Proceedings of Implicit Surfaces'99, Bordeaux, France, 13-15 Sept. 1999, Editors: John Hughes and Christophe Schlick.

Coupled Modeling of Solid Textures and Implicit Shapes

ERGUN AKLEMAN *
Visualization Laboratory
College of Architecture

JIANER CHEN †
Department of Computer Science
College of Engineering

Texas A&M University

Abstract

In this paper, we present a technique for effective texturing of implicit surfaces. Our technique is based on usage of similar functions for the description of both implicit surfaces and solid textures.

Modeling solid textures requires use of functions with almost unit gradient length. In order to construct such functions, we have discovered two operations which preserve almost unit gradient length property. Based on these operations, we have developed a technique similar to Constructive Soft Geometry for modeling both implicit surfaces and solid textures.

1 Introduction

One of the important problems in implicit modeling is effective texturing the implicit surfaces. Our goal in this paper is to develop a texturing scheme that can effectively provide uniform textures that can conform any orientable 2-manifold shapes.

Figure 1 shows an example of such an irregular topology textured by our method. Note that although the shape is complex, the textures conform well to the varied terrain and maintain a very uniform thickness with no obvious stretch of the textures.

Creating uniform solid textures requires a set of conditions that the functions must satisfy. These conditions require that the gradient length of the functions be less than but almost equal to 1, and we call functions meeting this condition *almost unit gradient length functions.*

A significant contribution of this work is the introduction of a set of binary operations that are closed over almost unit gradient length functions. Since these operations provide approximate set operations, desired functions can be constructed intuitively by the new binary operations. We observe that the new operations are also useful so that shape modeling and the same operations can be used for both shape and texture modeling by constructing the same type of functions.

2 Motivation for Almost Unit Gradient Length

Throughout the paper, we let $\mathcal{F}$ be the set of functions mapping vectors in the 3-dimensional Euclidean space $\mathcal{R}^3$ to real numbers.

Let ω be a positive real number and f be any function in $\mathcal{F}$, we call

$$\frac{1+\cos(\omega f)}{2}$$

*Address: 216 Langford Center, College Station, Texas 77843-3137. email: ergun@viz.tamu.edu. Supported in part by the Texas A&M, Scholarly & Creative Activities Program.

†Address: Department of Computer Science, College Station, TX 77843-3112. email: chen@cs.tamu.edu. Supported in part by the National Science Foundation under Grant CCR-9613805.

Figure 1: Textures over an irregular shape with several handles.

a *texture function* with texture frequency ω and define solid textures in terms of texture functions. Since the texture functions always give a real number between 0 and 1, by using a simple mapping we can describe shading parameters such as diffuse color, opacity or specular color. The following simple mapping from $\mathcal{R}$ to monochrome colors helps to visualize the behavior of a given function f in $\mathcal{R}^2$ as shown in Figure 2:

$$g(f) = \begin{cases} \text{black}, & \text{if } f < 0 \text{ and } (1+\cos(\omega f))/2 < 0.5, \\ \text{light gray}, & \text{if } f < 0 \text{ and } (1+\cos(\omega f))/2 \geq 0.5, \\ \text{dark gray}, & \text{if } f \geq 0 \text{ and } (1+\cos(\omega f))/2 < 0.5, \\ \text{white}, & \text{if } f \geq 0 \text{ and } (1+\cos(\omega f))/2 \geq 0.5, \end{cases}$$

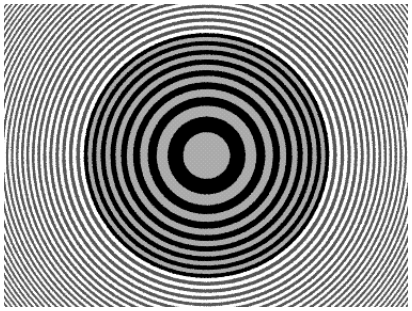

Figure 2: Example of the function $f(v) = v \bullet v - r^2$.

Based on this mapping, each color is assigned to a "ring" (or a "layer") bounded by two implicit curves $f(x,y) = (4k-1)\pi/2\omega$ and $f(x,y) = (4k+1)\pi/2\omega$ for $k = 0, \mp1, \mp2, \ldots$. Since the gradient length represents the rate at which the implicit curves vary, in order to make the thickness of rings (or layers) of the texture uniform, regardless of the value of ω, we would prefer that gradient length $|\nabla f|$ of the function f is approximately equal to 1. If this is the case, the parameter ω directly controls texture frequency. The following examples well illustrate this observation. In Figure 2 the function $f(v) = v \bullet v - r^2$, whose gradient length $|\nabla f| = 2|v|$ is unbounded and increases with $|v|$, does not provide uniform texture. On the other hand, the function $f(v) = |v| - r$, which describes the same implicit curve with $|\nabla f| = 1$, provides a uniform texture as shown in Figure 3 since its gradient length is equal to 1.

