Measuring Space Filling Diagrams and Voids[1]

Herbert Edelsbrunner[2] and Ping Fu[3]

Abstract. A space filling diagram is the union of finitely many balls in $\mathbb{R}^3$. We specify software that computes metric properties of such diagrams based on the inclusion-exclusion formulas developed in [6]. It consists of more than 100 functions and computes volume, surface area, total arc length, and number of corners of space filling diagrams. Similarly, it measures voids and surface area contributions of individual balls. The software is available via anonymous ftp.

Keywords. Computational chemistry and biology; space filling diagrams, duality, simplicial complexes, alpha shapes, metric inclusion-exclusion formulas; software specification.

[1]This work is supported by the National Science Foundation, under grant ASC-9200301 and the Alan T. Waterman award, grant CCR-9118874. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the National Science Foundation.

[2]Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, USA.

[3]National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, Champaign, Illinois 61820, USA.

Measuring Space Filling Diagrams and Voids[1]

Herbert Edelsbrunner[2] and Ping Fu[3]

Abstract. A space filling diagram is the union of finitely many balls in $\mathbb{R}^3$. We report on software that computes metric properties of such diagrams based on the inclusion-exclusion formula developed in [illegible]. It consists of three [illegible] the balls, [illegible] computes volume, surface area, total arc length, and number of vertices of [illegible] diagram. Similarly, it measures voids and surface area contributions of individual balls. The software is available [illegible] anonymous ftp.

Keywords. Computational chemistry and biology, space filling diagrams, voids, simplicial complexes, alpha shapes, [illegible] inclusion-exclusion formula, software specification.

[1] This work is supported by the National Science Foundation, under grant ASC-9200301 and the Alan T. Waterman award, grant CCR-9118874. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the National Science Foundation.

[2] Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, USA.

[3] National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, Urbana-Champaign, Illinois 61820, USA.

Table of contents

I.1 Introduction. A molecule is often modeled as the union of balls in three-dimensional Euclidean space, $\mathbb{R}^3$. Each ball represents an atom and its size is determined by the van der Waals radius of the atom. Such a model is known as the *space filling diagram* of the molecule, introduced by Lee and Richards [9], see also [12, 13]. This paper presents and documents software consisting of more than 100 functions, written in the C programming language [11], that can be used to measure a space filling diagram. The software makes no use of the fact that the union of balls is defined by atoms of a molecule. It is therefore more generally applicable to problems about balls in $\mathbb{R}^3$.

The problem of measuring space filling diagrams has received a fair amount of attention in computational biology and chemistry, and software implementing numerical and analytic approaches are available, see e.g. [3, 4, 10]. Our software belongs to the category of analytic approaches. Our work differs from earlier work within this category in at least two respects: it is based on the dual complex of the space filling diagram, see [7], and on inclusion-exclusion formulas with terms for intersections of at most four balls at a time, see [6]. In spite of the shallow terms, the formulas are correct (non-approximative) even if there are points covered by many more than four balls. The dual complex is part of the larger Delaunay or weighted Delaunay simplicial complex. This has the advantage that the unoccupied space, in the sense used in [1], is properly represented by the simplices not in the dual complex. Our software uses this representation to compute and measure all voids formed by a space filling diagram.

In II.2 and II.3 we specify the input to the software, and we describe the various metric properties it computes. These are the volume, surface area, total arc length, and number of corners of the space filling diagram and its voids, which are the bounded components of the diagram's complement. The dual complexes and their relationship to space filling diagrams are briefly explained in III.4. The interested reader is urged to consult [6] for a more detailed description of the relationship. The formulas for measuring a space filling diagram and its voids are expressed in algorithmic language in III.5 and III.6. Section III.7 briefly considers the envelope of the balls, that is, the region of space inaccessible from the outside. Equivalently, it is the space filling diagram union its voids. Section III.8 gives a formula for measuring the so-called outside fringe of the space filling diagram. Section III.9 discusses how the contributions of individual balls to the surface area can be computed. This applies to the surface area of the space filling diagram, its collection of voids, a single void, and the outside fringe. There are linear relations between the various measurements which can be used to test their correctness.

