces textures 3d sont faciles à implanter dans des algorithmes de rendu à balagage et lancer de rayons. Jusqu'à prèsent la recherche dans ce domaine était concentrée sur la création et le rendu de ces textures sans trop de considération pour les problèmes d'échantillonnage et de filtrage qui surviennent lors de l'utilisation de ces textures. La solution au problème de filtrage trop souvent adoptée était de tronquer le signal, éliminant ainsi l'aliasing des hautes fréquences. Tronquer un signal peut causer plusieurs problèmes allant d'une perte globale d'énergie à des pertes d'énergies à travers le spectre. Nous montrons comment ces problèmes peuvent apparaître dans de simples exemples. Nous proposons une technique de filtrage pour les textures 3d utilisant un noyau en forme de boîte. Plusieurs types de filtres peuvent alors être employés sur cette boite. Cette technique est illustrée avec deux noyaux: le premier aligné au plan tangent à la surface; le second aligné le long du rayon défini par les positions de l'oeil et du point d'intersection.

Mots clés: Texture en 3d, Aliasing, Tronquage, Filtrage, Règles de quadratures.

1 Introduction

In 1984 Fournier and Amanatides [grin84] and Perlin [perl84] talked about solid-textures. They defined a solid texture as a map which mapped $\Re^3 \rightarrow \mathcal{T}\{\rho_1, \rho_2, ...\rho_p\}$, where ρ_i is some shading parameter. This definition is the one which we are going to use in this paper and it can equally include textures which are defined procedurally or textures which are digitized and the values interpolated from the nearest sample point. Gardner[gard84, gard85] presented a system for generating transparency maps over ellipsoidal and quadric surfaces. Some of these textures were defined as functions of $\Re^3$ and so they could be called 3d textures. Using these maps he modelled clouds and terrain for use in flight simulators.

In 1985 Perlin [perl85] and Peachey [peac85] independently introduced the term solid texture. Peachey used a variety of solid textures which show how solid textures could be generated and applied. In this paper he recognized that aliasing artifacts could arise but did not address the issue of anti-aliasing his textures. Perlin proposed a system which used various transformations of a solid noise function to model several textures. Recognizing the aliasing problems which could be introduced, he suggested clamping the texture based on the size of pixels.

In 1989 Lewis [lewi89] introduced more noise deformation algorithms, his main contribution was the introduction of Wiener interpolation for the use of noise

*This research was partially supported by grants from the Natural Sciences and Engineering Research Council and from the University of British Columbia and equipment donations from IBM

generation and manipulation. He did not address filtering or sampling issues.

Perlin [perl89] extended the ideas presented in [perl85]. He used the noise function and its transformations to deform object surfaces. These were rendered using a ray marching algorithm. Some of the objects he generates have a very high frequency surface. He uses these high frequency deformations to model fur-like textures over object's surfaces. Again Perlin suggests the use of clamping to filter his images.

Kajiya and Kay [kaji89] presented a system for modeling and rendering fur like textures. Their texture parameters included colour, opacity, local coordinate frames, and shading parameters. In a typical application a small texel of this texture would be generated. This generic texel would then be replicated over the surface of the object and rendered. Their texels are rendered by ray tracing through the texture. There is no discussion of potential aliasing problems or filtering solutions.

These two papers are interesting since they both produce a similar set of 3d textures, but the process by which they arrive there is quite different. Kajiya and Kay set out to produce a model of fur which could be applied to objects, their approach was very focused on generating a texture which would have the visual characteristics of fur. Perlin on the other had uses the surface deformation algorithms to generate objects with increasingly higher frequency surfaces until he arrives at a model for a surface which resembles fur.

Volume rendering systems attempt to display textures without using any geometric objects. There are two approaches to the problem: pixel based and voxel based. Pixel based volume rendering systems calculate the pixel value for a particular view. Most pixel based volume renderers use a ray tracing method with a line integral being performed along the ray. Voxel based volume rendering renderers calculate the influence that each voxel will have on the screen and update the pixels affected.

Upson and Keeler [upso88] present two methods for volume rendering, ray casting and voxel by voxel processing. Their ray casting technique steps through the data starting near the eye and traveling into the data. The evaluation of the colour coefficients in the voxel method is performed by clipping the voxel by a projection of the pixel[1] into the data. Once the volume of the voxel which projects onto the pixel is computed a 3d integral is performed over this volume to evaluate the texture components and opacity. the pixel. The resulting colour and opacity values are composited onto the pixel.