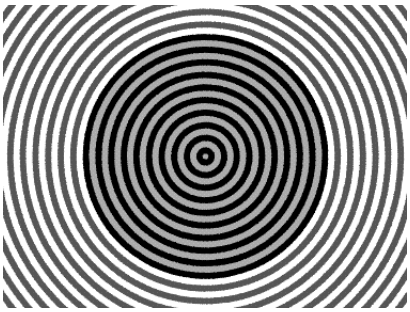

Figure 3: Example of $f(v) = |v| - r$.

We can identify another condition for texture modeling functions by viewing any volume rendering algorithms (such as marching cubes [11, 3] or ray marching [13]) as sampling of the volume which is required to be accurate. If $|\nabla f| \leq 1$, the Nyquist rate [6], which is the required sampling rate for accurate sampling, can be determined directly from the layering frequency ω. On the other hand, if there is no upper limit over the gradient length, there may be aliasing regardless of the choice of sampling frequencies.

Since the two conditions above require the functions whose gradient length is bounded by and approximately equal to 1, the most desirable functions for modeling solid textures are the ones with gradient length exactly equal to 1. Let $\mathcal{T}$ denote the set of unit gradient length functions

$$\mathcal{T} = \{f \in \mathcal{F} \mid |\nabla f(v)| = 1, \ \ \forall v \in \mathcal{R}^3\}.$$

It is known that there is a large variety of functions in the function set $\mathcal{T}$. Examples of functions in $\mathcal{T}$ are the *sphere function* $f(v) = |v - v_0| - r$, which defines a sphere in 3D with radius r and center v_0, and the *plane function* $f(v) = n_1 \bullet (v - v_0)$, which defines a plane in 3D passing through v_0 and with unit normal n_1. The following section presents a method to create new functions with unit gradient length.

3 Unit Gradient Length Preserving Operations

For effective modeling of solid textures, there is a need for a large variety of functions whose gradient length is exactly equal to 1. We have discovered a set of operations that preserve the unit gradient length property. By using these operations a set of essential unit gradient length functions can be obtained. Consider the following functional operation:

$$h[f_1, f_2] = \begin{cases} \sqrt{f_1^2 + f_2^2}, & \text{if } f_1 \geq 0 \text{ and } f_2 \geq 0, \\ f_1, & \text{if } f_1 \geq f_2 \text{ and } f_2 < 0, \\ f_2, & \text{if } f_2 \geq f_1 \text{ and } f_1 < 0, \end{cases}$$

with gradient

$$\nabla h[f_1, f_2] = \begin{cases} \dfrac{f_1 \nabla f_1 + f_2 \nabla f_2}{\sqrt{f_1^2 + f_2^2}}, & \text{if } f_1 \geq 0 \text{ and } f_2 \geq 0, \\ \nabla f_1, & \text{if } f_1 \geq f_2 \text{ and } f_2 < 0, \\ \nabla f_2, & \text{if } f_2 \geq f_1 \text{ and } f_1 < 0. \end{cases}$$

The modeling properties of operation h can be summarized as follows:

- Closure Property:
  - $h[f_1, f_2]$ is a unit gradient length function if f_1 and f_2 are unit gradient length functions and $\nabla f_1 \bullet \nabla f_2 = 0$. We will later illustrate the visual meaning of the condition $\nabla f_1 \bullet \nabla f_2 = 0$. This is straightforward for the case where either f_1 or f_2 or both f_1 and f_2 are negative. We need to show it only for the case where f_1 and f_2 are both positive:

$$\begin{aligned} & \nabla h[f_1, f_2] \bullet \nabla h[f_1, f_2] \\ = \ & \frac{f_1 \nabla f_1 + f_2 \nabla f_2}{\sqrt{f_1^2 + f_2^2}} \bullet \frac{f_1 \nabla f_1 + f_2 \nabla f_2}{\sqrt{f_1^2 + f_2^2}} \\ = \ & \frac{f_1^2 (\nabla f_1 \bullet \nabla f_1)}{f_1^2 + f_2^2} + \frac{f_2^2 (\nabla f_2 \bullet \nabla f_2)}{f_1^2 + f_2^2} \\ & + \frac{2 f_1 f_2 (\nabla f_1 \bullet \nabla f_2)}{f_1^2 + f_2^2} \\ = \ & \frac{f_1^2(1) + f_2^2(1) + 2 f_1 f_2(0)}{f_1^2 + f_2^2} \\ = \ & \frac{f_1^2 + f_2^2}{f_1^2 + f_2^2} \\ = \ & 1. \end{aligned}$$

  - $h[f, -f]$ is a unit gradient length function if f is a unit gradient length function. This property is easy to show since $h[f, -f] = |f|$. We will later illustrate the visual meaning of the operation $h[f, -f]$.

- Commutativity and Associativity:
  - Commutativity: The operation is commutative, i.e., $h[f_1, f_2] = h[f_2, f_1]$.
  - Associativity: The operation is associative, i.e., $h[h[f_1, f_2], f_3] = h[f_1, h[f_2, f_3]]$.

  Because of these two properties, although the operation is a binary operation, instead of nested operations the simple form $h[f_1, \ldots, f_n]$ can be used, without specifying the order of the functions.

