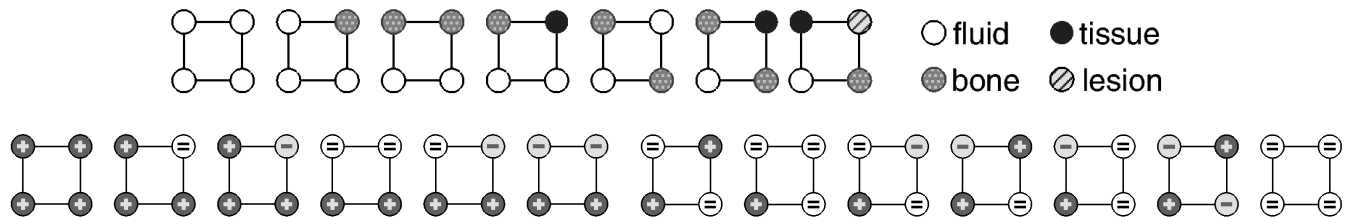# Counting Cases in Marching Cubes:
Toward a Generic Algorithm for Producing Substitopes

David C. Banks
Florida State University

Stephen Linton
University of Saint Andrews

Distinct cases of colorings for a square, assigning one color to each vertex.
Top row: seven cases result from using four colors (fluid, bone, tissue, lesion) when applying Separating Surfaces to a square.
Bottom row: thirteen cases result from using three colors (+ − =) when applying Marching Cubes to a square.

Abstract

We describe how to count the cases that arise in a family of visualization techniques, including Marching Cubes, Sweeping Simplices, Contour Meshing, Interval Volumes, and Separating Surfaces. Counting the cases is the first step toward developing a generic visualization algorithm to produce substitopes (geometric substitutions of polytopes). We demonstrate the method using a software system ("GAP") for computational group theory. The case-counts are organized into a table that provides a taxonomy of members of the family; numbers in the table are derived from actual lists of cases, which are computed by our methods. The calculation confirms previously reported case-counts for large dimensions that are too large to check by hand, and predicts the number of cases that will arise in algorithms that have not yet been invented.

CR Categories: G.2 [Discrete Mathematics]: Combinatorics—Counting problems; G.4 [Mathematical Software]: Algorithm Design and Analysis;

Keywords: level set, isosurface, orbit, group action, Marching Cubes, separating surfaces, geometric substitution, substitope.

1 The MC family of algorithms

The Marching Cubes (MC) algorithm was presented by Lorensen and Cline in 1987 [Lorensen and Cline 1987] as an exhaustive-search algorithm that generates a level set (isosurface) of a scalar function f. The algorithm iterates over each cube tesselating a compact subvolume of $\mathbb{R}^3$ on which the function f is defined.

The sign of $f(v_i) - c$ is evaluated at the eight vertices v_i of a cube, where c is some user-defined constant (the isovalue). Neglecting the degenerate case where the sign is exactly zero, each of the eight vertices can be in one of two states: negative or positive (black or white). These produce $2^8 = 256$ patterns. Many of these patterns turn out to be equivalent under the symmetries of the cube (such as rotation or mirror-reflection). Other patterns are equivalent under reversal of colors (for example, all-black being equivalent to all-white). Through patient brute-force organization of the 256 patterns, one discovers there to be fourteen or fifteen equivalence classes of the colorings. Among the fifteen cases is a chiral pair that are mirror images of each other, so these two are equivalent if orientation is ignored.

In the MC algorithm, the pattern of a given cube is matched to one of these fourteen or fifteen cases (via a look-up table), and a pre-determined arrangement of polygons is fitted to meet the constraint $f(p) - c = 0$ for points p within the cube. In other words, the cube is replaced by zero or more triangles approximating the level set. Examples are illustrated in figure 1, showing one of the geometric substitutions in a 2-simplex, a 3-simplex, a 2-cube, and a 3-cube.

1.1 Variations on MC

Since its original publication, MC has inspired numerous modifications and extensions. These variations suggest that a family of algorithms exists, whose members are distinguished by a few key parameters. A selection of these variations is surveyed below.

Variation of the shape. If a 3-simplex (tetrahedron), rather than a 3-cube, tiles the domain, then the scalar function is evaluated at only four vertices. Bloomenthal presented this approach in 1988 [Bloomenthal 1988], and Shen and Johnson called it "Sweeping Simplices" in their 1995 paper [Shen and Johnson 1995]. One advantage of using tetrahedra rather than cubes is that the analysis is simpler: only three cases arise for the vertex colorings of a tetrahedron, rather than fourteen for a cube.

Variation of the dimension. The two-dimensional version of MC is popularly called "Marching Squares," which provides a simple motivation for the three dimensional case. Although the algorithm is unpublished, it can easily be derived. One can find many descriptions of Marching Squares by searching the World Wide

Web. It is described, for example, in slides for a course on Data Visualization by Rheingans at UMBC, on Computer Graphics by Pfenning at Carnegie Mellon, on Computergraphik by Hanisch at Universität Tübingen, on Advanced Graphics by Dodgson at the University of Cambridge, on Informatik in der Medizin by Gaugler at Universität Karlsruhe, and many others.

When the MC algorithm is extended to dimension $n = 4$, two problems arise. First, the number of vertex patterns is large (65,536), so enumerating them all by hand is unrealistic. Second, it becomes quite difficult to perform the mental rotations to determine when two color patterns of a 4-cube are equivalent. As Lorensen and Cline pointed out in the case of the 3-cube, "triangulating the 256 cases is possible but error prone. ... we reduced the problem to 14 patterns by inspection" (page 165). Although this approach to counting cases works for Marching Squares, it does not scale to higher dimensions.

Recently, several researchers have tackled the case-counting problem for the four-dimensional case in different ways. In 1996, Weigle and Banks [Weigle and Banks 1996] demonstrated a technique (Contour Meshing) that divides the 4-cube into 4-simplexes. They observed that counting cases for vertex colorings is much simpler for the 4-simplex than for the 4-cube, and described how to substitute zero or more 3-simplexes to approximate a level set within a 4-simplex being traversed. In 1999, Roberts and Hill [Roberts and Hill 1999] counted 272 cases for the 4-cube by computation, numerically tagging equivalent cases. In 2000, Bhaniramka, Wenger, and Crawfis [Bhaniramka et al. 2000] followed a similar approach, announcing the existence of 222 cases for the 4-cube.

