Constant Time Updateable Operations for Implicit Shape Modeling

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 introduce the concept of constant time updateability for the functional operations that can be used for constructing functions. We believe that constant time updateability will be important in the future for guaranteeing interactive modeling of implicit surfaces that are defined by functional construction of functions. We have developed several tests to determine if a functional operator is constant time updateable. In addition, we have introduced constant time updateable exact and approximate functional set operators.

1 Introduction

One of the time consuming process during modeling of implicit surfaces is polygonization of surfaces. Most straightforward approach for polygonization is the sampling of the implicit functions at the corners of discrete grids [14, 8]. We call the corners of discrete grids *sampling points*. Regardless of the structure of discrete grid, an important portion of the computation cost of polygonization comes from computation of function values on these sample points. This cost is simply multiplication of the number of sampling points, k, with the cost of computing function value at a given sample point.

In this paper, we assume that a discrete grid is used for polygonization, the number and the positions of the sample points do not change during construction and the computation of $O(k)$ can be done in interactive speed. Under this assumption, the question we address in this paper is whether interactivity can be guaranteed during implicit shape construction.

We observe that in order to guarantee interactivity, function evaluation in any given sample point must take constant time regardless of shape complexity. If the function evaluation time is not constant, it is not possible to guarantee interactivity. In other words, even if a shape modeling system can provide interactivity for simple shapes, when shapes become complicated, achieving interactivity can become impossible.

We view the complexity of a shape as the number of building block functions that are used to describe the shape. Since each building block function will be evaluated at least once, computation time for function evaluation in a given sample point is at least $O(m)$ where m is the number of building block functions. Then, the overall computation time for function evaluation is at least $O(mk)$ and therefore even if $O(k)$ can be computed in interactive speed for a given value of k, $O(mk)$ may not be computed in interactive speed.

Fortunately, most shape constructions have the following properties:

- typically only constant number of the building block functions is either changed or added or deleted in a given time, and
- the value of the function prior to changing is known.

Images in Figure 1 show a simple shape construction example. An implicit representation of the polygon in the example is given by a set of operations over half-spaces. Each half space is described by a building block function that represents a boundary edge. In this example, the building blocks are affine functions. As shown in the example, the user can either change or add or delete a vertex. As it can easily be seen from the Figure, changing a vertex requires changing two building block functions. Adding a vertex requires adding a new building block function and changing another. Finally, deleting a vertex requires deleting one building function and changing another. In other words, regardless of the number of the edges, each vertex operation effects only a constant number of building blocks. The rest of the building block functions remains unchanged and, therefore, their previous values can be used for the computation of the overall function.

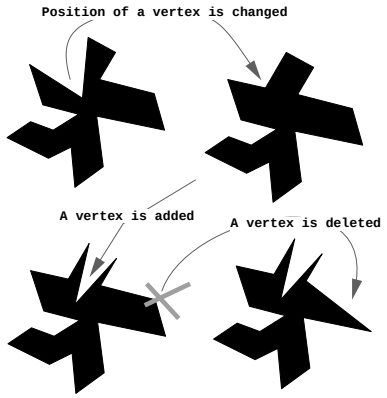


Figure 1: Constructing a polygon.

Since it is possible to use the previous values of unchanged building block functions, for some functions that are constructed by certain types of functional operations, it is possible to compute new function values in constant time. We call this property of functional operations *constant time updateability*. There exist various functional operations used in implicit modeling today. Not all of these functional operations are constant time updateable. It is an important question to introduce ways to identify those functional operations that construct functions that can be updated in constant time during modeling process.

In the rest of this paper, we introduce simple tests to identify whether a functional operation is constant time updateable. Our

* 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.

tests come from our theorems that an operation is constant time updateable if the functional operation is associative and commutative over the function space and inverse function can be computed in constant time.

Based on these tests we develop constant time updateable approximate and exact set operations.

2 Notations

Throughout this paper, we let $\mathcal{F}$ be the set of functions mapping vectors in the 3-dimensional Euclidean space $\mathcal{R}^3$ to real numbers in $\mathcal{R}$. Following the common convention, we denote by $\mathcal{F}^n$ the Cartesian product $\mathcal{F} \times \mathcal{F} \times \cdots \times \mathcal{F}$ of n copies of $\mathcal{F}$, and by $2^{\mathcal{F}}$ the collection of all subsets of $\mathcal{F}$.

In implicit representations, the functions, in general, are from $\mathcal{R}^3$ to $\mathcal{R}$ and implicitly represented shapes are given either as a surface

$$S(f) = \{v \mid f(v) = 0\}$$

or as a solid

$$V(f) = \{v \mid f(v) \leq 0\}.$$

One of the most useful property of implicit representations for shape modeling is that certain binary functional operations on functions provide shape operations on solid shapes. Let $V(f_1)$ and $V(f_2)$ be two implicitly represented solid shapes. Let H be a binary operator on $\mathcal{F}$, that is, given two functions f_1 and f_2 in $\mathcal{F}$, the operator H uniquely determines a new function $H[f_1, f_2]$ in $\mathcal{F}$. Certain binary operations H such as minimum or maximum provide nice operations X such as union or intersection on solid shapes as

$$X(V(f_1), V(f_2)) \approx V(H(f_1, f_2))$$

Note that since the number of building functions is two for the binary operator H, the resulting function $H[f_1, f_2]$ can be computed in constant time, in the sense that for any vector v in $\mathcal{R}^3$, if the values $f_1(v)$ and $f_2(v)$ are given, then the function value $H[f_1, f_2](v)$ can be computed in time $O(1)$.