Before we can translate the formulas in III.5 through III.9 into measurements, we need to identify the voids of the space filling diagram. To do this, we use their one-to-one correspondence with the voids of the dual complex. Each such void is represented as the union of tetrahedra. By the nature of being voids, these tetrahedra do not belong to the dual complex. This is described in IV.10, and IV.11 shows how a union-find data structure is employed to compute all voids. Lower level operations, such as deciding the status of a vertex, edge, and triangle, or finding the tetrahedra incident to a triangle, are described in IV.12.

The main part of this paper consists of V.13 through V.19, which specify the primitive metric functions of the software. They form a network of some 54 functions computing and measuring a variety of basic geometric objects. Since this paper also serves as a documentation of the software, we do not hesitate to give more details than seemingly necessary. The software is available via anonymous ftp at ftp.ncsa.uiuc.edu or 141.142.20.50. The connection between this document and the software is given by explicit references to the function names and by back-references from the functions to the sections of this paper.

II.2 The data. Each ball is specified by the three coordinates of its center, $x, y, z \in \mathbb{R}$, together with its radius, $w \in \mathbb{R}$. The collection of balls is stored in a linear array, B:

```
type Ball_type = record $x$, $y$, $z$, $w$: Vol_real end;
var  $B$: array [1..$n$] of Ball_type.
```

Here, `Vol_real` is the same as `double` and thus represents the data type of double-precision floating-point numbers. The inner workings of the software are based on the notion of the weight of a point or ball, and a real parameter, α. The parameter globally modifies all radii. For applications where radii do not change, α can be set to zero and henceforth be ignored. However, part of the versatility and efficiency of this software stems from the availability of this parameter, and it is instructive to understand how it interacts with the weight. For a ball with *initial* radius w, the specification of α changes it to the *actual* radius, which is

$$\sqrt{w^2\text{sgn}(w) + \alpha^2\text{sgn}(\alpha)}.$$

We refer to a situation where all initial radii are zero to the *unweighted* case. In such a case, all radii are equal to α. In general, it is possible and admissible that $w^2\text{sgn}(w) + \alpha^2\text{sgn}(\alpha)$ is negative. The corresponding radius is imaginary and the ball is ignored for all measurements. In most computations, it is possible to pass indices of B as parameters rather than individual coordinates and weights. Typically we use i, j, k, ℓ to denote such indices. An index is either referred to as a ball or a point or vertex. Another internal data type is

```
type Vector = record x, y, z: Vol_real end;
```

By convention, we denote variables of type **Vector** by p, s, t, u, v, with or without sub- or super-scripts. A frequently used operation is function vector, which creates a vector representing the center of a ball i in B.

II.3 The output. We consider four problems, namely measuring the union of the balls (the space filling diagram, $\bigcup B$), measuring the voids of $\bigcup B$, measuring the envelope (the union of ball together with the voids), and measuring the outside fringe (explained later). The main program can be executed interactively or in batch mode. Selections among the options are then done either in function switch_computation or with command_line specifications. After finishing computations, all memory is freed by function volbl_clean. We discuss the options in the above sequence.

The union of balls, $\bigcup B$, is represented by its dual complex, $\mathcal{K}$, which is a collection of vertices, edges, triangles, and tetrahedra, see III.4. The following metric information about $\bigcup B$ is computed:

the volume, $V_{\text{sf}} = V(\bigcup B)$,
the surface area, $A_{\text{sf}} = A(\bigcup B)$,
the total arc length, $L_{\text{sf}} = L(\bigcup B)$,
and the number of corners, $C_{\text{sf}} = C(\bigcup B)$.

A void, $\mathcal{V}$, is represented by the collection of tetrahedra whose union is the corresponding void in $\mathcal{K}$. We denote by $\mathcal{V}$ the void as a geometric object (a bounded component of $\mathbb{R}^3 - \bigcup B$) as well as a combinatorial object (the set of tetrahedra that belong to the corresponding void in $\mathcal{K}$.) The one-to-one correspondence between the two types of voids is established in [6]. The following metric information about $\mathcal{V}$ is computed:

the volume, $V_{\text{v}} = V(\mathcal{V})$,
the surface area, $A_{\text{v}} = A(\mathcal{V})$,
the total arc length, $L_{\text{v}} = L(\mathcal{V})$,
and the number of corners, $C_{\text{v}} = C(\mathcal{V})$.