- Functional operation $h[f_1, f_2]$ is related to intersection operation as:

$$V(f_1) \cap V(f_2) = V(h[f_1, f_2]).$$

  where $V(f)$ denotes an implicit solid $V(f) = \{v | f(v) \leq 0\}$.

- C^1 continuity:
  $h[f_1, f_2]$ is C^1 continuous everywhere except in $V(h[f_1, f_2])$ (inside and boundary of implicit surface $S(h[f_1, f_2])$). In other words, $S(h[f_1, f_2] - a)$ will not include sharp corners if $a > 0$ where $S(f)$ denotes an implicit surface as $S(f) = \{v | f(v) = 0\}$.

The last two properties of the operation h are particularly useful for intuitive creation of building block functions. Because of the shape property of h, the implicit surface $S(h[f_1,\ldots,f_n])$ can be constructed by using intersection operation. Although $S(h[f_1,\ldots,f_n])$ can have sharp corners $S(h[f_1,\ldots,f_n]-a)$ will not. In other words, $V(h[f_1,\ldots,f_n])$ can be used as a skeleton and $S(h[f_1,\ldots,f_n]-a)$ will be the shape which has minimum distance to the skeleton $V(h[f_1,\ldots,f_n])$.

Let n_1, n_2 be two orthonormal vectors in $\mathcal{R}^2$ and let $f_1(v) = n_1 \bullet (v - v_0)$, $f_2(v) = n_2 \bullet (v - v_0)$. By using operation h over these two functions, it is possible to create simple shapes. Let $f_3 = h[f_1 - a_1, -f_1 - a_1, f_2 - a_2, -f_2 - a_2]$ then $V(f_3)$ is simply a rectangular shape with corners $v_0+a_1v_1+a_2v_2$, $v_0+a_1v_1-a_2v_2$, $v_0-a_1v_1-a_2v_2$, and $v_0-a_1v_1+a_2v_2$. $S(f_3-a_3)$ is the loci of the points which are exactly minimum a_3 distance from the rectangle $V(f_3)$. Note that this approach is similar to Bloomenthal's minimum distance functions from a given skeleton shape [4]. If $a_2 = 0$ the rectangle becomes a line segment and $S(f_3 - a_3)$ becomes the loci of the points which are exactly a_3 distance from the line segment. If both $a_1 = a_2 = 0$, then $V(f_3)$ becomes a point and $S(f_3 - a_3)$ becomes a circle. Moreover, $V(h(f_3 - a_3, -f_3 + a_3))$ is a skeleton in the shape of $S(f_3 - a_3)$ which can be used to describe a toroid in 3D. These examples are shown in Figure 4.

By using the functions in $\mathcal{R}^2$ it is easy to go to $\mathcal{R}^3$. Let $n_3 = n_1 \times n_2$, $f_4 = n_3 \bullet (v - v_0)$, $f_5 = h[f_3 - a_3, f_4 - a_4, -f_4 + a_4]$. Then $S(f_5)$ is simply a rectangular prism, $S(f_5 - a_5)$ defines a variety of ellipsoidal look-a-like shapes as a loci of the points which are equal distance to the prism. By using h functions it is also possible to define toroidal shapes. Let $f_6 = h[f_3 - a_3, -f_3 + a_3, f_4, -f_4]$, then $S(f_6 - a_6)$ is a toroidal shape. If both $a_1 = a_2 = 0$ then the shape is regular toroid. Several examples of 3D building block shapes are shown in Figure 5. The figure includes one regular sphere, one regular toroid, a variety of ellipsoidal shapes and two non-regular toroidal shapes.

Figure 4: Examples of 2D building block functions.

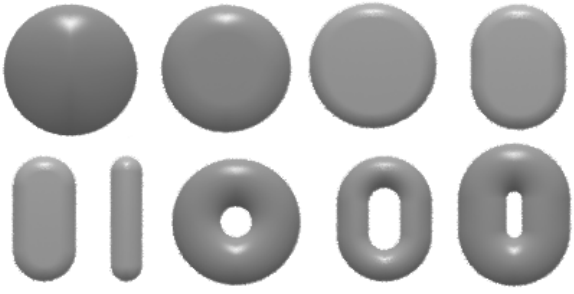

Figure 5: Examples of 3D building block shapes defined by the h operations.

4 Approximate Set Operations

It is important to note that exact union and intersection operations min and max are also closed over $\mathcal{T}$, However, the resulting functions after the application of max and min operations are not C^1. In order to get C^1 continuous functions, we need to relax exact unit gradient length condition and allow functions whose gradient length is bounded by and close to 1.