The user can create an hierarchy of these H operations by defining an ordered binary tree for the description of a complex shape. 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 H on $\mathcal{F}$, we can define inductively a functional T_H from $\mathcal{F}^m$ to $\mathcal{F}$, as follows.

If $m = 1$, then $T_H[f_1] = f_1$. 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 T'' has $m - r$ leaves. Then the function $T_H[f_1, f_2, \ldots, f_m]$ is defined as

$$T_H[f_1, f_2, \ldots, f_m] = H[T'_H[f_1, \ldots, f_r], T''_H[f_{r+1}, \ldots, f_m]]$$

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

In implicit modeling such tree functionals are widely used. The examples of operation H are Rvachev [16] and Ricci operations [19], where the former provides exact set operations with C^1 continuity and latter provides approximate set operations with C^1 continuity.

We will consider a variety of binary operators H on the function space $\mathcal{F}$, and discuss what kind of binary operators H induce tree functionals that support constant time updateability.

3 Constant Time Updateability

As we have discussed in introduction, there are three main operations during construction process: change, add and delete a building block. If an operation is constant time updateable regardless of the number of building block functions and the tree structure, then change, add and delete operations can be done in constant time. We start by the following definitions of constant time updateable operations.

Definition Let H be a binary operation from $\mathcal{F}^2$ to $\mathcal{F}$. The functional H is *constant time updateable* if the following two conditions hold.

1. The functional H is *constant time extendable*: i.e., for any vector $v \in \mathcal{R}^3$, any $m + 1$ functions f_1, ..., f_m, f_0, any index i, $1 \leq i \leq m + 1$, and any m-leaf tree T and $(m + 1)$-leaf tree T', the value

   $$T'_H[f_1, \ldots, f_{i-1}, f_0, f_i, \ldots, f_m](v)$$

   can be computed in constant time from the values $T_H[f_1, \ldots, f_m](v)$ and $f_0(v)$.

2. The functional H is *constant time reducible*: i.e., for any vector $v \in \mathcal{R}^3$, any m functions f_1, ..., f_m, any index i, $1 \leq i \leq m$, and any m-leaf tree T and $(m - 1)$-leaf tree T', the value

   $$T'_H[f_1, \ldots, f_{i-1}, f_{i+1}, \ldots, f_m](v)$$

   can be computed from the values $T_h[f_1, \ldots, f_m](v)$ and $f_i(v)$ in constant time.

Note that the definition only requires adding or deleting functions in constant time and does not include changing a function in constant time. The following lemma shows that changing a function in constant time can be easily derived from the definition of constant time updateability.

Lemma 3.1 *Let H be constant time updateable. Then for any vector $v \in \mathcal{R}^3$, and for any functions f_1, ..., f_m, f'_i, $1 \leq i \leq m$, and any m-leaf tree T, the value*

$$T_H[f_1, \ldots, f_{i-1}, f'_i, f_{i+1}, \ldots, f_m](v)$$

can be computed from the values $f_i(v)$, $f'_i(v)$, and $T_h[f_1, \ldots, f_{i-1}, f_i, f_{i+1}, \ldots, f_m](v)$ in constant time.

PROOF. Let T' be any tree of $m - 1$ leaves. Since the operation H is constant time updateable, we have

$$\begin{aligned} & T_H[f_1, \ldots, f_{i-1}, f_i, f_{i+1}, \ldots, f_m](v) \\ \Longrightarrow \quad & T'_h[f_1, \ldots, f_{i-1}, f_{i+1}, \ldots, f_m](v) \\ \Longrightarrow \quad & T_h[f_1, \ldots, f_{i-1}, f'_i, f_{i+1}, \ldots, f_m](v) \end{aligned}$$

The first $\Longrightarrow$ is constant time computable since by our assumption, the operator H is constant time reducible and the value $f_i(v)$ is given, and the second $\Longrightarrow$ is constant time computable since by

our assumption the operator H is constant time extendable and the value $f_i'(v)$ is given. This proves that the value

$$T_H[f_1, \ldots, f_{i-1}, f_i', f_{i+1}, \ldots, f_m](v)$$

can be computed from the value

$$T_h[f_1, \ldots, f_{i-1}, f_i, f_{i+1}, \ldots, f_m](v)$$

and values $f_i(v)$, and $f_i'(v)$ in constant time. □

We say that the operator H is *associative* if it satisfies the associativity law:

$$H[H[f_1, f_2], f_3] = H[f_1, H[f_2, f_3]]$$

for all functions f_1, f_2, and f_3 in $\mathcal{F}$. We say H is *commutative* if it satisfies the commutativity law:

$$H[f_1, f_2] = H[f_2, f_1]$$

for all f_1 and f_2 in $\mathcal{F}$.

If H is both commutative and associative, then for any two ordered binary trees T and T' of the same number of leaves, the functional T_H is equal to the functional T_H'. In other words, if the operator H is both associative and commutative, then the tree structure for a tree functional is irrelevant. Therefore, a tree functional T_H on a tree T of m leaves and an associative and commutative operator H can be simply written as H^m.

Theorem 3.2 *Let H be an associative and commutative operator. Then H is constant time extendable.*

PROOF. Let T_H be a tree functional based on the operator H and an ordered binary tree T of m leaves and and T_H' be a tree functional based on the operator H and an ordered binary tree T' of $m+1$ leaves. Since the operator is associative and commutative, the tree functional T_H can be written as H^m and the tree functional T_H' can be written as H^{m+1}.