Besides individual voids, we compute the total measure of all voids:

$$V_{\text{tv}} = \sum_{\text{voids } \mathcal{V}} V(\mathcal{V}),\ \ A_{\text{tv}} = \sum_{\text{voids } \mathcal{V}} A(\mathcal{V}),\ \ L_{\text{tv}} = \sum_{\text{voids } \mathcal{V}} L(\mathcal{V}),\ \ C_{\text{tv}} = \sum_{\text{voids } \mathcal{V}} C(\mathcal{V}).$$

The measurements of the envelope of $\bigcup B$, $\bigcup B$ union its voids, are

$$V_{\text{e}} = V_{\text{sf}} + V_{\text{tv}},\ \ A_{\text{e}} = A_{\text{sf}} - A_{\text{tv}},\ \ L_{\text{e}} = L_{\text{sf}} - L_{\text{tv}},\ \ C_{\text{e}} = C_{\text{sf}} - C_{\text{tv}}.$$

Finally, we measure the *outside fringe*. This is the part of $\bigcup B$ that reaches into the unbounded component of the complement of the dual complex. Its measurements, $V_{\text{of}}, A_{\text{of}}, L_{\text{of}}, C_{\text{of}}$, are computed using similar formulas as for $\bigcup B$ and the voids. The main reason for measuring the outside fringe is for software verification purposes. In particular, the following relations for the measurements can be used to double-check correctness.

$$\begin{aligned} V_{\text{sf}} + V_{\text{tv}} - V_{\text{tiv}} - V_{\text{sh}} - V_{\text{of}} &= 0, \\ A_{\text{sf}} - A_{\text{tv}} - A_{\text{of}} &= 0, \\ L_{\text{sf}} - L_{\text{tv}} - L_{\text{of}} &= 0, \\ C_{\text{sf}} - C_{\text{tv}} - C_{\text{of}} &= 0, \end{aligned}$$

where V_{tiv} is the total volume of the voids in the dual complex, and V_{sh} is the volume of the dual shape, as computed in shape_volume. Both are sums of tetrahedron volumes, namely those representing voids, and those representing the dual shape. This check is implemented in volbl_checking, and volbl_print can be used to print any of the above results. In order to relativize the unavoidable floating-point imprecision, the check-sums are reported as multiples of the volume, area, and radius of the largest ball in B. The latter is computed by largest_ball, and its measurements are reported by print_sizes.

By convention, all metric measurements (volume, area, and length) are taken in multiples of the natural unit implied by the point coordinates. All angles are measured in revolution, so that 0 is the empty angle and 1 the full angle. In our pseudo-code notation we use semicolons (;) if the order of the separated parameters is important, and commas (,) if it is irrelevant.

III.4 Geometric background. The software described in this paper assumes the availability of the dual complex, denoted $\mathcal{K}$, as a subcomplex of the regular simplicial complex, $\mathcal{R}$, of the balls. $\mathcal{R}$ is sometimes called the weighted Delaunay simplicial complex or triangulation. In the case where all radii are the same, $\mathcal{R}$ is the Delaunay simplicial complex of the ball centers. These concepts are described in appropriate detail in [6]. We just mention that the boundary of $\mathcal{K}$ is dual to the boundary of the space filling diagram, $\bigcup B$. This is illustrated in figure 1. The dual

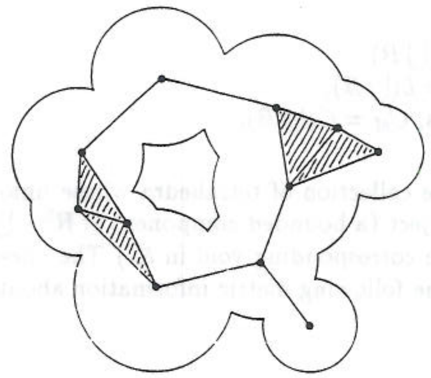

Figure 1: In the plane, the space filling diagram is the union of finitely many disks. The dual complex is a collection of vertices, edges, and triangles contained in the union of disks.