Variation of the shape's symmetry. The symmetries of a figure are due to transformations that preserve its shape and, perhaps, orientation. If one considers orientation (clockwise versus a counter-clockwise) to be irrelevant, one loses distinctions between certain colorings of squares. So the number of cases depends in part on the choice one makes when considering symmetries of the shape. The original MC deals with two cases that are "chiral," that is, not equivalent to their mirror images under the orientation-preserving symmetries of the cube. Accordingly the problem reduces to fifteen cases under orientation-preserving symmetries and to fourteen cases under the larger group which includes mirror reflection.

An analogous situation exists in every dimension – the group of orientation-preserving symmetries is, in general, only half as large as the group of all symmetries. So there may be figures that are equivalent under the larger group but not the smaller one. This is the reason that Roberts and Hill found a larger number of cases than Bhaniramka, Wenger, and Crawfis (272 versus 222 cases) –
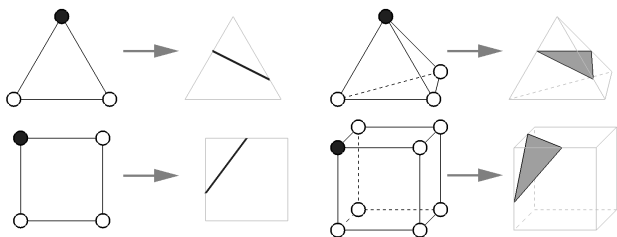
they were admitting a smaller symmetry group.

Variation of the number of colors. The works described above all share the goal of producing a level set of a scalar-valued function. But in 1997 Nielson and Sung showed that this strategy of counting cases and using pre-computed geometry can be used for other purposes. In their "Interval Volumes" [Nielson and Sung 1997], they generated subvolumes of a domain corresponding to the locus of points x satisfying $x : a < f(x) < b$. In the arena of computational geometry, this subvolume is represented by the Boolean intersection $L_a \cap L_b$, where L_a and L_b are the subvolumes in which $a < f(x)$ and $f(x) < b$, respectively. Previously this constructive solid geometry (CSG) problem had been approached in a totally different way by Thibault and Naylor in 1987 using a binary space partitioning (BSP) tree [Thibault and Naylor 1987]. Nielson and Sung's novel insight was that three discrete situations or "colors" can prevail at a vertex (i.e., $f < a$; $a < f < b$; $b < f$), leading to fifteen cases of vertex colorings of a tetrahedron. Figure 2 (top row) shows an example of the geometric substitution in Interval Volumes for a 2-simplex and a 3-simplex. They could have further reduced the cases, as Lorensen and Cline did, by treating as equivalent any two figures whose coloring schemes are reversed, i.e., changing the symmetry of the colors. Hege and colleagues also considered using multiple colors, which they describe in two technical reports [Hege et al. December 1997] [Stalling et al. December 1998]; their implementation is the basis of the "amira" visualization tool's GMC module [AMI n. d.], which has been used to simulate rat dissection in a virtual environment [Montgomery et al. 2001].

Weigle and Banks also considered the effect of changing the number of colors. They discussed, but did not enumerate, the cases where the function is exactly zero at a vertex of a simplex, representing a third "color." In Marching Squares, this third color leads to thirteen cases. These are illustrated at the top of this paper (bottom row of figure). Until now there has been no published case count for MC with this third color included; we calculate the solution and report it in section 5.

Variation of the colors' symmetry. In 1997, Nielson and Franke presented a technique for generating a separating surface [Nielson and Franke 1997]. A separating surface is the boundary between subvolumes, each of which has a discrete color or type. For a 3-simplex, it suffices to consider four available colors for the four vertices, Nielson and Franke treated as equivalent any two vertex colorings where the colors of one figure are a permutation of the colors of the other. For example, a tetrahedron with two vertices of $color_1$ and two of $color_2$ is equivalent to the case of two vertices of $color_3$ and two of $color_4$. Their paper lists four of the five possible cases (the remaining case being the trivial case where all vertices
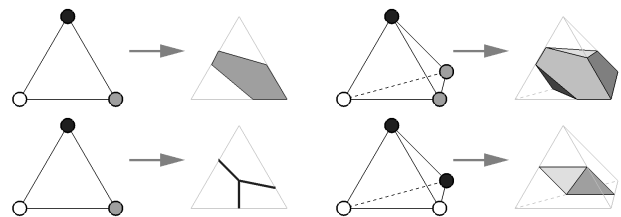

Figure 1: Examples of geometric substitution rules in Marching Cubes, generalized to n-simplexes and n-cubes, for $n \in [2..3]$. The colors correspond to sign of $f(v_i) - c$ at each vertex such as black for negative and white for positive. Upper left: 2-simplex replaced by line segment. Upper right: 3-simplex replaced by triangle. Lower left: 2-cube replaced by line segment. Lower right: 3-cube replaced by triangle.

Figure 2: Examples of geometric substitution rules. For Interval Volumes (upper row), Colors denote intervals such as white for (-∞, a), gray for (a, b), and black for (b, ∞). Left: 2-simplex replaced by line segment. Right: 3-simplex replaced by triangles. For Separating Surfaces (lower row), colors denote set membership such as fluid, bone, tissue, lesion. Left: three-colored 2-simplex replaced by line segments. Right: two-colored 3-simplex replaced by triangles.

have the same color). Figure 2 (bottom row) shows an example of geometric substitution in Separating Surfaces for a 2-simplex and a 3-simplex. Lorensen and Cline reduced the number of cases by considering equivalences induced by the reversal of color. By comparison, Nielson and Franke considered color symmetry to include not just reversal of an ordered set of colors, but all possible permutations of colors. The important role of groups acting on shapes and colors is noted in Hege's abstract [Hege May 25-29, 1998] from a seminar at Dagstuhl.

1.2 Generic Marching Cubes

The techniques surveyed in section 1.1 share a basic approach but vary in detail. The basic approach is as follows.

1. A polytope (whether a cube or a tetrahedron or a 4-cube or a 4-simplex) in some domain is inspected.

2. Each vertex v_i is assigned a color $f(v_i)$, as dictated by some user interaction such as moving a slider bar to select a different isovalue.

3. (Optional) The polytope coloring is matched to representative case via a look-up table.