Consider any $m+1$ functions $f_1, \ldots, f_m$, and f_0, and any index i, $1 \leq i \leq m+1$. Let v be any fixed vector in $\mathcal{R}^3$, and suppose that the values $H^m[f_1, \ldots, f_m](v)$ and $f_0(v)$ are known. We have

$$\begin{aligned} & H^{m+1}[f_1, \ldots, f_{i-1}, f_0, f_i, \ldots, f_m](v) \\ = \; & H[H^m[f_1, \ldots, f_{i-1}, f_i, \ldots, f_m], f_0](v) \\ = \; & H[H^m[f_1, \ldots, f_{i-1}, f_i, \ldots, f_m](v), f_0(v)] \end{aligned}$$

The last formula can be computed in constant time. This proves that the associative and commutative operator H is constant time extendable. □

This theorem gives a practical indicator for testing if an operator H is constant time extendable since both associativity and commutativity of an operation H can easily be checked.

Corollary 3.3 *Suppose that an associative and commutative operator H is constant time reducible. Then the operator H is constant time updateable.*

The constant time reducibility of an operator can be verified using a variety of methods. For example, the following theorem is a convenient method for checking constant time reducibility of operators.

Theorem 3.4 *Let the associative and commutative operator H have an identity element f_u over $\mathcal{F}$ and for any given function f in $\mathcal{F}$ there exists an inverse function f^{-1} (i.e. $(\mathcal{F}, H)$ forms an Abelian group), such that for any vector v in $\mathcal{R}^3$, the value $f^{-1}(v)$ can be computable from $f(v)$ in constant time. Then H is constant time reducible.*

PROOF. Let $f_1, \ldots, f_m$ be m functions in $\mathcal{F}$ and suppose that the values $H^m[f_1, \ldots, f_m](v)$ and $f_i(v)$ are given, where $1 \leq i \leq m$ and v is a vector in $\mathcal{R}^3$. We first compute the value $f_i^{-1}(v)$. By the assumption, this can be done in constant time. Now we have

$$\begin{aligned} & H^{m-1}[f_1, \ldots, f_{i-1}, f_{i+1}, \ldots, f_m](v) \\ = \; & H[H^{m-1}[f_1, \ldots, f_{i-1}, f_{i+1}, \ldots, f_m], f_u](v) \\ = \; & H^m[f_1, \ldots, f_{i-1}, f_u, f_{i+1}, \ldots, f_m](v) \\ = \; & H^m[f_1, \ldots, f_{i-1}, H[f_i, f_i^{-1}], f_{i+1}, \ldots, f_m](v) \\ = \; & H^{m+1}[f_1, \ldots, f_{i-1}, f_i, f_i^{-1}, f_{i+1}, \ldots, f_m](v) \\ = \; & H[H^m[f_1, \ldots, f_{i-1}, f_i, f_{i+1}, \ldots, f_m](v), f_i^{-1}(v)] \end{aligned}$$

Since the values $H^m[f_1, \ldots, f_m](v)$ and $f_i^{-1}(v)$ are both known, and since the binary operator H is computable in constant time, we conclude that the value

$$H^{m-1}[f_1, \ldots, f_{i-1}, f_{i+1}, \ldots, f_m](v)$$

can be computed from the values $H^m[f_1, \ldots, f_m](v)$ and $f_i(v)$ in constant time. That is, the operator H is constant time reducible. □

Figure 2 shows the relationship between the theorems. As it can easily be seen from Figure 2, if $(\mathcal{F}, H)$ is an Abelian group then H is constant time updateable. For example, the addition operation which is used for combining Blobby shapes [21, 22] creates an Abelian group over function space, so it is constant time updateable. However, Abelian condition is too restrictive to check the constant time updateability of more complicated operations. Instead we may use Corollary 3.3 and check directly constant time reducibility property along with associativity and commutativity. An example of complicated constant time updateable operations is the Minkowski operations.

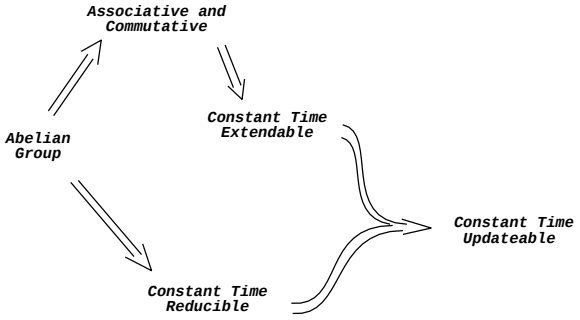


Figure 2: The relationship between theorems.

3.1 Minkowski Operations

We define *Minkowski operations* as $H_p[f_1, f_2] = (|f_1|^p + |f_2|^p)^{1/p}$ [15]. This operation is particularly interesting for shape modeling since it provides approximate union and intersection of shapes (positive p are for intersection, negative p are for union [1, 4]). Positive ray-linear functions are closed under Minkowski operation [1]. Ray-linear functions are a special case that requires a limited number of samples for polygonization (namely, in the range of $100,000$). Therefore interactive polygonization is possible for

the implicit surfaces defined by these functions. The first author of this paper used the constant time updateability property of the Minkowski operation to develop a software for guaranteed interactive construction of ray-linear functions (ray-linear functions f describe star shapes in the implicit form $S(f-1)$) [1, 3].