In our work with unit gradient length functions and operations over them, we have discovered that the operation

$$w_p(f_1, f_2) = \frac{1}{p}\log\left(e^{pf_1} + e^{pf_2}\right)$$

is especially useful, where p is a positive number. This operation provides the approximate set operations

$$\begin{aligned} V(w_p(f_1, f_2)) &\approx V(f_1) \cap V(f_2), \\ V(-w_p(-f_1, -f_2)) &\approx V(f_1) \cup V(f_2), \\ V(w_p(f_1, -f_2)) &\approx V(f_1) - V(f_2). \end{aligned}$$

Moreover, as p becomes large, the operation w_p approaches the min and max operations which are commonly used in shape modeling

$$\begin{aligned} \lim_{p\to\infty} V(w_p(f_1, f_2)) &\longrightarrow V(max(f_1, f_2)) \\ &\quad = V(f_1) \cap V(f_2), \\ \lim_{p\to\infty} V(-w_p(-f_1, -f_2)) &\longrightarrow V(min(f_1, f_2)) \\ &\quad = V(f_1) \cup V(f_2), \\ \lim_{p\to\infty} V(w_p(f_1, -f_2)) &\longrightarrow V(max(f_1, -f_2)) \\ &\quad = V(f_1) - V(f_2). \end{aligned}$$

In addition, the operation w_p satisfies all of the following conditions:

1. Let $\mathcal{N}$ denote the set of functions whose gradient length is bounded by 1: $\mathcal{N} = \{f \in \mathcal{F} \mid |\nabla f(v)| \leq 1 \quad \forall v \in \mathcal{R}^3\}$. Then $\mathcal{N}$ is closed under w_p. In other words, if f_1 and f_2 are in $\mathcal{N}$, the function $w_p(f_1, f_2)$ will also be in $\mathcal{N}$.

2. For high values of p, operation w_p over functions whose gradient length almost equal to 1 creates new functions with almost unit length gradient functions.

3. Operation w_p can construct C^1 functions from C^1 functions.

These properties state that by using w_p and a set of functions whose gradient length is almost 1 or exactly 1 as initial building block functions we can intuitively construct functions whose gradient length is bounded by and close to 1.

Images in Figure 6 show various properties of the w_p operation. The set theoretic representation of the polygon at the top is used to describe the functions by replacing each set operation with related approximate w_p operation. Here building block functions are circle functions that describe the boundaries of the shape at the top. As seen in the figure, the higher p values provide almost unit gradient length. On the other hand, lower p values give smoother shape. In the rest of this section, we show that w_p satisfies Properties 1,2,and 3.

4.1 Closure Property of w_p

Property 1 claims that $\mathcal{N}$ is closed under w_p. It means that the operation w_p over two functions whose gradient length is bounded by 1 always gives a function with a gradient length bounded by 1.

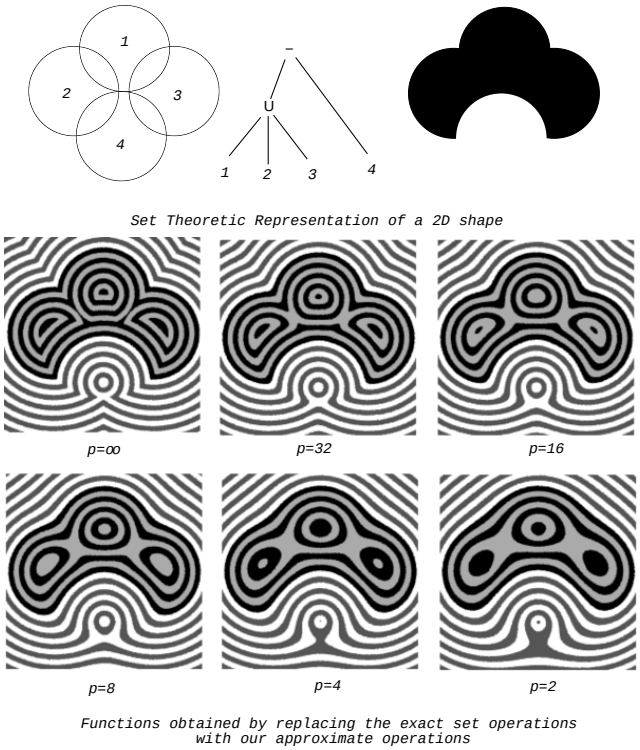

Figure 6: Constructing functions from circle functions and the operation w_p with various p values.

It is easy to show this property. Let $|\nabla f_1| \leq 1$ and $|\nabla f_2| \leq 1$ for $\forall v \in \mathcal{R}^3$, then

$$\begin{aligned}
|\nabla w_p[f_1, f_2]| &= |\nabla \left(\frac{1}{p} \log \left(e^{pf_1} + e^{pf_2} \right) \right)| \\
&= |\frac{e^{pf_1}}{e^{pf_1} + e^{pf_2}} \nabla f_1 + \frac{e^{pf_2}}{e^{pf_1} + e^{pf_2}} \nabla f_2| \\
&= |\frac{e^{pf_1}}{e^{pf_1} + e^{pf_2}} \nabla f_1 \\
&\quad + \left(1 - \frac{e^{pf_1}}{e^{pf_1} + e^{pf_2}} \right) \nabla f_2| \\
&= |a \nabla f_1 + (1-a) \nabla f_2|, \\
&\quad \text{where } \; 0 \leq a = \frac{e^{pf_1}}{e^{pf_1} + e^{pf_2}} \leq 1 \\
&\leq a|\nabla f_1| + (1-a)|\nabla f_2| \\
&\leq 1; \quad \text{for } \forall v \in \mathcal{R}^3.
\end{aligned}$$