Westover [west90] presents a method for calculating the convolution of a reconstruction filter with a set of samples using a precomputed *footprint*. He first computes a generic footprint for a generic kernel, a sphere of radius 1. The footprint is generated by projecting the sphere onto the screen. For a discreet grid in the resulting circle the integral along z through[2] the kernel is computed. The resulting values represent the influence that a particular data point will have over its footprint. When a view is selected the kernel surrounding each sample s_i is projected to its footprint $\mathcal{F}_i$ and the generic footprint $\mathcal{F}_g$ is transformed onto $\mathcal{F}_i$. This 'view-transformed' footprint can then be used to calculate which pixels are affected and in what manner. When this has been done for all the samples and the information appropriately accumulated the process is done.

Even though these methods perform filtered voxel rendering they are not appropriate for the display of textured objects in traditional computer graphics applications. When we are displaying textured objects we are not interested in displaying all the texture data, but rather we are interested in calculating the texture coordinates of a point on the object.

1.1 Clamping

In 1982 Norton et al. [nort82] introduced a clamping method for the purpose of anti-aliasing, in this method the signal is clamped so that the aliasing frequencies are damped. In their paper the clamping method is developed for signals whose frequency spectrum is known so that when the signal is clamped it can be replaced with its average. Towards the end of the paper they recognize that many textures must be dealt with for which there is no spectral information. In this case they suggest clamping the signal since in their experience this has lead to acceptable results. Even though clamping the signal in such a fashion will remove the aliasing frequencies it is easy to show that the energy of the resulting signal has been diminished.

Parseval's theorem [rose76] can be considered as a stating the conservation of energy between a signal $s(t)$ and its Fourier transform $F(s)$.

$$\int_{-\infty}^{\infty} |s(t)|^2 dt = \int_{-\infty}^{\infty} |F(s)|^2 \, ds$$

Thus replacing the function $s(t)$ with a clamped function $s'(t)$ which contains no frequencies greater than some frequency f_o will remove energy from the right hand of the above equation since

$$\begin{aligned} \int_{-\infty}^{\infty} |F(s)|^2 dt &= \int_{-\infty}^{-f_o} |F(s)|^2 ds + \int_{-f_o}^{f_o} |F(s)|^2 \, ds \\ &\quad + \int_{f_o}^{\infty} |F(s)|^2 \, ds \\ &> \int_{-f_o}^{f_o} |F(s)|^2 \, ds \\ &= \int_{-\infty}^{\infty} |F(s')|^2 \, ds \end{aligned}$$

and the energy we have removed from the signal is

$$\mathcal{E}(s') = \int_{-\infty}^{-f_o} |F(s)|^2 \, ds + \int_{f_o}^{\infty} |F(s)|^2 \, ds$$

The following example clearly illustrates the problem. In this example we will use the function $f(t) = \sum_{n=1}^{20} \sin(nt)$. This function and a plot of $|f(t)|^2$ are shown in figure 1 for different values of N. The first plot illustrates the function $f(t)$ on the left and on the

[1] Pixels are square in this context

[2] This calculation is done relative to the traditional eye coordinate system

right a plot of $|f(t)|^2$ for $t \in [-10, 10]$ and N=20. In the subsequent plots we show the result of clamping the signal for $N = 15$, and 10 respectively. The integral $I = \int_{-10}^{10} |f(t)|^2 dt$ in these plots decreases significantly from plot to plot as the signal is progressively clamped.

Our criticism of clamping would not be complete without a set of images which show the effects of clamping. In the image presented in figure 4 we show a plane which has been textured with the turbulence function presented by Perlin [perl85].

$$f(u,v,w) = \sum_{N=1}^{pixel<\frac{1}{2^n}} \frac{1}{2^n} |noise(x2^n, y2^n, z2^n)|$$

Near the horizon we have significant aliasing artifacts which we would like to get rid of. In the next image (figure 5) we see the result of clamping the texture. The texture on the plane near the horizon has been clamped to 0.

A simple solution to this energy loss problem would be to compensate the rest of the texture components so that the energy is conserved. This can lead to chroma aliasing as the following example shows. Consider a two colour texture in which each colour signal is narrow band. If one of these signals has much higher spectral frequencies than the other, then the clamping of this signal will occur sooner on a perspective plane. If we try to compensate for the energy loss by transferring the energy from the clamped signal to the remaining signal we will have introduced an incorrect colour.