It is easy to see that the Minkowski operator H_p is associative and commutative. In order to show that the operator H_p is constant time reducible, we have

$$\begin{aligned}
& ((H_p^m[f_1,\ldots,f_m](v))^p - |f_i(v)|^p)^{1/p} \\
= \;& ((|f_1(v)|^p + \cdots + |f_m(v)|^p)^{1/p})^p - |f_i(v)|^p)^{1/p} \\
= \;& (|f_1(v)|^p + \cdots + |f_m(v)|^p - |f_i(v)|^p)^{1/p} \\
= \;& (|f_1(v)|^p + \cdots + |f_{i-1}(v)|^p + |f_{i+1}(v)|^p \\
& + \cdots + |f_m(v)|^p)^{1/p} \\
= \;& H_p^{m-1}[f_1,\ldots,f_{i-1},f_{i+1},\ldots,f_m](v)
\end{aligned}$$

Therefore, given the values $H_p^m[f_1,\ldots,f_m](v)$ and $f_i(v)$, the value

$$H_p^{m-1}[f_1,\ldots,f_{i-1},f_{i+1},\ldots,f_m](v)$$

can be computed in constant time. That means the operator H_p is constant time reducible. By Corollary 3.3, any tree functional based on Minkowski operator H_p and an ordered binary tree is constant time updateable.

4 Approximate Set Operations

An important operation which is widely used in implicit surface modeling is Ricci's approximate set operations [19]. These operations smooth out the sharp corners resulted from exact set operations and provide smoothly blended versions of shapes constructed by exact set operations. The binary versions of Ricci operations are given as $H[f_1,f_2] = (e^{pf_1} + e^{pf_2})^{1/p}$. As it can easily be checked that this operation is commutative but, unfortunately, not associative. Therefore, neither Corollary 3.3 nor Theorem 3.4 is applicable. We have currently discovered a new approximate set operation that satisfies Corollary 3.3. This binary operation which we call w_p is given by

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

where p is a positive number. This operation w_p, plus a simple variation w_p' of w_p, provides the approximate set operations similar to Ricci operations

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

As p becomes large, the operation w_p approaches the max operation which is commonly used in shape modeling

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

It is easy to see that the operation w_p is constant time updateable by checking its associativity, commutativity and constant time reducibility.

1. The operation w_p is associative:

$$\begin{aligned}
& w_p[w_p[f_1,f_2],f_3] \\
= \;& \frac{1}{p}\log\left(e^{p\frac{1}{p}\log(e^{pf_1}+e^{pf_2})} + e^{pf_3}\right) \\
= \;& \frac{1}{p}\log\left(e^{pf_1} + e^{pf_2} + e^{pf_3}\right) \\
= \;& \frac{1}{p}\log\left(e^{pf_1} + e^{p\frac{1}{p}\log(e^{pf_2}+e^{pf_3})}\right) \\
= \;& w_p[f_1,w_p[f_2,f_3]].
\end{aligned}$$

2. The operation w_p is commutative:

$$\begin{aligned}
w_p[f_1,f_2] &= \frac{1}{p}\log\left(e^{pf_1} + e^{pf_2}\right) \\
&= \frac{1}{p}\log\left(e^{pf_2} + e^{pf_1}\right) \\
&= w_p[f_2,f_1].
\end{aligned}$$

3. The operation w_p is constant time reducible:

$$\begin{aligned}
& \frac{1}{p}\log(e^{p(w_p^m[f_1,\ldots,f_m](v))} - e^{pf_i(v)}) \\
= \;& \frac{1}{p}\log(e^{p(\frac{1}{p}\log(e^{pf_1(v)}+\cdots+e^{pf_m(v)})} - e^{pf_i(v)}) \\
= \;& \frac{1}{p}\log(e^{pf_1(v)} + \cdots + e^{pf_m(v)} - e^{pf_i(v)}) \\
= \;& \frac{1}{p}\log(e^{pf_1(v)} + \cdots + e^{pf_{i-1}(v)} + e^{pf_{i+1}(v)} \\
& \qquad + \cdots + e^{pf_m(v)}) \\
= \;& w_p^{m-1}[f_1,\ldots,f_{i-1},f_{i+1},\ldots,f_m](v)
\end{aligned}$$

We also would like to point out that the shapes described by the Ricci implicit formulation [19] $\left(e^{pf_1} + e^{pf_2} + \cdots + e^{pf_m}\right)^{1/p} - 1 = 0$ and the shapes described by our w_p operation $w_p(f_1,f_2\ldots,f_m) = 0$ are exactly the same

$$\begin{aligned}
\left(e^{pf_1} + e^{pf_2} + \cdots + e^{pf_m}\right)^{1/p} - 1 &= 0 \quad \longleftrightarrow \\
\left(e^{pf_1} + e^{pf_2} + \cdots + e^{pf_m}\right)^{1/p} &= 1 \quad \longleftrightarrow \\
\log\left(e^{pf_1} + e^{pf_2} + \cdots + e^{pf_m}\right)^{1/p} &= 0 \quad \longleftrightarrow \\
\frac{1}{p}\log\left(e^{pf_1} + e^{pf_2} + \cdots + e^{pf_m}\right) &= 0 \quad \longleftrightarrow \\
w_p(f_1,f_2,\ldots,f_m) &= 0.
\end{aligned}$$

In other words, the operation w_p does not only provide constant time updateability property but also describes the same class of shapes that the Ricci operation [19] describes. It can also be proven that the shapes described by our w_p operation includes all the shapes that can be created by Blinn's exponentials [7].