4.2 Preservation of Almost Unit Gradient Length

Property 2 claims that for high values of p, the operation w_p over almost unit length gradient functions results in functions with almost unit length gradient. Note that as shown in subsection 4.1, the new gradient vector is the interpolation of the gradient vectors ∇f_1 and ∇f_2 with the interpolation terms $\frac{e^{pf_1}}{e^{pf_1}+e^{pf_2}}$ and $\frac{e^{pf_2}}{e^{pf_1}+e^{pf_2}}$ respectively. For high values of p, even if f_1 and f_2 are only slightly different from each other, one of the interpolation terms becomes almost 0, while the other becomes almost 1 as shown in Figure 7. Thus, if the lengths of ∇f_1 and ∇f_2 are almost 1, the length of the new gradient vector will also be almost 1. This effect is also obvious in the images in Figure 6. These images show that when p becomes smaller, gradient length tends to be shorter. On the other hand, higher p values create almost unit gradient length.

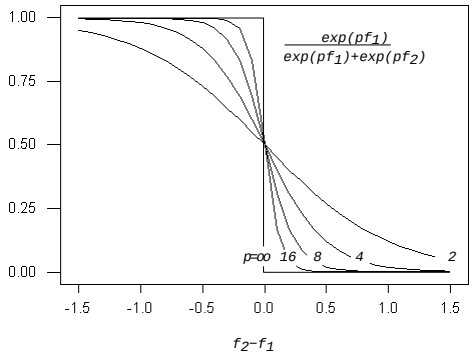

Figure 7: Behavior of Gradient Interpolation Function.

4.3 Continuity Control

Property 3 claims that the operation w_p constructs C^1 functions from C^1 functions. This claim can easily be seen since

$$\nabla w_p[f_1, f_2] = \frac{e^{pf_1}}{e^{pf_1} + e^{pf_2}} \nabla f_1 + \frac{e^{pf_2}}{e^{pf_1} + e^{pf_2}} \nabla f_2,$$

so that ∇w_p is continuous everywhere when f_1, f_2, ∇f_1 and ∇f_2 are continuous.

C^0 functions can also be obtained from C^1 functions by using the operation w_p. In fact, if p goes to ∞, the gradient function becomes discontinuous at $f_1 = f_2$:

$$\nabla max[f_1, f_2] = \begin{cases} \nabla f_1 & \text{if } f_1 > f_2, \\ \nabla f_2 & \text{otherwise.} \end{cases}$$

Note that p values control smoothness. The images in Figure 6 show this control. When p becomes smaller, shapes become smoother. On the other hand, for higher p values, shapes become less smooth.

Entirely similar to above discussion, we can define another operation for approximate union

$$\overline{w}_p(f_1, f_2) = -w_p(-f_1, -f_2)$$

and show that the operation $\overline{w}_p$ also satisfies all properties of w_p.

5 Function Construction

To illustrate our modeling of the shape modeling functions and the texture functions, we first introduce a set of definitions. Let w_p be the binary operator on the function space $\mathcal{F}$ defined in Section 4.

We say that a binary tree T is *ordered* if every non-leaf node in T has exactly two children, which are labeled as *left child* and *right child*, respectively. In particular, the leaves of an ordered binary tree can be uniquely ordered from left to right into a sequence.

Given an ordered binary tree T of m leaves and a binary operator w, we can define inductively a functional T_w from $\mathcal{F}^m$ to $\mathcal{F}$, as follows.

If $m = 1$, then $T_w[f_1] = f_1$. Suppose $m > 1$. Let T' be the subtree in T rooted at the left child of the root of T and let T'' be the subtree in T rooted at the right child of the root of T. Moreover,

assume that the subtree T' has r leaves and the subtree T'' has $m-r$ leaves. Then the function $T_w[f_1, f_2, \ldots, f_m]$ is defined as

$$T_w[f_1, f_2, \ldots, f_m] = w[T'_w[f_1, \ldots, f_r], T''_w[f_{r+1}, \ldots, f_m]]$$

Intuitively, the function $T_w[f_1, f_2, \ldots, f_m]$ can be obtained from the tree T and the operator w as follows: label the m leaves of T by the m functions f_1, f_2, ..., f_m from left to right in this strict order, and label each non-leaf node of T by the operator w. Now each leaf labeled by f_i represents the function f_i while each non-leaf node y with a left children u and a right child v represents the function $w[f_u, f_v]$, where f_u and f_v are the functions represented by the left child u and right child v of the node y. Now the function $T_w[f_1, f_2, \ldots, f_m]$ is the one that is represented by the root of the ordered binary tree T. The functional T_w will be called a *tree functional* (based on the ordered tree T and the binary operator w).