2 3D Box filter

The task of selecting a filter is a complex one. This selection process must select a filter whose cost and quality matches the requirements of the application. The computer graphics literature is full of good filters which can be used in a variety of 2d-texture filtering applications. These filters provide users with a wide variety of features, strengths, and weaknesses. However when one considers the task of filtering a 3d-texture there is little to choose from even though in some applications the careful application of clamping may provide an adequate anti-aliasing technique. We wish to extend this choice of filters as part of an ongoing project to develop filtering techniques for 3d-textures. A first step approximation can be generated by performing a weighted integral

$$\mathcal{T}(P_h) = \int_{\mathcal{K}} \mathcal{F}(x - x_o, y - y_o, z - z_o) t(x, y, z) dv$$

over the rectangular volume $\mathcal{K}$. In this section we show the construction of two integral volumes $\mathcal{K}_{surf}$ and $\mathcal{K}_{ray}$.

The first integral volume $\mathcal{K}_{surf}$ is intended to be used for objects made of materials with a low transparency coefficient such as wood or stone. This volume is aligned with the tangent plane of the surface and typically has a small dimension perpendicular to this plane. The second volume $\mathcal{K}_{ray}$ is intended to be used for objects which exhibit some degree of translucency. This kernel is aligned with the ray from the object to the eye and can penetrate the object to arbitrary depths.

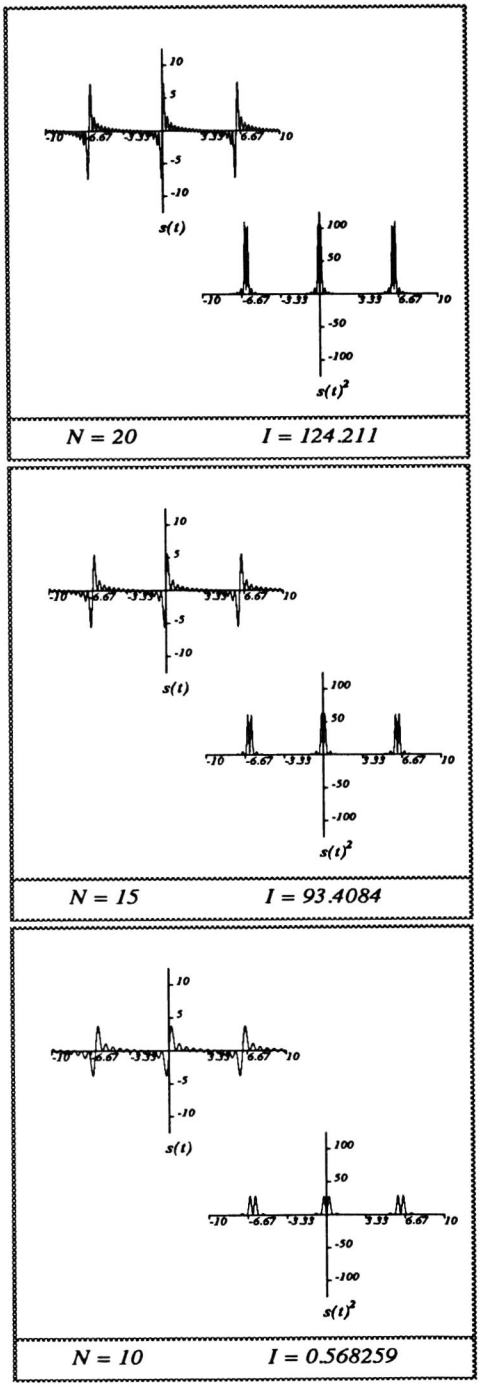

Figure 1: Function $s(t)$ and $s(t)^2$ for $N = \{20, 15, 10\}$ where I is the integral $\int_{-10}^{10} s(t)^2$

2.1 A surface box filter

A simple yet effective approach to filtering 3d textures is to construct a box kernel which is aligned with the tangent plane of an object. We wish this box to be a good representation of the projection of the pixel onto the tangent plane. One such box is defined by calculating the box which is the projection of a circular pixel onto the tangent plane of the object. Note that when perspective projection is being used most of the cones generated by the projection of circular pixels on the screen will not be circular cones but elliptical cones. Since we are interested in a quick approximation we will approximate these elliptical cones by the smallest circular cones which will enclose the elliptical cone. This approximation will cause us to over filter some pixels. This overfiltering may become apparent in the edge pixels of images with strong perspective.