complex $\mathcal{K}$ of $\bigcup B$ is a subcomplex of $\mathcal{R}$. Indeed, the balls can be grown by increasing α so that $\mathcal{K}$ contains more and more simplices of $\mathcal{R}$ until $\mathcal{K} = \mathcal{R}$ when the balls are sufficiently large. The dual complex is constructed using the programs regtri, mkalf, and alvis available via anonymous ftp at ftp.ncsa.uiuc.edu. The third program, alvis, also visualizes $\mathcal{K}$ and can be used to select interesting ball sizes for the given set of centers. The next five sections show how to use $\mathcal{K}$ and $\mathcal{R}$ to measure a space filling diagram, its voids, their union, the outside fringe, and how to separate contributions to the surface area made by different balls. The corresponding formulas are developed and proved in [6].

III.5 Space filling diagrams. The straight inclusion-exclusion formulas measuring $\bigcup B$, see [6, section 6], are implemented in function space_filling_measurements. They are evaluated in a single loop over the simplices of $\mathcal{K}$. All output parameters are initialized to zero:

$V_{sf} := A_{sf} := L_{sf} := C_{sf} := 0$.

The main loop considers all simplices of $\mathcal{K}$ and computes intersections of one, two, three, or four balls. It is convenient to denote a simplex by the list of vertices.

```
for each $\sigma \in \mathcal{K}$ do if $\sigma = i$ then $V_{sf} := V_{sf} + \text{ball\_volume}(i)$;
                                   $A_{sf} := A_{sf} + \text{ball\_area}(i)$
                    endif;
                    if $\sigma = ij$ then $V_{sf} := V_{sf} - \text{ball2\_volume}(i,j)$;
                                   $A_{sf} := A_{sf} - \text{ball2\_area}(i,j)$;
                                   $L_{sf} := L_{sf} + \text{ball2\_length}(i,j)$
                    endif;
                    if $\sigma = ijk$ then $V_{sf} := V_{sf} + \text{ball3\_volume}(i,j,k)$;
                                   $A_{sf} := A_{sf} + \text{ball3\_area}(i,j,k)$;
                                   $L_{sf} := L_{sf} - \text{ball3\_length}(i,j,k)$;
                                   $C_{sf} := C_{sf} + 2$
                    endif;
                    if $\sigma = ijk\ell$ then $V_{sf} := V_{sf} - \text{ball4\_volume}(i,j,k,\ell)$;
                                   $A_{sf} := A_{sf} - \text{ball4\_area}(i,j,k,\ell)$;
                                   $L_{sf} := L_{sf} + \text{ball4\_length}(i,j,k,\ell)$;
                                   $C_{sf} := C_{sf} - 4$
                    endif
endfor.
```

The area computations are slightly more involved than indicated above because they also determine the contribution to A_{sf} of each individual ball, see III.9. As it turns out, the implementation of the straight inclusion-exclusion formulas, as described above, is considerably slower than the computations following the decomposable formula, see below. It is therefore possible to obtain the measurements of a space filling diagram more efficiently using the linear relations in III.3. This is indeed done in function space_filling_measurements_2.

III.6 Voids. The space filling diagram in figure 1 has a single void also shown in figure 2. This void of $\bigcup B$ is contained in a corresponding void in $\mathcal{K}$. Its volume is measured by subtracting the pieces of the spheres reaching into the void of $\mathcal{K}$. The corresponding theory is expressed by the decomposable inclusion-exclusion formulas in [6,

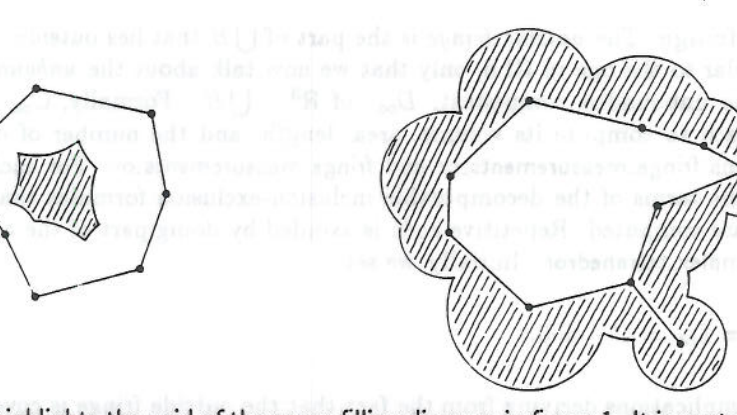

Figure 2: The left drawing highlights the void of the space filling diagram in figure 1. It is contained in the corresponding void of the dual complex. To the right, we see the outside fringe of the same space filling diagram.