In our modeling, both shape modeling and texture functions will be modeled with two types of trees T_h and $T^{(k)}$, where T_h trees are based on the ordered binary trees and the operation h given in Section 3, and $T^{(k)}$ trees are composition of the ordered binary trees based on the binary operations w_p and $\overline{w}_p$ given in Section 4. The $T^{(k)}$ trees are inductively defined as follows. The $T^{(0)}$ trees are the T_h trees. Each $T^{(k)}$ tree is associated with a positive number p and is given by one of the following three structures:

- A $T^{(k)}$ tree is a T_{w_p} tree whose leaves are labeled by $T^{(i)}$ trees, where $i < k$ (this corresponds to the approximate set intersection);
- A $T^{(k)}$ tree is a $T_{\overline{w}_p}$ tree whose leaves are labeled by $T^{(i)}$ trees, where $i < k$ (this corresponds to the approximate set union); and
- A $T^{(k)}$ tree has its root labeled by $(-, p)$ with two children labeled by two $T^{(i)}$ trees, where $i < k$ (this corresponds to the approximate set difference operation $w_p(f_1, -f_2)$, where f_1 and f_2 are functions represented by $T^{(i)}$ trees with $i < k$).

Based on the results we proved in Section 4, simple induction shows that the functional based on each $T^{(k)}$ tree is closed on the function space $\mathcal{N}$, preserving almost unit gradient length, and having good control on continuity.

$T^{(k)}$ will be used not only to compute the value of the functions, but also their gradients. For the computation of gradient the node operations w_p, $\overline{w}_p$ and h will be replaced by related operations on the gradients which are given in sections 3 and 4 respectively. In the computation of the gradient, inputs of T_h will be the n_i unit vectors which are used to describe the initial plane functions. The value of the vector functional that describes the gradient will similarly computed by the gradient operations given in sections 3 and 4. Note that since the function values obtained by the trees T_h and $T^{(k)}$ need to be used in computing the gradient, the operations used in computation of the gradients are dependent on the position of v. Therefore, although the input vectors are constant, the computed gradient is dependent on the position v.

We use the gradient tree to compute also the diffuse color of any given point. We assign a diffuse color to each initial plane function and these diffuse colors are chosen to be the inputs of T_h. Then, the diffuse color of any given point v is similarly computed as explained in the previous paragraph.

6 Coupled Modeling of Shape and Texture

For the description of shapes and textures, one can use different functions by using completely different tree functionals $T^{(k)}$. However, this approach does not provide much control over texture functions. In order to effectively model texture functions, the local effects need to be controlled.

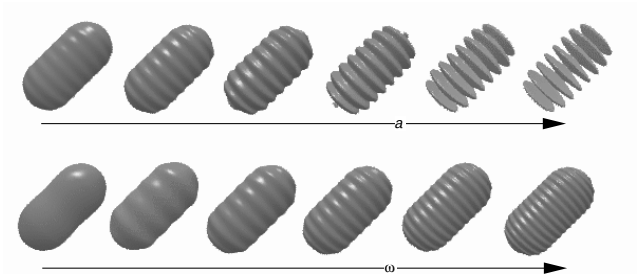

Figure 8: Example of effects obtained by the function given in Equation (1).

We have observed that the best local control can be achieved by using similar functions for both shapes and textures. Let a shape be given by a $T^{(k)}$ tree, we obtain texture functions by changing only the lowest level $T^{(0)} = T_h$ (which we call *building block functions*) and keeping the rest of $T^{(k)}$. The visual effect to be achieved by such replacement can easily be understood by viewing each building block T_h separately.

In order to illustrate the relationship between shape and texture, we use the following function to determine opacity values,

$$f = f_0 + a(cos(\omega f_1) + 1) \tag{1}$$

where coefficient a is a positive real *amplitude.* We define opacity values by the simple transformation

$$opacity[v] = \begin{cases} 1, & \text{if } v \in V(f), \\ 0, & \text{otherwise,} \end{cases}$$

Note that the result of this operation is an implicit surface consisting of a shape described by f_0 and a displacement texture described by f_1. This displacement texture gives a very good understanding of the behavior of texture function f_1. Figure 8 shows how textures can be illustrated by this operation, where we have chosen the shape modeling function f_0 to define the loci of points in $\mathcal{R}^3$ with equal distance to a line segment, and the texture function f_1 to define a plane perpendicular to the line segment. In the top figures the layering frequency ω is held fixed while the amplitude a is varied, and in the bottom figure, the layering amplitude is held fixed while the frequency ω is varied.

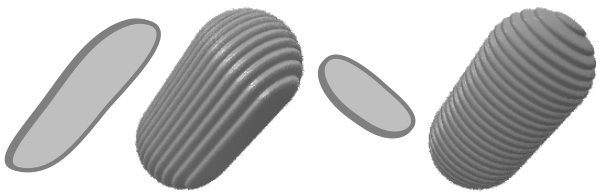

Figure 9: Example of using related building blocks for the shapes and textures.

The examples in Figure 9 show the effects of two different texture functions onto the same shape basic shape. As shown in the figure by using different building blocks textures can be controlled. We have also observed that it is even possible to get additional control by replacing T_h with $T^{(k)}$ type of functions. For instance, the

effect of using an approximate union operation over two $T^{(k)}$ building block functions as texture functions is shown in Figure 10. Examples of more local control of textures are shown in Figure 11.