Images in Figure 3 show various properties of the w_p operation. The set theoretic representation of the convex polygon at the top is used to describe the functions by replacing intersection operation with related approximate w_p operation. Here building block functions are affine functions which are in the form of $f_i = n_i \bullet (v - v_i)$ where n_i is unit normal to the line and v_i is any point on the line. These line functions describe the boundaries of the convex polygon at the top. In order to show the behavior of the functions, we

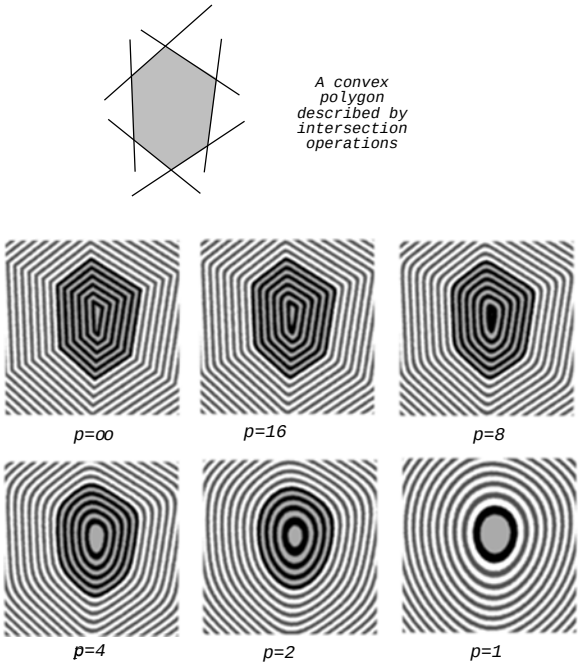


Figure 3: Pseudocoloring the functions that are obtained by replacing the intersection operations with functional approximate intersection operations w_p with various p values and by replacing half spaces with affine building block functions.

convert the function values to monochrome colors by using the following pseudocoloring operation:

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

As seen in the figure, the higher p values provide more precise approximation for the intersection, while lower p values give smoother shape.

Similarly, it can be shown that approximate union operation $w'_p[f_1, f_2] = -w_p[-f_1, -f_2]$ is also constant time updateable. However, the approximate set difference operation $w_p[f_1, -f_2]$ is neither commutative nor associative. Therefore, its constant time updateability cannot be derived directly from Corollary 3.3 or Theorem 3.4.

5 Exact Set Operations

An important exact set operation is Rvachev operations that are introduced in Solid Modeling by Shapiro, Pasko, Adziev, Sourin and Savchenko [18, 16]. Rvachev [16] used

$$w[f_1, f_2] = \frac{1}{1+\alpha}(f_1 + f_2 + \sqrt{f_1^2 + f_2^2 - 2\alpha f_1 f_2})$$

where $\alpha = \alpha(f_1, f_2)$ is an arbitrary continuous function that satisfies the following conditions $-1 < \alpha(f_1, f_2) < 1$ and $\alpha(f_1, f_2) = \alpha(f_2, f_1) = \alpha(-f_1, f_2) = \alpha(f_1, -f_2)$. If we choose $\alpha = 1$, $w[f_1, f_2]$ becomes maximum operator. This function operation provides exact set operation as

$$\begin{aligned} V(w[f_1, f_2]) &= V(f_1) \cap V(f_2), \\ V(-w[-f_1, -f_2]) &= V(f_1) \cup V(f_2), \\ V(w[f_1, -f_2]) &= V(f_1) - V(f_2). \end{aligned}$$

These equalities are not exactly correct but they are very close approximations. Most desired property of Rvachev operation is that it gives C^1 continuous functions everywhere except at $\{v \mid f_1(v) = f_2(v) = 0\}$ [16]. Unfortunately, Rvachev operation is not associative. Thus, it is unclear whether Rvachev operation is constant time updateable. We have discovered a new piecewise C^1 exact set operation h_p that is commutative, associative and constant time updateable.

$$h_p[f_1, f_2] = \begin{cases} (f_1^p + f_2^p)^{1/p} & \text{if } f_1 \geq 0 \text{ and } f_2 \geq 0 \\ f_1 & \text{if } f_1 \geq 0 \text{ and } f_2 < 0 \\ f_2 & \text{if } f_1 < 0 \text{ and } f_2 \geq 0 \\ -(|f_1|^{-p} + |f_2|^{-p})^{-1/p} & \text{if } f_1 < 0 \text{ and } f_2 < 0 \end{cases}$$

The function $-\left(|f_1|^{-p} + |f_2|^{-p}\right)^{-1/p}$, at the first look, seems to go to ∞ when either f_1 or f_2 or both goes to 0. However, this first impression is not correct since the function is in fact equal to $\frac{-f_1 f_2}{(|f_1|^p + |f_2|^p)^{1/p}}$ and instead of going to ∞ it approaches to 0 when either f_1 or f_2 or both goes to 0. Similar to Rvachev operation , the operation h_p almost satisfies the following equalities and provides exact set operations: :

$$\begin{aligned} V(h_p[f_1, f_2]) &= V(f_1) \cap V(f_2), \\ V(h'_p[f_1, f_2]) = V(-h_p[-f_1, -f_2]) &= V(f_1) \cup V(f_2), \\ V(h_p[f_1, -f_2]) &= V(f_1) - V(f_2). \end{aligned}$$

Moreover, similar to Rvachev operation, the operation h_p is C^1 continuous everywhere except at $\{v \mid f_1(v) = f_2(v) = 0\}$ for p values larger than 2.