section 8]. In contrast to the straight inclusion-exclusion formulas in III.5, the decomposable formulas have terms weighted by angles. These formulas are implemented in function measure_a_void, which measures a single void $\mathcal{V}$ of

$\bigcup B$. To avoid repetitive code, parts of the computations are done in the auxiliary functions do_void_tetrahedron, do_tetra_vertex, do_tetra_edge, and do_tetra_triangle. By accumulating single void measurements, we get the total size of all voids in voids_measurements. To measure a single void $\mathcal{V}$, we initialize the volume to the sum of volumes of the representing tetrahedra; the area, length, and number of corners is initially zero.

```
$V_v := A_v := L_v := C_v := 0$;
for each tetrahedron $\sigma = ijk\ell \in \mathcal{V}$ do $V_v := V_v + \text{tetrahedron\_volume}(i, j, k, \ell)$ endfor.
```

The main loop considers all tetrahedra in $\mathcal{V}$ and their faces, if they are in $\mathcal{K}$. For each such tetrahedron-face pair, it measures a sector, wedge, or a pawn (half the intersection of three balls).

```
for each $\sigma = ijk\ell \in \mathcal{V}$ do if $i \in \mathcal{K}$ then $V_v := V_v - \text{sector\_volume}(i; j, k, \ell)$;
                                    $A_v := A_v + \text{sector\_area}(i; j, k, \ell)$
                         endif; do the same for $j$, $k$, and $\ell$;
                         if $ij \in \mathcal{K}$ then $V_v := V_v + \text{wedge\_volume}(i, j; k, \ell)$;
                                    $A_v := A_v - \text{wedge\_area}(i, j; k, \ell)$;
                                    $L_v := L_v + \text{wedge\_length}(i, j; k, \ell)$
                         endif; do the same for $ik$, $i\ell$, $jk$, $j\ell$, and $k\ell$;
                         if $ijk \in \mathcal{K}$ then $V_v := V_v - \text{pawn\_volume}(i, j, k)$;
                                    $A_v := A_v + \text{pawn\_area}(i, j, k)$;
                                    $L_v := L_v - \text{pawn\_length}(i, j, k)$;
                                    $C_v := C_v + 1$
                         endif; do the same for $ij\ell$, $ik\ell$, and $jk\ell$
endfor.
```

The area computations are slightly more involved than shown, for the same reasons mentioned in III.5.

III.7 Envelopes. As mentioned in III.3, the measurements related to the impact of $\bigcup B$ on its environment are sometimes of interest. This is for example the case in the study of nano-crystals, where the volume is defined so it reflects the number of water molecules that would otherwise occupy their space. These water molecules have no way to access the voids of $\bigcup B$. We therefore measure the union of balls union the voids. The volume of the envelope is the sum of the volume of $\bigcup B$ and its voids, whereas the area, length, and the number of corners are obtained as differences of measurements. This is implemented in function envelope_measurements.

III.8 Outside fringe. The *outside fringe* is the part of $\bigcup B$ that lies outside the dual complex, see figure 2. The situation is similar to the one in III.6, only that we now talk about the *unbounded* component, C_∞, of $\mathbb{R}^3 - \bigcup \mathcal{K}$. C_∞ contains the unbounded component, D_∞, of $\mathbb{R}^3 - \bigcup B$. Formally, $C_\infty - D_\infty$ is the outside fringe. The following approach to compute its volume, area, length, and the number of corners is implemented in different ways by functions fringe_measurements_cx and fringe_measurements_ov. For each vertex, edge, and triangle of the dual complex, the terms of the decomposable inclusion-exclusion formulas that locally lie outside $\mathcal{K}$ and outside the voids of $\mathcal{K}$ are computed. Repetitive code is avoided by doing part of the computations in auxiliary functions, including do_complex_tetrahedron. Initially, we set