4. Geometric substitution is performed, replacing the polytope with some other geometry, *e.g.*, to represent an isosurface.

Geometric Substitution. Geometric substitution was used by Lindenmayer in 1971 [Lindenmayer 1971] and by Prusinkiewicz in 1990 [Prusinkiewicz and Lindenmayer 1990] to model natural shapes; Glassner used geometric substitution to create complex shapes [Glassner 1992]. Geometric substitution was used to simplify polygonal meshes by Lounsbery, DeRose, and Warren [Lounsbery et al. 1997] and by Kobbelt, Campagna, and Seidel [Kobbelt et al. 1998], whose figure 1 shows the explicit use of a geometric substitution rule. So geometric substitution is by no means exclusive to MC. There is no commonly used name for polytopes that result from geometric substitution of other polytopes; we propose calling them *substitopes*.

Weigle and Banks demonstrated with Contour Meshing that the recursive nature of substitopes permits an MC-style technique to be applied repeatedly to a dataset: they reduced the dimension from four to three to two, generating surfaces in $\mathbb{R}^4$.

Acceleration Schemes. Some, but not all, of the variations on MC pre-compute a look-up table, which serves as an acceleration technique when the geometric substitution is applied; the geometric substitution can also be performed procedurally.

Another way to accelerate the algorithm is to skip over the trivial substitutions rather than to employ an exhaustive traversal of the domain. Traversing the domain is the most inefficient portion of these algorithms, because in practice most polytopes are replaced by the empty set. The performance of MC improves considerably when a spatial data structure is available that delivers the subdomain containing only polytopes for which the geometric substitution is non-trivial. Shen, Hansen, Livnat, and Johnson showed in 1997 how a hierarchical data structure vastly improves the speed of the algorithm by spending most of the computation on the non-trivial replacements [Shen et al. 1997]. Their work built on previous work by Wilhems and Van Gelder that imposed octrees on the spatial domain [Wilhelms and Gelder 1992], and work by Gallagher [Gallagher 1991] that inverted the spatial database to support queries based on the value of the scalar field, which were incorporated into Sweeping Simplices.

Parameters needed for counting cases. As the variations listed in section 1.1 suggest, there are five key parameters that determine the number of cases that arise for colored polytopes. The number of cases is independent of the actual geometric substitution that is employed. The five parameters are

1. the symmetry applied to the polytope;
2. the symmetry applied to the colors;
3. the choice of polytope from the set {n-simplex, n-cube};
4. the dimension n of the polytope; and
5. the number k of colors.

The remainder of this paper explains how to count cases by using group theory: orbits of groups acting on sets are enumerated using a computational algebra package. Section 2 describes the aspects of group theory that are required for solving the case-counting problem. Section 3 describes how a tool for computational group theory can be programmed to count cases and shows the results of the calculations organized into a table. Section 5 indicates where various Marching-Cubes-style algorithms fit into this new taxonomy.

2 Action of a Group on a Set

For years mathematicians have studied problems similar to counting cases of polytope colorings. In order for us to apply their results we first convert the problem of counting cases in various visualization algorithms into the appropriate mathematical language. This task requires the use of group theory, described briefly below.

The theory of groups owes its name to a paper published in 1854 by Arthur Cayley [Cayley 1854], "On the theory of groups." A group is a set with a binary operation satisfying four criteria:

1. the set is closed under the operation;
2. the operation obeys the associative law;
3. the set has an identity element (denoted by the symbol 1); and
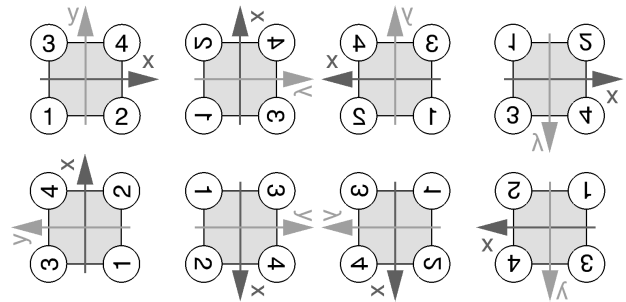4. each element has an inverse.

comment	$\hat{g}$	$\hat{g}(\mathbf{s})$	g
identity	()	(v_1, v_2, v_3, v_4)	()
swap	$(x\ y)$	(v_1, v_3, v_2, v_4)	(2 3)
flip	$(x\ \text{-}x)$	(v_2, v_1, v_4, v_3)	(1 2)(3 4)
flip	$(y\ \text{-}y)$	(v_3, v_4, v_1, v_2)	(1 3)(2 4)
swap flip	$(x\ \text{-}x)(x\ y)$	(v_3, v_1, v_4, v_2)	(1 2 4 3)
swap flip	$(y\ \text{-}y)(x\ y)$	(v_2, v_4, v_1, v_3)	(1 3 4 2)
swap flip flip	$(y\ \text{-}y)(x\ \text{-}x)(x\ y)$	(v_4, v_2, v_3, v_1)	(1 4)
flip	$(y\ \text{-}y)(x\ \text{-}x)$	(v_4, v_3, v_2, v_1)	(1 4)(2 3)

Figure 3: Permutations acting on axes and vertices. Column $\hat{g}$ gives permutations of x and y axes. Column $\hat{g}(\mathbf{s})$ shows the effect of a permutation on vertices of the square $\mathbf{s}=(v_1, v_2, v_3, v_4)$. Column g gives corresponding permutations of vertices. Permutations are written as cycles, acting by composition from right to left.

Figure 4: Permutations from figure 3 acting on the x and y axes and on the square $\hat{\mathbf{s}}$. Top row: first four permutations applied to $\hat{\mathbf{s}}$. Bottom row: next four permutations applied to $\hat{\mathbf{s}}$. Note that half of the permutations preserve orientation, one on the top row and three on the bottom.

Often the appearance of the binary operation is suppressed, so $a * b$ is written as ab, and $a * a$ is written as a^2. More details about groups can be found in textbooks on modern algebra, such as the popular one by Fraleigh [Fraleigh 1998]. Familiar examples of groups include integers with the addition operation, and rational numbers (without zero) under multiplication.

In creating the table for MC, Lorensen and Cline produced a set of 256 cube colorings. Then they considered the action of a symmetry group on the 256 cube colorings. The group operation is composition: a permutation (of vertices and colors) composed with another permutation is again a permutation, satisfying requirement (1) of a group, namely, closure. One can readily determine that permutations also meet the other three criteria for being a group.