Our goal is to show h_p is constant time updateable. It is easy to see that h_p is commutative. For associativity, we have

$$h_p[h_p[f_1, f_2], f_3] = \begin{cases} (f_1^p + f_2^p + f_3^p)^{1/p} \\ \qquad \text{if } f_1 \geq 0 \text{ and } f_2 \geq 0 \text{ and } f_3 \geq 0 \\ (f_1^p + f_2^p)^{1/p} \\ \qquad \text{if } f_1 \geq 0 \text{ and } f_2 \geq 0 \text{ and } f_3 < 0 \\ (f_1^p + f_3^p)^{1/p} \\ \qquad \text{if } f_1 \geq 0 \text{ and } f_2 < 0 \text{ and } f_3 \geq 0 \\ (f_2^p + f_3^p)^{1/p} \\ \qquad \text{if } f_1 < 0 \text{ and } f_2 \geq 0 \text{ and } f_3 \geq 0 \\ f_1 \\ \qquad \text{if } f_1 \geq 0 \text{ and } f_2 < 0 \text{ and } f_3 < 0 \\ f_2 \\ \qquad \text{if } f_1 < 0 \text{ and } f_2 \geq 0 \text{ and } f_3 < 0 \\ f_3 \\ \qquad \text{if } f_1 < 0 \text{ and } f_2 < 0 \text{ and } f_3 \geq 0 \\ -\left(\sum_{i=1}^{3} (-f_i)^{-p}\right)^{-1/p} \\ \qquad \text{if } f_1 < 0 \text{ and } f_2 < 0 \text{ and } f_3 < 0 \end{cases}$$

It is easy to verify that $h_p[f_1, h_p[f_2, f_3]] = h_p[h_p[f_1, f_2], f_3]$. Therefore, the operation h_p is associative. Thus h_p is constant time extendable.

Based on the associativity property, it is not very difficult to ver-

ify that the operation h_p over m functions can be formulated as

$$h_p^m[f_1,\ldots,f_m](v) = \begin{cases} \left(\sum_{f_j(v)\geq 0} f_j^p(v)\right)^{1/p} \\ \qquad \text{if } \exists j \text{ with } f_j(v) \geq 0 \\ -\left(\sum_{j=1}^{m} |f_j(v)|^{-p}\right)^{-1/p} \\ \qquad \text{if } \forall j,\ f_j(v) < 0 \end{cases}$$

In order to show the constant time reducibility of the function $h_p^m[f_1,\ldots,f_m]$, for each vector v, we keep a triple

$$(h_p^+[f_1,\ldots,f_m](v), h_p^-[f_1,\ldots,f_m](v), n^+[f_1,\ldots,f_m](v)) \tag{1}$$

where

$$h_p^+[f_1,\ldots,f_m](v) = \left(\sum_{f_j(v)\geq 0} (f_j(v))^p\right)^{1/p},$$

$$h_p^-[f_1,\ldots,f_m](v) = \left(\sum_{f_j(v)<0} (f_j(v))^{-p}\right)^{-1/p},$$

and $n^+[f_1,\ldots,f_m](v)$ is the number of functions f_j such that $f_j(v) \geq 0$. Note that all functions $h_p^+[f_1,\ldots,f_m](v)$, $h_p^-[f_1,\ldots,f_m](v)$, and $n^+[f_1,\ldots,f_m](v)$ are commutative and associative. Moreover, the function h_p^m can be rewritten as

$$h_p^m[f_1,\ldots,f_m](v) = \begin{cases} h_p^+[f_1,\ldots,f_m](v) \\ \text{if } n^+[f_1,\ldots,f_m](v) > 0 \\ \\ h_p^-[f_1,\ldots,f_m](v) \\ \text{if } n^+[f_1,\ldots,f_m](v) = 0 \end{cases}$$

We now show how to compute in constant time the triple

$$\begin{array}{l} (h_p^+[f_1,\ldots,f_{i-1},f_{i+1},\ldots,f_m](v), \\ h_p^-[f_1,\ldots,f_{i-1},f_{i+1},\ldots,f_m](v), \\ n^+[f_1,\ldots,f_{i-1},f_{i+1},\ldots,f_m](v)) \end{array} \tag{2}$$

from the triple in (1) and the value $f_i(v)$.

If $f_i(v) \geq 0$, then

$$\begin{aligned} & h_p^+[f_1,\ldots,f_{i-1},f_{i+1},\ldots,f_m](v) \\ = \; & ((h_p^+[f_1,\ldots,f_m](v))^p - (f_i(v))^p)^{1/p} \\ & h_p^-[f_1,\ldots,f_{i-1},f_{i+1},\ldots,f_m](v) \\ = \; & h_p^-[f_1,\ldots,f_m](v) \\ & n^+[f_1,\ldots,f_{i-1},f_{i+1},\ldots,f_m](v) \\ = \; & n^+[f_1,\ldots,f_m](v) - 1 \end{aligned}$$

On the other hand, if $f_i(v) < 0$, then

$$\begin{aligned} & h_p^+[f_1,\ldots,f_{i-1},f_{i+1},\ldots,f_m](v) \\ = \; & h_p^+[f_1,\ldots,f_m](v) \\ & h_p^-[f_1,\ldots,f_{i-1},f_{i+1},\ldots,f_m](v) \\ = \; & -((-h_p^-[f_1,\ldots,f_m](v))^{-p} - (f_i(v))^{-p})^{-1/p} \\ & n^+[f_1,\ldots,f_{i-1},f_{i+1},\ldots,f_m](v) \\ = \; & n^+[f_1,\ldots,f_m](v) \end{aligned}$$

As a result, the triple in (2) can always be computed from the triple in (1) in constant time. This gives the illustration that the operation h_p is constant time reducible (in a more general sense). According to Corollary 3.3, the operation h_p is constant time updateable.