Let P_h be the projection of the centre of the pixel onto the object. It is easy to show that P_h does not correspond to the centre of the projected ellipse, rather it lies on the major axis somewhere between the two foci. To calculate the box we need to calculate the vectors U and V which will lie parallel to the minor and major axis respectively. The centre of the box P_c is then calculated by computing the two quantities u_A and u_B which are the distances from the intersection point P_h to either end of the major axis respectively, as illustrated in figure 3. The centre of the box is the point $P_c = \frac{u_A - u_B}{2} U + P_h$

Let R be a ray which is shot from the eye through the centre of the pixel. This ray and the circumference of the pixel define an elliptical cone. When we examine the angle between the line R and the edges of the cone we find that for every pixel there is a maximum angle θ which can be calculated by

$$\theta = tan^{-1}\left(\frac{\max(\Delta_x, \Delta_y)}{d}\right)$$

$$\Delta_x = view_x/res_x, \Delta_y = view_x/res_y$$

and d is the distance from the eye to the projection plane.

We can then calculate a box aligned with the tangent plane defined by the vectors U, V, W as illustrated in figure 2. If the tangent plane is perpendicular to the incident ray then we choose U and V so that the kernel is aligned with the pixel. If the tangent plane is not perpendicular to the incident ray then we define the vectors $\hat{U}, \hat{V}$, and $\hat{W}$ as follows

$$\hat{V} = \frac{N \times E}{||N \times E||}$$

$$\hat{U} = \frac{N \times \hat{V}}{||N \times \hat{V}||}$$

$$\hat{W} = N$$

We now need to calculate the values u_o, v_o, and w_o where u_o, v_o, w_o are half the length, width, and height of the kernel respectively. Since the centre of the pixel does not project to the centre of the required box we must perform some intermediate calculations, namely the calculation of u_A and u_B

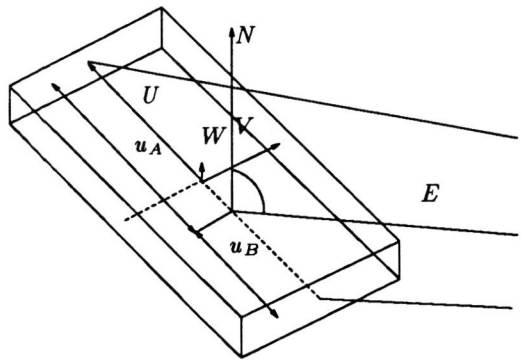

Figure 2: The Box Kernel construction

$$u_A = \frac{d_h \sin(\theta)}{\cos(\theta + \varphi)}$$

$$u_B = \frac{d_h \sin(\theta)}{\cos(\theta - \varphi)}$$

$u_o = \frac{u_A + u_B}{2}$ and $v_o = d_h \sin(\theta)$

The last dimension ($w_o \epsilon$) of the box is the height of the box perpendicular to the tangent plane, in our current implementation we have allowed the user to set this as a parameter. We have implemented this as a user defined parameter since different settings of this parameter will produce quite different effects. If, for example, one wished to present an object with a soft or translucent object this parameter should be set quite large. When hard surfaces are desired ϵ should be set close to 0. modeled by setting w close to 0. In figure 6 we present the boxes which were generated over the surface of a sphere for a 30 by 30 image. The image on the left of figure 6 is rendered with the same viewing parameters as those used to generate the kernels. The image on the right shows the same scene from a higher point of view. This second view shows the variety of shapes that our kernels can assume.

2.2 A volume box filter

Kajiya [kaji89] and Perlin [perl89] use a ray marching algorithm to compute the integral

$$I = \int_{s_o}^{s_1} t(r(s))ds$$

where $r(s)$ is the equation of the ray and s_o, s_1 are the entry and exit points into the texture. In this case we calculate a kernel $\mathcal{K}_m$ which is aligned along the ray. In the general case we can define $\mathcal{K}_w$ by the vectors $\hat{U}_m, \hat{V}_m$, and $\hat{W}_m$ where $\hat{U}_m$ and $\hat{V}_m$ define a plane perpendicular to the ray and $\hat{W}_m$ is parallel to the ray. The kernel $\mathcal{K}_m$ is then defined by the vectors $U = u_o\hat{U}, V = v_o\hat{V}$, and