A group acts on a set X by mapping it to itself in a particular kind of way. The requirements of a group action are given below.

Definition. A group G is said to **act on a set** X if (1) the identity fixes every element of X *i.e.*, $1x = x$, and (2) the associative law holds; *i.e*, $(g_2 g_1)x = g_2(g_1 x)$, where $1, g_1, g_2 \in G$, and $x \in X$. (**Note:** some authors apply actions from the right rather than the left, thus writing $x g_1 g_2$.)

Example. The *symmetric group S_2* of all permutations of coordinates x and y acts on $\mathbb{R}^2$.

$$S_2 = \{(x \rightarrow x, \ y \rightarrow y), \ (x \rightarrow y, \ y \rightarrow x)\}$$

The first (identity) element leaves the x and y coordinates fixed; the second element sends x to y and y to x, producing a reflection about a diagonal line. Both actions preserve the shape of an axis-aligned square centered at the origin.

The usual convention when writing a permutation is to list the cycles it induces on elements of the set. For example, the permutation $(x \rightarrow y, \ y \rightarrow x)$ sends x to y which goes to x; the permutation is denoted by the cycle $(x \ y)$. The identity mapping is, by convention, denoted $()$ rather than $(x)(y)$, and trivial cycles like (x) and (y) are suppressed when the permutation is written out. Thus the symmetric group on two letters is the set with two permutations: the identity, written $()$, and the swap, written $(x \ y)$.

The shape of the square is also preserved by the action of mirror reflections (flips) exchanging x with $-x$ or y with $-y$. These flip groups contain the permutations $\{(), (x \ -x)\}$ and $\{(), (y \ -y)\}$; each flip group is equivalent (isomorphic) to the group S_2. Their direct product contains all four combinations of flip operations.

$$S_2 \times S_2 = \{ \ (), \quad (x \ -x), \quad (y \ -y), \quad (x \ -x)(y \ -y) \ \}$$

These four operations correspond to the identity, a flip of the x-axis, a flip of the y-axis, and flips of both axes.

2.1 Group Acting on the Set of Vertices

The full set of symmetries on an n-cube is the wreath product (written $\wr$) of a flip with the permutations. We let *shapeGroup* represent the symmetry group acting on a polytope, so *shapeGroup* $= S_2 \wr S_n$ for the cube. The wreath product is too complicated to describe here; for its definition see the algebra textbook by Cohn [Cohn 1984].

One particular geometric incarnation of a square is $\hat{s}$, which has vertices labeled as $v_1 = (-1, -1)$, $v_2 = (1, -1)$, $v_3 = (-1, 1)$, and $v_4 = (1, 1)$, corresponding to the lower left, lower right, upper left, and upper right vertices of a square centered at the origin. Squares, for the purpose of counting cases, result from any action of *shapeGroup* on this reference square $\hat{s}$. This observation is formalized below.

Definition. A standard square is the tuple $\hat{s} = (v_1, v_2, v_3, v_4)$ and its image under any action of *shapeGroup*. That is, **s** is a standard square if and only if $\mathbf{s} = g\hat{s}$ for some $g \in$ *shapeGroup*.

Example. The element $(x \ -x)$ of *shapeGroup* acts on the square, flipping it in the x direction. So $(x \ -x)(\hat{s}) = (v_2, v_1, v_4, v_3)$.

Although we defined *shapeGroup* in terms of its actions on the plane (in particular, its actions on the positive and negative axes), we would prefer to think of it in terms of its actions on vertices. In the example above, the flip $(x \ -x)$ puts vertex v_1 into the second slot and puts v_2 into the first slot, since negating the x coordinates swaps the bottom two vertices. By looking at the tuple on the right hand side, one can deduce what permutation acted on $\hat{s}$: an out-of-position vertex must have been permuted. So if v_i is put into position j, then the permutation maps $i \rightarrow j$.

Example. Under the action of $(x \ -x)$ on the plane, vertex v_1 in the square **s** moves to position 2 and vertex v_2 moves to position 1. Likewise, vertices v_3 and v_4 swap positions in the tuple. The group element $(x \ -x)$ can be re-labeled accordingly in terms of its effect on the vertices of $\hat{s}$, namely $(1 \ 2)(3 \ 4)$.

This re-labeling is important in section 3, which describes how the computational algebra package "GAP" can create *shapeGroup* automatically.

All eight actions of *shapeGroup* are listed in figure 3. In the left-most column of the table in figure 3, element $\hat{g} \in$ *shapeGroup* is written in terms of the coordinates x and y. The middle column shows its action on the square $\hat{s} = (v_1, v_2, v_3, v_4)$. The right hand column re-names the group element as g, which acts on the vertices of the square.

2.2 Group Acting on the Set of Colors

In counting cases for MC, we see that one group acts on the vertices of a square by moving them around; another group acts on the set of colors by permuting them. We call the second group *colorGroup*. A vertex v_i in a square can be labeled with two symbols + and - or, equivalently, can be marked with two colors $color_1$ and $color_2$ to indicate the sign of $f(v) - c$, where c is the isovalue. The color of vertex v_i is determined by a coloring function χ which maps vertices to colors. If $\hat{\sigma}$ is a permutation on the colors, then $color_i$ is mapped to the color $\hat{\sigma}(color_i)$. The notation is simplified if we use the permutation σ that maps one color *index* to another color *index*. Thus

$$\hat{\sigma}(color_i) = color_{\sigma(i)}$$

Example. The permutation $\sigma = (1 \ 2)$ acts as follows on the color indexes 1 and 2.

$$(1 \ 2) \ (1) = 2 \qquad \textit{i.e., } \sigma \textit{ of 1 is 2}$$

$$(1 \ 2) \ (2) = 1 \qquad \textit{i.e., } \sigma \textit{ of 2 is 1}$$

So there is one group (*i.e., shapeGroup*) that acts on the vertices of a square, and another (*i.e., colorGroup* $= S_2$) that acts on the colors. Together they act on the combinatorial set of all $2^4 = 16$ colorings of the square. The next section describes this action.

2.3 Group Acting on the Set of Colorings

Having defined actions on vertices and on colors, we can now define a group action on colored vertices. It is convenient to write $\chi(v_i)$ as χ_i, suppressing the v, so that a coloring of the square can be written in the compact form given below.