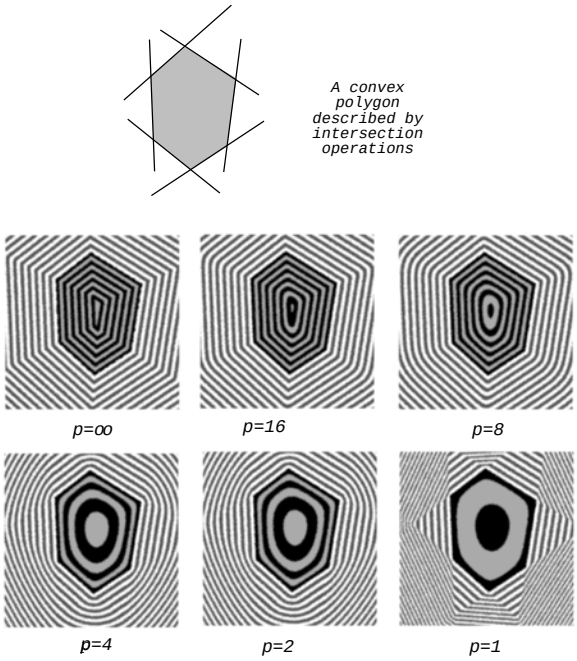


Figure 4: Pseudocoloring the functions that are obtained by replacing the intersection operations with functional exact intersection operation h_p with various p values and by replacing half spaces with affine building block functions.

Figure 4 shows functions described by the h_p operation. The images in Figure 4 were created in a similar way as we used to create the images in Figure 3. The operation h_p can define distance functions [9, 10] under some restrictions [5]

In a similar way, we can show that exact union operation $h'_p[f_1, f_2] = -h_p[-f_1, -f_2]$ is also constant time updateable. However, whether the exact set difference operation $h_p[f_1, -f_2]$ is constant time updateable is not clear since the operation is neither commutative nor associative.

6 General Trees

Up to this point, we assume that only one operation is used to describe the shapes. Earlier examples show only approximate or exact functional intersection operations. Intersection operations can only create convex shapes from convex building block shapes. In order to create more complicated shapes, we need to use various operations in the same tree. The example in Figure 5 shows a simple non-convex polygon described by two types of set operations. The Peterson style formula [17] (a monotone formula with no half-spaces appearing more than once) of CSG representation of the polygon is constructed by using the algorithm proposed by Dobkin, Guibas, Hersberger and Snoeyink [11].

Figure 6 shows the functions described by replacing set operations with related constant time updateable operations. In general, a tree T of m leaves in which each non-leaf node is associated with a function operation can naturally define a functional operation from $\mathcal{F}^m$ to $\mathcal{F}$ when we associate each leaf of T with a basic block function in $\mathcal{F}$. Such a function is called a *tree function.*

Figures 7 and 8 give some colored 3D shapes described by general tree functions based on union and difference by the exact set operation h_p.

Assume that all operations associated with a tree function are constant time updateable. Then it is easy to see that the updating time for the tree function is dependent only on the depth of the tree while independent of the number of basic block functions. Of course in the worst case the depth of the tree can be as large as the number of basic block functions. However, in most cases the depth of the trees is much smaller.

In order to illuminate the amount of speed up that can be achieved by using constant time updateable operations, we use a simple example. Consider a tree whose every node is a functional given by a tree functional T_H. Let us assume that the tree is balanced with a height of M and each level is given by an index i between 1 and M. Let us also assume that in level i each node has exactly m_i children. For updating such a tree there exist three strategies: recompute values, reuse old values for updating, and constant time updateability. Recompute is the computation of function without using old values. Reuse is the updating of function by using old values. Constant Time Updateability is using only constant time updateable operations and computation of the function by using old values. The following table shows a comparison of these strategies for computing the functions described by the example tree in any sample point regarless of its position. For a general grid with k number of samples these numbers have to be multiplied by k.

Recompute values		Reuse old values		Constant Time Updateable	
Time	Space	Time	Space	Time	Space
$\prod_{i=1}^{M} m_i$	0	$\sum_{i=1}^{M} m_i$	$\prod_{i=1}^{M} m_i$	M	$\prod_{i=2}^{M} m_i$

As it can be seen from the table the recomputation of the values does not use any space but it can be extremely time consuming. Therefore, it is not a proper approach for interactive shape modeling.

Reusing the old values makes the speed much faster but it still depends on the complexity of the shape. Moreover, the memory used to store the values can be significant.

Using constant time updateable operations seems to improve both time and space complexity significantly. Constant time updateable operations make the computation independent of the complexity of the shape and the computation time is $(\sum_{i=1}^{M} m_i)/M$ times faster than reusing the old values. Moreover, when using constant time updateable operations, we do not need to store the values of building block functions f_i's. Therefore, the space requirement is $\prod_{i=2}^{M} m_i$. In other words, constant time updateable case uses m_1 times less memory than reusing the old values.

The speedup and memory reduction depend on the values of m_i's. In 3D, if the construction starts from convex shapes, m_1's can generally be larger than 6. A hexahedron requires 8 building blocks, an icosahedron is the intersection of 20 building block functions that represent faces, and a soccer ball requires 32 building blocks. If the shape is given by the union of convex shapes, m_2 represents the number of convex shapes and it can be extremely large. Therefore, the value of $\sum_{i=1}^{M} m_i$ is in general much larger than M.

7 Final Remarks

As mentioned earlier, a system for guaranteed interactive shape construction for ray-linears [1] and ray-quadrics [3] has already been developed based on constant time updateable Minkowski operations. For ray-linears we used trees of depth 2 (approximate union of approximate intersections) and for ray-quadrics we used trees of depth 3 (approximate set difference of approximate unions of approximate intersections), respectively. An important property of ray-linears and ray-quadrics is that for the polygonization it is only necessary to sample the surface of a sphere. This property limits the number of samples to be used for effective polygonization: for achieving an acceptable quality, the samples in the range of $100,000$ are enough. For $100,000$ samples, polygonization permits interactive shape construction regardless of the number of building blocks on an SGI O2.