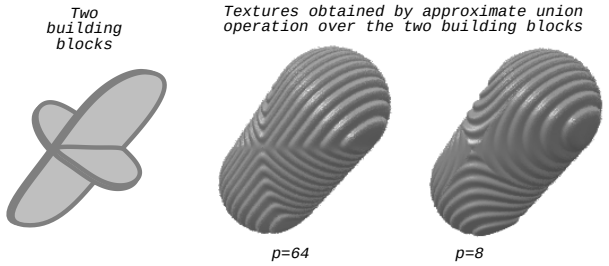

Figure 10: Examples of using approximate union.

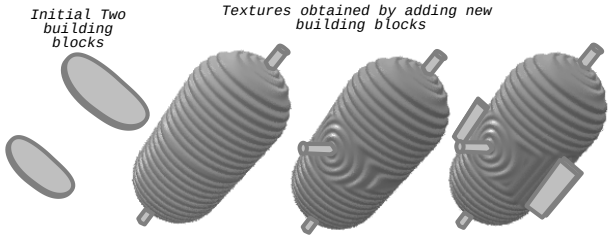

Figure 11: Examples of adding various building blocks.

In order to demonstrate the power of our texture construction method, we model a shape with several holes by using $T^{(k)}$ type of functions. Figures 1, 12 and 13 show texture functions which are obtained by replacing each T_h function with a different T_h function (colored versions of Figures 12 and 13 are also included in this proceedings). The shape in Figure 12 uses a general function in order to create an oil painted look

$$f = f_0 + \sum_{j=1}^{m} a_j(cos(\omega_j f_1) + 1) \qquad (2)$$

where coefficients a_j's are positive real numbers chosen to generate fractal noise.

7 Implementation and Generalization of Our Method

Any function whose gradient length is bounded by 1 satisfies the following Lipschitz condition:

$$|f(v_1) - f(v_2)| \leq |v_1 - v_2| \quad \forall v_1, v_2 \in \mathcal{R}^3.$$

When both functions f_0 and f_1 in Equation (1) in section 6 satisfy the Lipschitz condition, then the corresponding function f also satisfies the Lipschitz condition [15]. Under this condition, the speed of ray tracing and ray marching can be greatly improved [10, 9, 15]. In this paper, we use the sphere tracing method developed by Hart[8, 9].

In rendering these images, for the computation of the pixel opacity, we use *over operation* suggested by Drebin et. al. [5]. For the computation of the color values, we use a derivation of non-photorealistic lighting model suggested by Gooch et. al. [7].

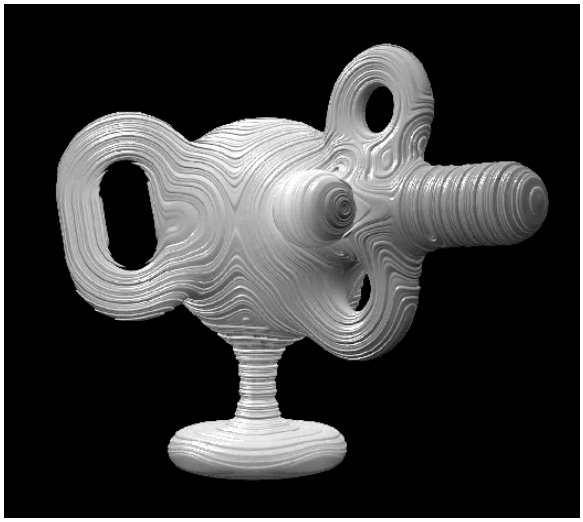

Figure 12: Fractal noise like textures over an irregular shape with several handles.

Figure 13: Textures over an irregular shape.

One advantage of the function based approach is that the normal vectors at every point is defined by the gradient of the function. The gradient of the function given in Equation (2) in section 6 is simply

$$\nabla f = \nabla f_0 - \sum_{j=1}^{m} a_j \omega_j \sin(\omega_j f_1) \nabla f_1$$

where ∇f_1 is computed by tree functional $T^{(k)}$ as explained before.

In another paper, we have introduced the concept of constant time updateability of functions constructed by functional operations [1]. Both operations w_p and h create constant time updateable functions. The constant time updateability property suggests that interactive modeling is possible when special hardware dedicated for rendering volumes is developed.

We also point out that the shapes described by the Ricci implicit formulation [14] $\left(e^{pf_1} + e^{pf_2} + \ldots, e^{pf_m}\right)^{1/p} - 1 = 0$ and the shapes described by our w_p^m operation $w_p(f_1, f_2 \ldots, f_m) = 0$ are exactly the same. It is easy to show that the shapes described by

Blinn's exponential functions [2] can also be exactly represented by functions constructed by w_p. In other words, the operation w_p does not only provide uniform textures and constant time updateability but also describes the same class of shapes that the Ricci and Blinn operations describe.