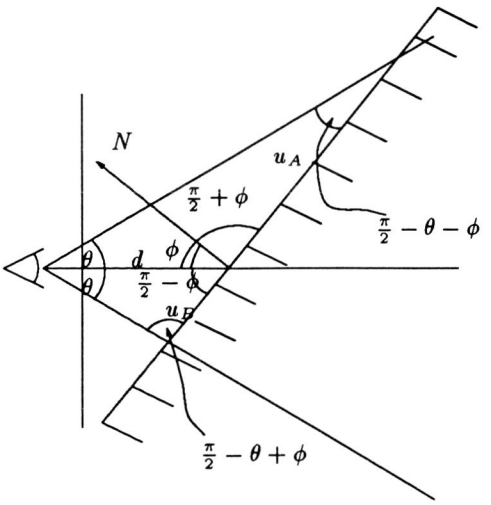

Figure 3: Calculation of u_A and u_B

$W = \frac{\|r(s_1)-r(s_o)\|}{2}\hat{W}$. For most applications we would set u_o/v_o equal to the aspect ration of the display device. In our current application the aspect ratio is 1 so we have set $u_o = v_o = d_h \sin(\theta)$. Once we have defined the kernel we can simply use the quadrature methods described later to evaluate the integral.

In our current implementation the user can include in the parameters a spread factor which is applied to the u_o and v_o parameters. By increasing/decreasing this spread factor we can produce over/under filtered images. The use of this parameter is illustrated in figure 8

3 Evaluating the integral over the kernel

Given a function $f : \Re^3 \rightarrow \Re^1$ defined for all u, v, w in $[-1,1]\times[-1,1]\times[-1,1]$ we wish to calculate a numerical approximation to the integral, we must also satisfy the constraint that $\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} du dv dw = 1$.

$$I = \frac{1}{8}\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} f(u,v,w)\,du\,dv\,dw$$

with a maximum error ϵ. We can extend the common adaptive quadrature method based on Simpson's method [burd81] to deal with this case as follows. From calculus we can rewrite the integral

$$8I = I_u = \int_{-1}^{1} \hat{f}_u(u)\,du$$

where

$$\begin{aligned} I_v &= \hat{f}_u(u) \\ &= \int_{-1}^{1}\int_{-1}^{1} f(u,v,w)\,dv\,dw \\ &= \int_{-1}^{1} \hat{f}(v)\,|_{u=u_o}\,dv \end{aligned}$$

similarly

$$\hat{f}_v(v_o)\,|_{u=u_o} = \int_{-1}^{1} \hat{f}_w(w) \Big|_{\substack{u=u_o \\ v=v_o}} dw$$

and

$$I_w = \hat{f}_w(v) \Big|_{\substack{u=u_o \\ v=v_o}} = \int_{-1}^{1} f(u_o, v_o, w)$$

We now have three integrals in one variable. Each of these can be approximated using appropriate quadrature rules. We have chosen to use adaptive Simpson's rule for the approximation of I_u and I_v. When the magnitude of ϵ is small compared to u_o and v_o we use a order 3 gaussian approximation for the evaluation of I_w. If ϵ is large then we use an adaptive Simpson's rule. For a more detailed discussion of quadrature rules please see [burd81].

4 Timing: or we knew you would ask this question

In those cases where a closed form solution for the convolution integral can be calculated apriori timing should not be a problem, unless the evaluation of the resulting function proves to be to costly. In the general case the increased cost is dependent on the quadrature method(s) used in the convolution evaluation, on the function, and on the size of the kernel $\mathcal{K}$. In the situation where we have chosen to use Simpson's quadrature rule for the first two levels of the integral and a degree 3 Gaussian approximation for the last integral. The minimum number of texture evaluations required would be $5 \times 5 \times 3 = 75$. When adaptive quadrature methods are being used the texture evaluations could become much greater. In general functions with high frequency components will cause these adaptive quadrature methods to work harder. For many applications it may prove sufficient to use a fixed quadrature method for all our integrals.

Using a network of 4 IBM RS/6000 workstations and 13 SGI iris workstations the images in figures 7 and 8 were rendered at a resolution of 320 by 240. The point sampled version of the marble block in figure 8 required 45 seconds for display. Using a degree 3 Gaussian approximation for the integrals the display time was 2 minutes. When we used adaptive quadrature for the display of this image the display time was 6 minutes.

5 Examples

Figure 7 shows a progression of a sphere textured with the texture function

$$\mathcal{T}(p) = \begin{cases} (1,1,1) & \text{if } \|p\| > 0.9 \\ (0,1,0) & \text{if } 0.9 \geq \|p\| > 0.7 \\ (t(10x, 10y, 10z), 0, 0) & \text{otherwise} \end{cases}$$

$\epsilon = .1$	$\epsilon = .4$	$\epsilon = .7$
$\epsilon = .2$	$\epsilon = .5$	$\epsilon = .8$
$\epsilon = .3$	$\epsilon = .6$	$\epsilon = .9$