For the polygonization of general implicit surfaces, the number of samples need to be at least one order of magnitude larger than $100,000$ somewhere in the range of $10,000,000$. Although, we cannot currently achieve interactivity for $10,000,000$ samples, we expect that in the near future, interactivity will be possible. Then, reusing the old values with constant time updateable operations will guarantee interactivity for complicated shapes that are described by the trees with bounded depth.

We have recently observed that the computation time can be dractically reduced by updating only a subset of sample points. An interesting research direction can be the identification of these subsets of sample points during shape construction. In the current framework, constant time updateable operations are useful only when the positions of sample points do not change during construction. We also plan to investigate function updating when the positions of constant number of sample points change.

Before we close this paper, we would like to also point out that the operations h_p and w_p have another property which is important for shape modeling. Let gradient lengths of f_1 and f_2 be bounded by a constant, then the gradients of $h_p[f_1, f_2]$ and $w_p[f_1, f_2]$ are also bounded by a constant [5]. In other words, the functions constructed by h_p and w_p satisfy Lipschitz condition. Therefore, the resulting shapes described by such functions can be efficiently ray traced [6, 12, 13, 20]. The colored Figures 7 and 8 are rendered using sphere tracing [12, 13, 20] which is based on Lipschitz condition. The 3D shapes described by functions that are constructed by approximate set operations are illustrated in [5].

References

[1] E. Akleman, "Interactive Construction of Smoothly Blended Star Solids", *Proceedings of Graphical Interface'96*, pp 159-167, May, 1996.

[2] E. Akleman, "Ray-Quadrics", *Proceedings of Implicit Surfaces'96*, pp 89-98, Oct, 1996.

[3] E. Akleman, "Interactive Construction of Ray-Quadrics", *Proceedings of Implicit Surfaces'98*, pp 105-114, June, 1998.

[4] E. Akleman and J. Chen, "Generalized Distance Functions", *Proceedings of Shape Modeling'99*, pp 72-79, March, 1999.

[5] E. Akleman and J. Chen, "Coupled Modeling of Solid Textures and Implicit Shapes", *Proceedings of Implicit Surfaces'99*, October, 1999 (this proceedings).

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

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

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

[9] 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.

[10] J. Bloomenthal and K. Shoemake “Convolution Surfaces”, *Computer Graphics*, vol 25, no. 4, pp. 251-256, August 1991.

[11] D. Dobkin, L. Guibas, J. Hersberger and J. Snoeyink “An Efficient Algorithm for Finding the CSG Representation of a Simple Polygon”, *Computer Graphics*, vol 22, no. 4, pp. 31-40, August 1988.

[12] 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.

[13] 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.

[14] 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.

[15] D. S. Mitrinovic, *Analytic Inequalities*, Springer-Verlag, New York, 1970.

[16] A. Pasko, V. Adzhiev, A. Sourin and V. Savchenko, “Function Representation in Geometric Modeling: Concepts, Implementations and Applications”, *Visual Computer*, no. 11, pp. 429-446, 1995.

[17] D. Peterson, “Halfspace Representation of Extrusions, Solids of Revolution, and Pyramids”, *Sandia Report*, Sand84-0572, Sandia National Laboratories, 1984.

[18] V. Shapiro, “Real Functions for Representation of Real Solids”, *Computer Aided Geometric Design*, no. 11, pp. 153-157, 1994.

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

[20] S. P. Worley and J. C. Hart, “Hyper Rendering of Hyper-Textured Surfaces”, *Proceedings of Implicit Surfaces’96*, pp. 99-104, Oct., 1996.

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

[22] 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.

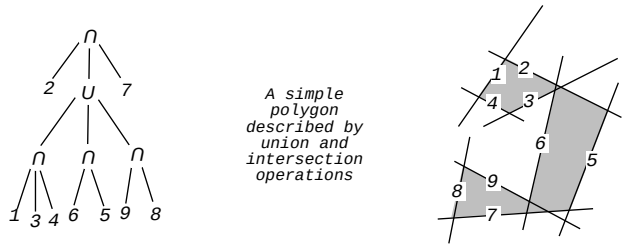
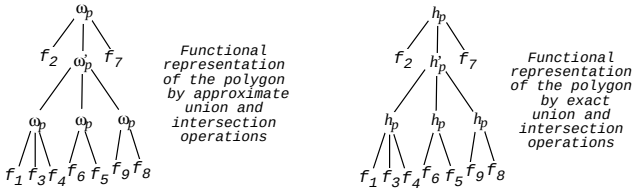


Figure 5: A simple polygon described by union and intersection operations.



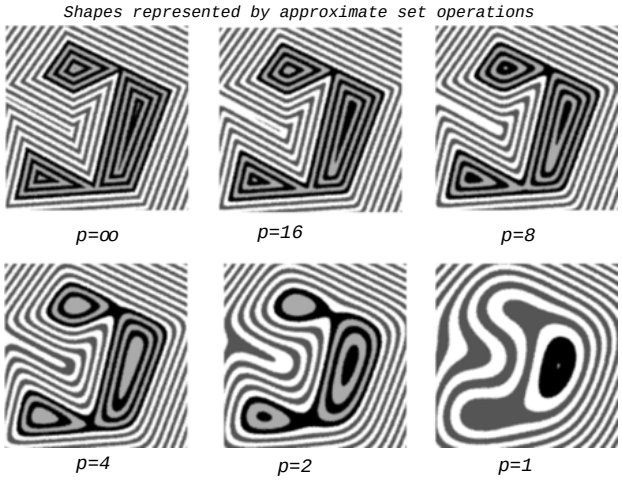


Figure 6: Pseudocoloring the functions that are obtained by replacing the union and intersection operations with functional operations w_p and h_p with varying p values and by replacing half spaces with affine building block functions.