Definition. A **coloring** of the square is the 4-tuple of colors $(\chi_1, \chi_2, \chi_3, \chi_4)$ and its permutations by *shapeGroup* $\times$ *colorGroup*.

Examples. Suppose $color_1$ is purple and $color_2$ is orange. The coloring $(1, 1, 1, 1)$ has all purple vertices. The coloring $(1, 1, 2, 2)$ has purple for the bottom two vertices and orange for the top two.
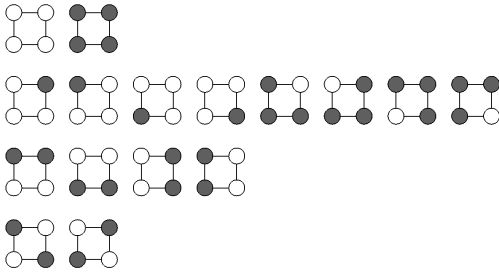
Figure 5: Orbits of *shapeGroup* × colorGroup acting on colorings of the square. In each row, any coloring can be mapped to any other via the action of some element (g, σ) of coloringGroup.

The coloring (1, 2, 1, 2) has purple on the left side and orange on the right.

The direct product *shapeGroup* × *colorGroup* acts on a coloring in the obvious way: an element of *shapeGroup* shuffles the *order* of the four colors, and an element of *colorGroup* permutes the *value* of the colors. We call this product *coloringGroup*. An element h of *coloringGroup* has the form (g, σ), where g acts on vertices and σ acts on colors.

Example. The action of $((2\ 3), \hat{\sigma})$ on a coloring is

$$((2\ 3), \hat{\sigma})\,(\chi_1, \chi_2, \chi_3, \chi_4) = (\hat{\sigma}(\chi_1), \hat{\sigma}(\chi_3), \hat{\sigma}(\chi_2), \hat{\sigma}(\chi_4))$$

The two middle terms get switched, the result of permutation (2 3) acting on the tuple, and the colors get permuted.

Representing $color_i$ by its subscript i allows the action on the coloring (1, 1, 2, 1) to be written as follows, using σ rather than $\hat{\sigma}$.

$$((2\ 3), \sigma)\,(1, 1, 2, 1) = (\sigma(1), \sigma(2), \sigma(1), \sigma(1))$$

Again the middle two elements get swapped by (2 3) and σ is applied to the color indexes. We next show an example with a specific permutation from *shapeGroup* and a specific permutation from *colorGroup* acting on a specific coloring.

Example. The group element $((2\ 3), (1\ 2))$ acts on the coloring (1, 1, 2, 1) as follows.

$$((1\ 2)(1), (1\ 2)(2), (1\ 2)(1), (1\ 2)(1)) = (2, 1, 2, 2)$$

So the middle elements of the tuple get swapped and all the colors get reversed.

Two colorings x_1 and x_2 are said to be equivalent if a group action maps one into the other (by permuting the vertices and colors). For example, all eight of the squares are equivalent whose vertices are three black and one white or one black and three white. Each of these squares can be mapped to any other via the action of some element of *coloringGroup*. Each equivalence class of colorings forms an orbit, which is defined below.

Definition. The **orbit** of the group G acting on the coloring x_1 is the set of colorings $\{x_2 : gx_1 = x_2,\ \text{for some}\ g \in G\}$.

As figure 5 shows, *coloringGroup*, acting on the 16 colorings of a square, has four orbits: an orbit with 2 elements (all colors the same), an orbit with 8 elements (a singleton color), an orbit with 4 elements (adjacent pairs of a color), and another orbit with 2 elements (diagonal pairs of a color). Figure 6 shows these same four orbits, written in the notation of a four-tuple of vertex colors as in figure 4; the goal of section 3 is to produce this numerical depiction of the orbits as tuples.

(1, 1, 1, 1)	(2, 2, 2, 2)		
(1, 1, 1, 2)	(1, 1, 2, 1)	(2, 1, 1, 1)	(1, 2, 1, 1)
(2, 2, 2, 1)	(2, 2, 1, 2)	(1, 2, 2, 2)	(2, 1, 2, 2)
(1, 1, 2, 2)	(2, 2, 1, 1)	(1, 2, 1, 2)	(2, 1, 2, 1)
(1, 2, 2, 1)	(2, 1, 1, 2)		

Figure 6: Orbits of *shapeGroup* × colorGroup acting on colorings of the square. These encodings of colorings correspond to the images in figure 5, where white=1 and black=2.

For two-dimensional Marching Squares, counting orbits of products of groups acting on colorings of vertices is no improvement over drawing a mere sixteen figures by hand and inspecting them for equivalence. However, by casting the problem in terms of combinatorial algebra we can exploit powerful computational tools to count the orbits for us in situations where the large dimension or large combination of colorings makes hand-enumeration overwhelming.

In this section we used the square as an example of the process of counting orbits of group actions on colorings, but our goal is to consider other shapes as well (such as triangles and tetrahedra), other sets of colors, and other kinds of groups acting on each of them. Before generalizing the algebraic details, we first describe a computational algebra package and demonstrate its ability to enumerate the number of cases for a square.

3 Computational Group Theory

Many practical questions in group theory can be answered by sheer calculation. Computational group theory is concerned with the numerical solution of problems in group theory, a notable example being the solutions to Rubik's cube. An article by Seress gives an overview of computational group theory [Seress 1854].

Two numerical packages for computational group theory are widely used: GAP (Groups, Algorithms, and Programming), which is free software, and Magma, costing about $US 1000 for a single license at the time of this writing. For more details about these packages, see "GAP – Groups, Algorithms and Programming" [Schönert 1994] and "An Introduction to MAGMA" [Cannon and Playoust 1993].

Practitioners of visualization, and other casual users of computational group theory, are likely to choose GAP because it is free software. So we describe how to use GAP to solve the particular problem of counting cases for a two-colored square. (Users should be aware that GAP uses the convention of applying group actions from the right, as noted in the definition of action in section 2.)