Table 1: Parameters for figure 7

Box filter	Bartlett filter	Volume filter
$\epsilon = 0.01$ $spread = 1.0$	$\epsilon = 0.01$ $spread = 1.0$	$\epsilon = 0.025$ $spread = 1.0$
$\epsilon = 0.01$ $spread = 2.0$	$\epsilon = 0.01$ $spread = 2.0$	$\epsilon = 0.100$ $spread = 1.0$
$\epsilon = 0.01$ $spread = 4.0$	$\epsilon = 0.01$ $spread = 4.0$	$\epsilon = 0.200$ $spread = 1.0$

Table 2: Parameters for figure 8

where $t(x, y, z) = turbulence(x, y, z)$ is the turbulence function proposed by perlin [perl85]. Nine versions of the sphere are presented with various penetration depths for the integral volume $\mathcal{K}_{ray}$. The ϵ values for this figure are presented in table 1 In figure 8 we illustrate a marble block whose size is $2 \times 2 \times 2 \times$ rendered with various filter parameters. In the first column we have a surface box filter, in the second column we have an application of a bartlett filter to the same box, and in the third column we apply a volume filter to the texture. In table 2 table we outline the parameters of the filters in this image.

6 Conclusion

In this paper we have shown that there is a need for 3d texture filtering. We illustrated some of the shortcomings of clamping as a filtering or anti-aliasing technique. We proposed the use of filters evaluated over rectangular boxes. These boxes allow the application of arbitrary filters over rectangular volumes of the texture. We illustrated the use of these boxes with two filter kernels: the first, a kernel aligned with the tangent plane of the object; the second, a kernel oriented with the ray. We presented a set of images which illustrate the applications of these techniques.

7 Future work

This work is a first step in a project whose goal is to develop *adequate* filters for 3d textures. The author is very much challenged by the idea of extending the constant cost 2d filtering algorithm presented by Fournier and Fiume [four88].

Acknowledgements

Alain Fournier was an excellent challenger and generator of ideas. Pierre Poulin did not believe anything I said. Peter Cahoon again became the lab photographer. The members of the Imager and GraFiC lab did not complain (to much) when I was using all the machines. Ruhamah provided a constant reminder of reality. Mary Jane constantly encouraged me. To all of you THANKS!!!!!!!

References

[burd81] R. Burden, J. Faires, and A. Reynolds. *Numerical Analysis, Second edition.* PWS Publishers, Boston, Massachusetts, 1981.

[four88] Alain Fournier and Eugene Fiume. "Constant-Time Filtering with Space-Variant Kernels". *Computer Graphics (SIGGRAPH '88 Proceedings)*, Vol. 22, No. 4, pp. 229–238, August 1988.

[gard84] Geoffrey Y. Gardner. "Simulation of Natural Scenes Using Textured Quadric Surfaces". *Computer Graphics (SIGGRAPH '84 Proceedings)*, Vol. 18, No. 3, pp. 11–20, July 1984.

[gard85] G.Y. Gardner. "Visual Simulation of Clouds". *Computer Graphics (SIGGRAPH '85 Proceedings)*, Vol. 19, No. 3, pp. 297–303, July 1985.

[grin84] D. Grindal. "The Stochastic Creation of Tree Images". Master's thesis, University of Toronto, Toronto, Canada, 1984.

[kaji89] James T. Kajiya and Timothy L. Kay. "Rendering Fur with Three Dimensional Textures". *Computer Graphics (SIGGRAPH '89 Proceedings)*, Vol. 23, No. 3, pp. 271–280, July 1989.

[lewi89] J. P. Lewis. "Algorithms for Solid Noise Synthesis". *Computer Graphics (SIGGRAPH '89 Proceedings)*, Vol. 23, No. 3, pp. 263–270, July 1989.

[nort82] A. Norton, A.P. Rockwood, and P.T. Skolmoski. "Clamping: A Method of Antialiasing Textured Surfaces by Bandwidth Limiting in Object Space". *Computer Graphics (SIGGRAPH '82 Proceedings)*, Vol. 16, No. 3, pp. 1–8, July 1982.

[peac85] D.R. Peachey. "Solid Texturing of Complex Surfaces". *Computer Graphics (SIGGRAPH '85 Proceedings)*, Vol. 19, No. 3, pp. 279–286, July 1985.

[perl84] K. Perlin. "A tutorial on texture mapping". *Siggraph course.*, July 1984.

[perl85] K. Perlin. "An Image Synthesizer". *Computer Graphics (SIGGRAPH '85 Proceedings)*, Vol. 19, No. 3, pp. 287–296, July 1985.