Tree functional $T^{(k)}$ does not include any deformations, although deformations are considered an important part of Constructive Soft Geometry [17]. We observe that it is possible to develop 3D deformations that satisfy our conditions. One approach might be extending some of the models used for modeling cloth, where surface length metrics tend to remain nearly constant over deformations. By including such deformations, tree functional $T^{(k)}$ can be made more powerful.

In the paper we use only a single texture function that only provides displacement texture. However, we should point out that it is possible to use several texture functions to obtain more complex effects. For instance, paint strokes or blobby particles can be obtained by our approximate intersection operation w_p over two and three textures respectively. The volumetric waves can be obtained simply by supplying a phase variable τ in the texture description, giving $cos(\omega f_1 + \tau)$. Other shader parameters such as diffuse color or surface accessibility [12] can also be determined by texture functions.

We also remark that close coupling of texture functions with shape modeling functions is mainly useful for making the texturing process almost automatic for implicit shapes. However, for user controlled applications such coupling is not really a necessity. In fact, if we do not enforce the coupling, the texture functions can even be used over parametric or polygonal surfaces. Since the operations that create the texture functions are constant time updateable [1] and building block functions are skeleton based, we expect that it is possible to develop user-friendly systems for fast and intuitive texturing parametric or polygonal surfaces. Such a 3D texturing system can especially be useful for animation and movie industries where most of computer graphics objects are hand painted by using 3D painting systems.

8 Conclusion

In the paper, we focused on demonstration of the feasibility of coupled modeling of shapes and textures for better texture design. We have identified that the gradient length of texture functions must be smaller and almost equal to 1. We have also developed a method to construct such texture functions. Since these texture functions can effectively be used for modeling implicit surfaces, we have introduced the concept of coupled modeling of both shapes and textures for creating texture that conforms well with the shapes.

References

[1] E. Akleman and J. Chen, "Constant Time Updateable Operations for Implicit Shape Modeling", *Proceedings of Implicit Surfaces'99,* October 1999 (this proceedings).

[2] J. I. Blinn, "A Generalization of Algebraic Surface Drawing", *ACM Transaction on Graphics*, vol 1, no. 3, pp. 235-256, 1982.

[3] J. Bloomenthal, "Polygonization of Implicit Surfaces", *Computer Aided Geometric Design*, vol. 5, no. 4, pp. 341-355, Nov. 1988.

[4] J. Bloomenthal, "Techniques for Implicit Modeling", In B. Wyvill and J. Bloomenthal eds., *Modeling and Animating with Implicit Surfaces,* ACM SIGGRAPH'90 Course Notes no. 23, August, 1993.

[5] R. A. Drebin, L. Carpenter and P. Hanrahan, "Volume Rendering", *Computer Graphics*, vol 22, no. 3, pp. 65-74, 1988.

[6] A. S. Glassner, *Principles of Digital Image Sythesis,* Volume One, Morgan Kaufman Publishers, Inc., San Fransisco, 1995.

[7] A. Gooch, B. Gooch, P. Shirley and E. Cohen, "A Non-Photorealistic Lighting Model For Automatic Technical Illustration", *Computer Graphics*, vol 32, no. 3, pp. 447-452, 1998.

[8] J. C. Hart, D. J. Sandin and L. H. Kauffman, "Ray Tracing Deterministic 3D Fractals", *Computer Graphics*, vol 23, no. 3, pp. 289-296, 1989.

[9] J. C. Hart, "Sphere Tracing: Simple Robust Antialiased Rendering of Distance-Based Implicit Surfaces", In J. Bloomenthal and B. Wyvill eds., *Modeling, Visualizing and Animating with Implicit Surfaces,* ACM SIGGRAPH'93 Course Notes no. 25, August, 1993.

[10] D. Kalra and A. H. Barr, "Guaranteed Ray Intersections with Implicit Surfaces", *Computer Graphics*, vol 23, no. 3, pp. 297-306, 1989.

[11] W. E. Lorenson and H. E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", *Computer Graphics*, vol 21, no. 3, pp. 163-170, 1987.

[12] G. Miller, "Efficient Algorithms for Local and Global Accessibility Shading", *Computer Graphics*, vol 28, no. 3, pp. 319-326, 1994.

[13] K. Perlin and E. M. Hoffert, "Hypertexture", *Computer Graphics*, vol 23, no. 3, pp. 297-306, 1989.

[14] A. Ricci, "A Constructive Geometry for Computer Graphics", *The Computer Journal*, vol 16, no. 2, pp. 157-160, May 1973.

[15] S. P. Worley and J. C. Hart, "Hyper Rendering of Hyper-Textured Surfaces", *Proceedings of Implicit Surfaces'96,* pp. 99-104, Oct., 1996.

[16] G. Wyvill, C. McPheeters, and B. Wyvill, "Data Structure for Soft Objects", *The Visual Computer,* vol 2, no. 4, pp. 227-234, 1997.

[17] B. Wyvill, A. Guy, and E. Galin, "Extending The CSG Tree: Warping, Blending and Boolean Operations in an Implicit Surface Modeling System", *Proceedings of Implicit Surfaces'98,* pp. 113-121, 1998.