Below is a transcript of an interactive session using GAP, slightly edited for formatting purposes. The user input is shown in sans-serif font, and GAP's reply is shown in the fixed-width typewriter font. We begin by creating *shapeGroup*, *colorGroup*, and *coloringGroup* for dimension n with k colors (note: the double semicolons suppress feedback from GAP).

```
n := 2;;
k := 2;;
shapeGroup := WreathProductProductAction (SymmetricGroup(2),
    SymmetricGroup(n));;
colorGroup := Group (PermList (Reversed ([1..k])));;
coloringGroup := DirectProduct (shapeGroup, colorGroup);;
```

Next we construct projection operators to extract the two groups

back from their direct product.

```
shapeProjection := Projection (coloringGroup, 1);;
colorProjection  := Projection (coloringGroup, 2);;
```

Next we generate the list of colors and colorings, allowing GAP to answer back with its results.

```
numVerts := 2^n;;
coloredVerts := ListWithIdenticalEntries (numVerts, [1..k]);
[ [1..2], [1..2], [1..2], [1..2] ]
colorings := Cartesian (coloredVerts);

[[1,1,1,1], [1,1,1,2],[1,1,2,1], [1,1,2,2],
 [1,2,1,1], [1,2,1,2],[1,2,2,1], [1,2,2,2],
 [2,1,1,1], [2,1,1,2],[2,1,2,1], [2,1,2,2],
 [2,2,1,1], [2,2,1,2],[2,2,2,1], [2,2,2,2] ]
```

Then we define a function to produce the action of a group element on a coloring. The projections of element (g, σ) of *coloringGroup* yield the components g and σ that shuffle the order of the tuple and permute the colors.

```
action := function (coloring, groupElement)
  local   shapePerm, colorPerm, shuffled, result;
  shapePerm   := Image (shapeProjection, groupElement);
  colorPerm   := Image (colorProjection, groupElement);
  shuffled    := Permuted (coloring, shapePerm);
  result      := OnTuples (shuffled, colorPerm);
  return result;
end;;
```

We now let GAP produce the orbits and count how many there are. Notice that these orbits agree exactly with the tuples we computed in figure 6.

```
orbits := OrbitsDomain (coloringGroup, colorings, action);
[[[1,1,1,1], [2,2,2,2]],

 [[1,1,1,2], [1,1,2,1], [1,2,1,1], [2,2,2,1],
  [2,1,1,1], [2,2,1,2], [2,1,2,2], [1,2,2,2]],

 [[1,1,2,2], [2,2,1,1], [1,2,1,2], [2,1,2,1]],

 [[1,2,2,1], [2,1,1,2]]]
Length (orbits);
4
```

This demonstration shows how GAP can enumerate the orbits of a group action in Marching Squares, and thus to determine the number of cases for polytope colorings that arise in the two-dimensional version of Marching Cubes. The variable names suggest how to extend this example to handle other cases; for example, one can simply change the value of n from 2 to 3 to enumerate the orbits and count them for MC. One can also change the definition of *shapeGroup* or *colorGroup* at the beginning of the code to generate the orbits for still other colorings. The next section describes how this approach can be extended to handle additional geometries and symmetries, constructing a complete taxonomy of case-counts for substitopes.

4 Taxonomy of Substitopes

In order to extend the case-counting capabilities of our demonstration GAP program, we must express the shape groups and color groups for various substitopes. The colorings of interest to us are the ones that arise in algorithms like MC. The shape groups involve simplexes and cubes, with orientation-preserving (direct) symmetry and with full symmetry acting on them. The color groups include reversal and full permutation.

Shape groups. A polytope in n-dimensional space is acted on by symmetries of that space. The orthogonal group $O(n)$, consisting of $n \times n$ orthogonal matrices, forms a continuous group under matrix multiplication. This group contains finite subgroups that produce the symmetries of the n-simplex and of the n-cube. The symmetries of the simplex form a subgroup of $O(n)$ isomorphic to S_{n+1}, the symmetric group permuting the $n+1$ vertices of the n-simplex. The symmetries of the cube form a subgroup of $O(n)$ isomorphic to $S_2 \wr S_n$, also known as the hyperoctahedral group. These two groups are the *full symmetry* groups of the simplex and the cube.

The set of orientation-preserving symmetries of $\mathbb{R}^n$ forms the special orthogonal group $SO(n)$, a subgroup of $O(n)$. Its intersection with a full symmetry group, called a *direct symmetry*, yields the orientation-preserving symmetries on the simplex or the cube. The direct symmetry group for the n-simplex is the *alternating group* A_n. The direct symmetry group for the n-cube is the *direct-cube group*.

Color groups In the examples cited in Section 1.1 above, variations on Marching Cubes have employed three different groups to permute color indexes. These are listed below.

(1) The simplest color group is the *identity* group, which leaves each index fixed. Nielson and Sung considered the ordering of the colors to be significant, meaning the *identity* group Id_k acted on k colors.

(2) The *reversal* group Rev_k on the numbers $\{1..k\}$ swaps the first with the last element, the second with the next-to-last element, and so forth. Lorensen and Cline used the reversal group to reorder two colors. The group Rev_k contains only two permutations: the identity permutation () and the permutation ρ defined below.

$$\rho = \begin{cases} (1\ k)(2\ k\text{-}1)..(\frac{k}{2}\ \frac{k+2}{2}) & \text{if } k \text{ is even} \\ (1\ k)(2\ k\text{-}1)..(\frac{k-1}{2}\ \frac{k+3}{2}) & \text{if } k \text{ is odd} \end{cases}$$

(3) Nielson and Franke considered two color orderings to be equivalent no matter how they were permuted, thereby allowing the symmetric group S_k to act on the k colors.

So we see in the literature variations on MC in which *shapeGroup* is one of the two types of symmetry {*direct, full*} acting on a polytope $p \in \{simplex, cube\}$, and *colorGroup* is one of the three groups $\{Id_k, Rev_k, S_k\}$. We programmed GAP to fill in a table (table 1) of combinations of these parameters, with both the dimension n and the number k of colors in the range [1..4]. Each entry in the table gives the number of cases (orbits) for the corresponding colored polytopes. A table entry noted in boldface indicates a combination of parameters that was at work in any of six algorithms surveyed in section 1.1. The GAP source code `orbitTable.gap` that generated the table is freely available for download at the GAP Web site.