[perl89] Ken Perlin and Eric M. Hoffert. "Hypertexture". *Computer Graphics (SIGGRAPH '89 Proceedings)*, Vol. 23, No. 3, pp. 253–262, July 1989.

[rose76] A. Rosenfeld and A. Kak. *Digital Picture Processing.* Academic publishers, New York, New York, 1976.

[upso88] Craig Upson and Michael Keeler. "V-BUFFER: Visible Volume Rendering". *Computer Graphics (SIGGRAPH '88 Proceedings)*, Vol. 22, No. 4, pp. 59–64, August 1988.

[west90] Lee Westover. "Footprint Evaluation for Volume Rendering". *Computer Graphics (SIGGRAPH '90 Proceedings)*, Vol. 24, No. 4, pp. 367–376, August 1990.

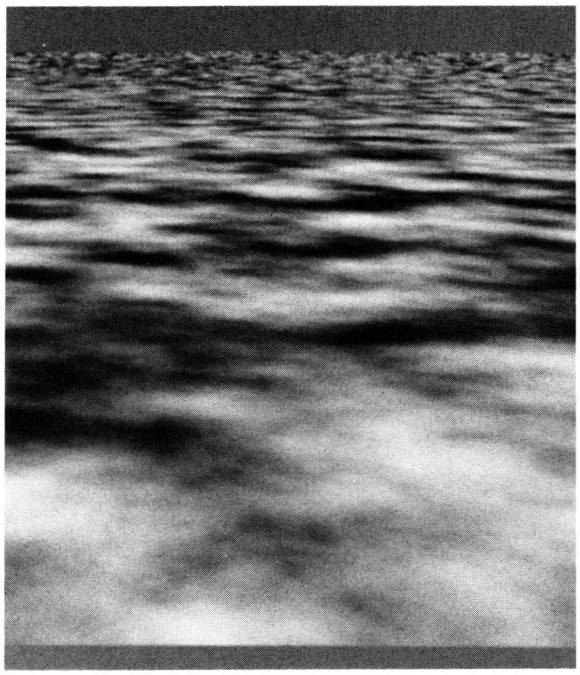

Figure 4: Point sampled turbulence function

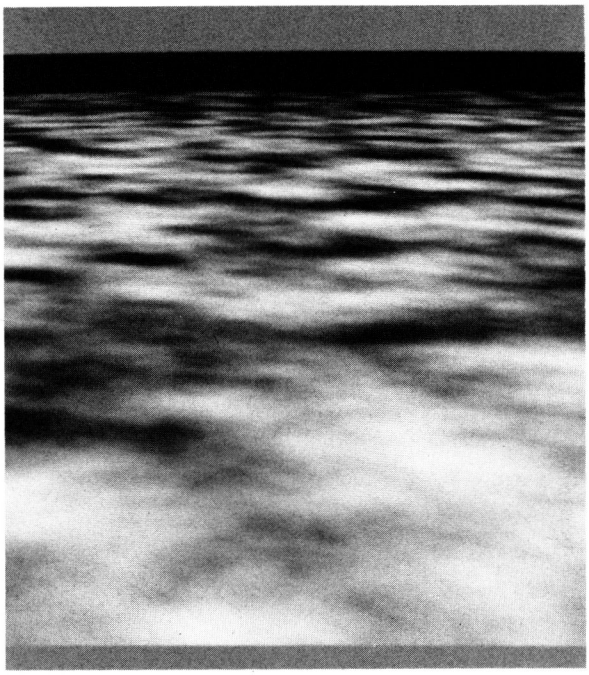

Figure 5: Clamped turbulence function

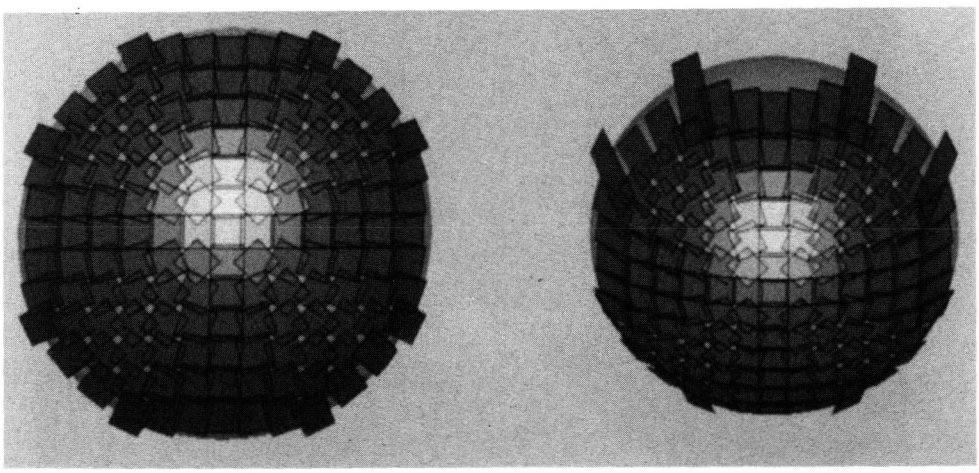

Figure 6: Kernels generated by the system

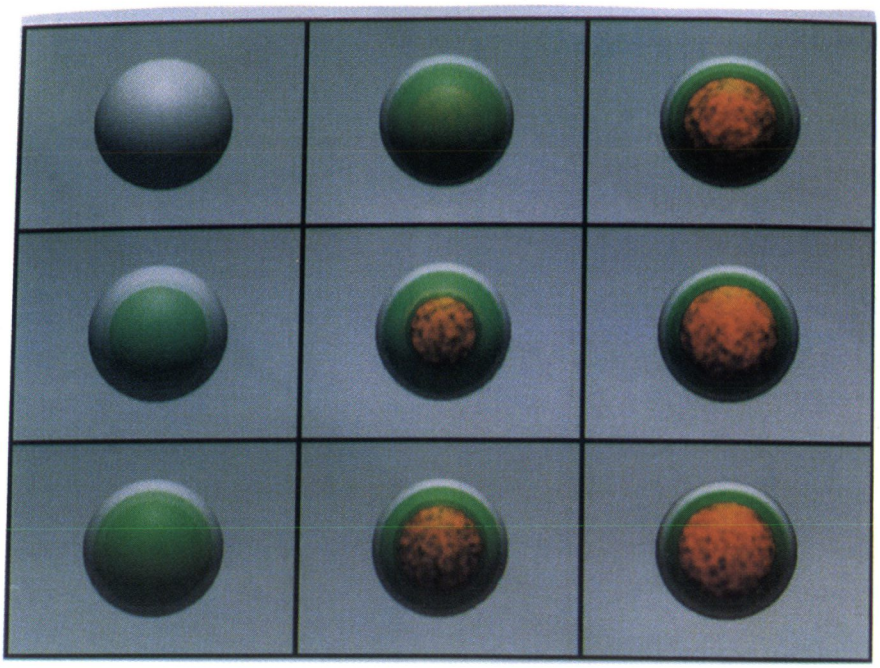

Figure 7: Filter aligned with the ray

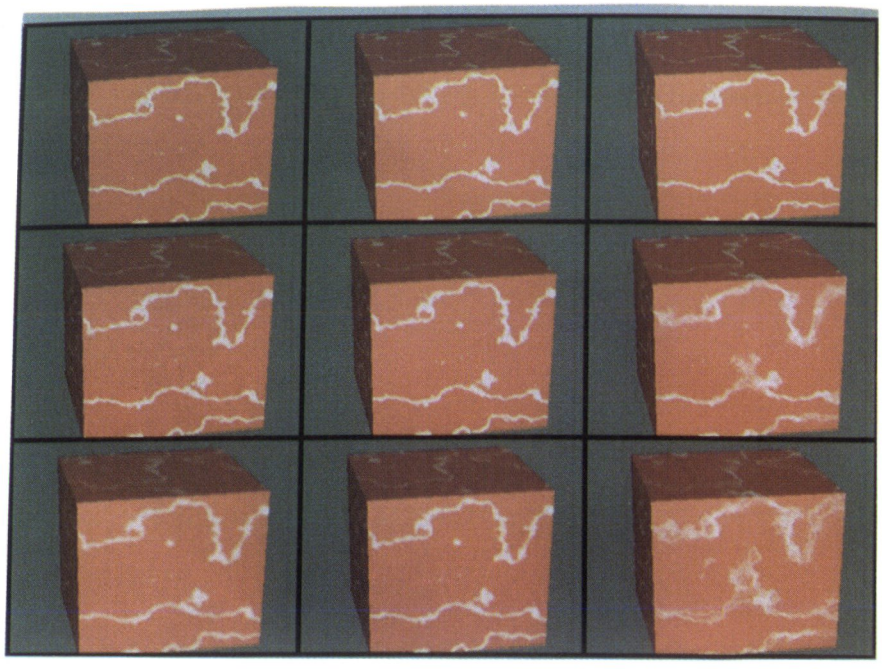

Figure 8: Different filter results