The table is easily computed on a desktop machine for n and k in the range [1..3]. For this range, we measured the table generation time at about ten seconds on a desktop machine with 1 GB memory and a 1.7 GHz Intel Xeon processor. But the memory demands increase significantly for the 4-cube with more than two colors; calculating the orbits exceeded the capacity of our desktop machine. In order to enumerate the six coloring groups acting on the 4-cube with 3 colors, we used the parallel GAP package ParGAP by Gene Cooperman [Cooperman 1999] and ran it on a Beowulf cluster [Sterling et al. 1995] composed of 85 nodes, each node having dual 2.4GHz processors, connected by 100 Mb/s Ethernet. Using two processes per node (with 0.5 GB of workspace per process) on twelve nodes, calculating each of the six table entries for the 4-cube with three colors took about 300-500 seconds of wall-clock time, and about 1000-3000 seconds of CPU time. In other words, calculating the six cases $n = 4$, $k = 3$ for the cube took more than a thousand times longer than calculating the 54 cases where $n \leq 4$ and $k \leq 3$.

Calculating the case-counts for four colors ($k = 4$) exceeded even the memory capacity of our parallel version, although it appears that the ParGAP version of our code could be further modified to exploit

	n-simplex						*n*-cube			
n*k*	*1*	*2*	*3*	*4*		*n**k*	*1*	*2*	*3*	*4*
1	1	4	9	16	(*direct, Id*$_k$)	*1*	1	4	9	16
2	1	4	11	24		*2*	1	6	24	70
3	1	5	15	36		*3*	1	23	333	2916
4	1	6	21	56		*4*	1	496	230076	>22000000
	1	2	5	8	(*direct, Rev*$_k$)		1	2	5	8
	1	2	6	12			1	4	14	38
	1	3	9	20			1	[1]**15**	183	1508
	1	3	12	28			1	[2]**272**	115606	>11000000
	1	2	2	2	(*direct, S*$_k$)		1	2	2	2
	1	2	3	3			1	4	6	7
	1	3	4	5			1	15	72	166
	1	3	5	6			1	272	38914	>930000
	1	3	6	10	(*full, Id*$_k$)		1	3	6	10
	1	4	10	20			1	6	21	55
	1	5	[3]**15**	35			1	22	267	1996
	1	6	21	56			1	402	132102	>11000000
	1	2	4	6	(*full, Rev*$_k$)		1	2	4	6
	1	2	6	10			1	4	[7]**13**	31
	1	[4]**3**	[7]**9**	19			1	[1]**14**	[7]**147**	[7]**1036**
	1	[5]**3**	[5,7]**12**	28			1	[2]**222**	[7]**66524**	>5600000
	1	2	2	2	(*full, S*$_k$)		1	2	2	2
	1	2	3	3			1	4	6	[7]**7**
	1	3	4	[6]**5**			1	14	58	[7]**124**
	1	3	5	6			1	222	22490	>460000

Table 1: Table of case counts for substitopes. Each of the twelve sub-tables contains the case-counts for the tuple (*shapeGroup, colorGroup, polytope, n, k*), with *n* and *k* in the range [1..4]. Each row of sub-tables shares (*shapeGroup, colorGroup*), as indicated in the middle. The left column contains sub-tables for simplexes; the right column for cubes. Case-counts specifically mentioned in this paper are highlighted in boldface. [1]Marching Cubes. [2]Marching Hypercubes. [3]Interval Volume. [4]Sweeping Simplices. [5]Contour Meshing. [6]Separating Surfaces. [7]Counting Cases (this paper).

finer-grain parallelism and thus satisfy the memory constraint. We therefore merely estimate a lower bound for each of these entries, based on the fact that no orbit can be bigger than the order of the group.

5 Summary

Marching Cubes (MC), and algorithms like it, share the essential feature of applying geometric substitution to polytope colorings to produce substitopes. These algorithms vary in the choice of groups acting on vertices and on colors, in the choice of polytope, and in the choice of dimension *n* and number of colors *k*. We presented a technique for enumerating the cases that arise in counting the cases of polytope colorings, and showed how this technique can be applied using software for computational group theory software (called GAP). One benefit of a tool for computational algebra is that it independently confirms the results announced by Bhaniramka, Wenger, and Crawfis [Bhaniramka et al. 2000] and by Roberts and Hill [Roberts and Hill 1999] for counting the cases in four-dimensional MC, results that cannot reasonably be checked by hand. Moreover it permits us to predict the size of tables for MC variants that have yet to be implemented. The following examples illustrate the predictive utility of the table.

Example. What happens when the algorithm for Separating Surfaces is extended to 3-cubes with four colors? Consulting the table

(*full, S_k, n*-cube, 3, 4) we see that 124 cases arise. For the square ($n = 2$) with four colors, only seven cases arise; they are illustrated in the diagram at the top of the paper (top row), with a possible interpretation of colors that might be derived from medical data.

Example. What happens when MC is extended to handle the degenerate situation where $f(v_i) - c = 0$ at vertices v_i? This case would almost never happen (that is, would occur on a set with measure zero) if the scalar function were truly real-valued. But in practice one routinely encounters integer-valued isosurfaces of integer-valued datasets, so a level set may, with non-zero probability, pass through many grid points. When the set of corresponding colors is augmented to include the degenerate case (*full, Rev_k, n*-cube, 3, 3) we see that 147 cases arise. For the square ($n = 2$) with three colors, only 13 cases arise; they are illustrated in the lower part of the diagram at the top of this paper (bottom row).

What happens when Marching Hypercubes is extended to handle the degenerate situation where $f(v_i) - c = 0$? The table predicts that for (*full, Rev_k, n*-cube, 4, \{2,3\}) the number of cases explodes from 222 to 66,524.

Example. Weigle and Banks briefly discussed the degenerate situation where $f(v_i) - c = 0$ for *n*-simplexes in Contour Meshing, but did not enumerate all the cases. How many cases would they have found for the 4-simplex? The table predicts that for (*full, Rev_k, n*-simplex, 4, 3) there are 12 cases.

Example. What happens when Interval Volumes is applied to cubes instead of tetrahedra? The table predicts that for (*full, Rev_k, n*-cube, 3, 3) there are 147 cases.

What happens when an interval is added, creating the four "colors" $(-\infty, a)$, (a, b), (b, c), and (c, ∞)? Letting $k = 4$, we find the table predicts that for (*full, Rev_k*, \{*n*-simplex, *n*-cube\}, 3, 4) there are 19 cases for the tetrahedron and 1036 cases for the cube.

These examples illustrate how table 1 can be used to determine the number of cases required to implement a variation on existing MC-style algorithms. The table also imposes a clear taxonomy on this collection of algorithms where no such organization has heretofore been suggested. This indicates that a very deep, very generic underlying algorithm for visualization exists, which can be incarnated in many different ways. The table also shows the intimate connection between group theory, geometry, and visualization. The availability of GAP, a free software package for computational group theory, should encourage the research community to explore group actions for generating substitopes in novel ways; not only can GAP count the number of cases (orbits) automatically, but it can also enumerate the members of each orbit. This capability eliminates a very difficult part of any substitope algorithm. We look forward to the novel applications that this general framework invites.

We are actively working on ways to expand the table to larger values of *n* and *k*, and to determine the asymptotic case-counts of certain coloring groups as functions of *n* and *k*. We are also actively exploring ways in which colored polytopes might arise in other types of data and in other visualization techniques.

References

TEMPLATE GRAPHICS SOFTWARE. *Module: GMC*. (User manual entry for amira visualization software).

BHANIRAMKA, P., WENGER, R., AND CRAWFIS, R. 2000. Isosurfacing in higher dimensions. In *Proceedings of IEEE Visualization 2000*, IEEE, 267–273.

BLOOMENTHAL, J. 1988. Polygonization of implicit surfaces. In *Computer Aided Geometric Design*, vol. 5, 341–355.

CANNON, J., AND PLAYOUST, C. 1993. *An Introduction to MAGMA*. School of Mathematics and Statistics, Sydney University.

CAYLEY, A. 1854. On the theory of groups, as depending on the symbolic equation $\theta^n = 1$. *Philosophical Magazine 7*, 40–47.

COHN, P. M. 1984. *Algebra, volume 1 (second edition)*. Wiley.

COOPERMAN, G. 1999. *Parallel GAP/MPI (ParGAP/MPI), Version 1*. College of Computer Science, Northeastern University.

FRALEIGH, J. G. 1998. *A First Course in Abstract Algebra (6th edition)*. Addison-Wesley Publishing.

GALLAGHER, R. S. 1991. Span filter: An optimization scheme for volume visualization of large finite element models. In *Proceedings of IEEE Visualization 1991*, IEEE, 68–75.

THE GAP GROUP. 2002. *GAP – Groups, Algorithms, and Programming, Version 4.3*. (http://www.gap-system.org).

GLASSNER, A. 1992. A tutorial on geometric replacements. *IEEE Computer Graphics & Application 12*, 1 (January), 22–36.

HEGE, H.-C., SEEBASS, M., STALLING, D., AND ZÖCKLER, M. December 1997. *A Generalized Marching Cubes Algorithm Based on Non-Binary Classifications*. Konrad-Zuse-Zentrum für Informationstechnik Berlin. Technical Report SC 97-05.

HEGE, H.-C. May 25-29, 1998. Enumeration of symmetry classes in mesh generation. In *Report on Dagstuhl Seminar 9821: Hierarchical Methods in Computer Graphics*, M. Gross, H. Müller, P. Schröder, and H.-P. Seidel, Eds., 9–10.

KOBBELT, L., CAMPAGNA, S., AND SEIDEL, H.-P. 1998. A general framework for mesh decimation. In *Graphics Interface*, 43–50.

LINDENMAYER, A. 1971. Developmental systems without cellular interaction, their languages and grammars. *Journal of Theoretical Biology*, 455–484.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of SIGGRAPH 1987*, ACM Press, 163–169.

LOUNSBERY, M., DEROSE, T. D., AND WARREN, J. 1997. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics 16*, 1, 34–73.

MONTGOMERY, K., BRUYNS, C., AND WILDERMUTH, S. 2001. A virtual environment for simulated rat dissection: a case study of visualization for astronaut training. In *Proceedings of IEEE Visualization 2001*, IEEE, 509–512.

NIELSON, G. M., AND FRANKE, R. 1997. Computing the separating surface for segmented data. In *Proceedings of IEEE Visualization 1997*, IEEE, 229–233.

NIELSON, G. M., AND SUNG, J. 1997. Interval volume tetrahedrization. In *Proceedings of IEEE Visualization 1997*, IEEE, 221–228.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1990. *The algorithmic beauty of plants*. Springer-Verlag, New York.

ROBERTS, J. C., AND HILL, S. 1999. Piecewise linear hypersurfaces using the marching cubes algorithm. In *Visual Data Exploration and Analysis VI, Proceedings of SPIE*, IS&T and SPIE, R. Erbacher and A. Pang, Eds., 170–181.

SCHÖNERT, M. 1994. *GAP - Groups, Algorithms and Programming*. Lehrstuhl D für Mathematik, RTWH, Aachen.

SERESS, Á. 1854. An introduction to computational group theory. *Notices of the AMS 44*, 6 (June/July), 671–679.

SHEN, H.-W., AND JOHNSON, C. R. 1995. Sweeping simplices: A fast iso-surface extraction algorithm for unstructured grids. In *Proceedings of IEEE Visualization 1995*, IEEE, 143–151.

SHEN, H.-W., HANSEN, C. D., LIVNAT, Y., AND JOHNSON, C. R. 1997. Isosurfacing in span space with utmost efficiency (issue). In *Proceedings of IEEE Visualization 1996*, IEEE, 287–294.

STALLING, D., ZÖCKLER, M., SANDER, O., AND HEGE, H.-C. December 1998. *Weighted Labels for 3D Image Segmentation*. Konrad-Zuse-Zentrum für Informationstechnik Berlin. Technical Report SC 98-39.

STERLING, T., SAVARESE, D., BECKER, D. J., DORBAND, J. E., RANAWAKE, U. A., AND PACKER, C. V. 1995. BEOWULF: A parallel workstation for scientific computation. In *Proceedings of the 24th International Conference on Parallel Processing*, I:11–14.

THIBAULT, W. C., AND NAYLOR, B. F. 1987. Set operations on polyhedra using binary space partitioning trees. In *Proceedings of ACM SIGGRAPH 1987*, ACM, 153–162.

WEIGLE, C., AND BANKS, D. C. 1996. Complex-valued contour meshing. In *Proceedings of IEEE Visualization 1996*, IEEE, 173–180.

WILHELMS, J., AND GELDER, A. V. 1992. Octrees for faster isosurface generation. *ACM Transactions on Graphics 11*, 3 (July), 201–227.