$$V_{\text{of}} := A_{\text{of}} := L_{\text{of}} := C_{\text{of}} := 0.$$

To avoid the complications deriving from the fact that the outside fringe is covered only partially by tetrahedra of $\mathcal{R}$, we compute the angles at vertices and edges and use them as weights in the inclusion-exclusion formulas. Let φ_σ denote the angle in C_∞ at σ. We first show how to use it and later discuss how to compute it.

```
for each $\sigma \in \mathcal{K}$ do if $\sigma = i$ then $V_{\text{of}} := V_{\text{of}} + \varphi_\sigma * \text{ball\_volume}(i)$;
                                    $A_{\text{of}} := A_{\text{of}} + \varphi_\sigma * \text{ball\_area}(i)$
                  endif;
                  if $\sigma = ij$ then $V_{\text{of}} := V_{\text{of}} - \varphi_\sigma * \text{ball2\_volume}(i,j)$;
                                    $A_{\text{of}} := A_{\text{of}} - \varphi_\sigma * \text{ball2\_area}(i,j)$;
                                    $L_{\text{of}} := L_{\text{of}} + \varphi_\sigma * \text{ball2\_length}(i,j)$
                  endif;
                  if $\sigma = ijk$ then $V_{\text{of}} := V_{\text{of}} + \varphi_\sigma * \text{ball3\_volume}(i,j,k)$;
                                    $A_{\text{of}} := A_{\text{of}} + \varphi_\sigma * \text{ball3\_area}(i,j,k)$;
                                    $L_{\text{of}} := L_{\text{of}} - \varphi_\sigma * \text{ball3\_length}(i,j,k)$;
                                    $C_{\text{of}} := C_{\text{of}} + 2 * \varphi_\sigma$
                  endif
endfor.
```

The implementations differ from this simplifying scheme in two respects. For a vertex, edge, or triangle σ in the interior of $\mathcal{K}$, we have $\varphi_\sigma = 0$, and σ is discarded without computing any angle. Second, the area computations include a separate accumulation of contributions by individual balls, see III.9.

The main difference between the two implementations, fringe_measurements_cx and fringe_measurements_ov, is the way they go about computing the angles φ_σ. Recall that φ_σ is the angle at σ outside $\bigcup \mathcal{K}$. For a triangle $\sigma \in \mathcal{K}$ this means that φ_σ is 0, $\frac{1}{2}$, or 1, depending on whether two, one, or zero incident tetrahedra belong to $\mathcal{K}$ or to a void of $\mathcal{K}$. Function fringe_measurements_cx computes an angle indirectly, by subtracting angles inside tetrahedra of $\mathcal{K}$ or voids from the full angle. To avoid storing φ_σ, we adjust the measurements for each partial angle directly. So in the end, the function is a loop through all vertices, edges, triangles, and tetrahedra of $\mathcal{K}$. For each non-interior vertex, edge, and triangle, the contribution with full angle $\varphi_\sigma = 1$ is recorded; interior vertices, edges, and triangles are ignored. For each tetrahedron in $\mathcal{K}$, the contributions of its non-interior vertices, edges, and triangles inside the tetrahedron are subtracted from the corresponding initial contributions. Finally, the measurements are adjusted by subtracting the measurements of all voids.

While fringe_measurements_cx computes angles from the inside, fringe_measurements_ov computes angles from the outside, using tetrahedra that lie outside $\mathcal{K}$ and all voids of $\mathcal{K}$. These tetrahedra are collected and processed in the same way as described for voids in III.6. This method is made complicated by the fact that only a small part of C_∞ is covered by tetrahedra, namely the part inside the convex hull of all points, which is the same as $\bigcup \mathcal{R}$. To account for the rest, the convex hull vertices, edges, and triangles are traversed in depth-first order using function traverse_ch. The traversal requires auxiliary functions for a marking mechanism (mark_triangle, mark_vertex, is_triangle_marked, is_vertex_marked) and for accessing the neighboring convex hull triangle (ch_neighbor). The marks are stored in arrays allocated parallel to the vertex and the triangle arrays.

The contributions to measurements are computed by another three auxiliary functions, do_ch_vertex, do_ch_edge, and do_ch_triangle. For these contributions, it is necessary to compute the angle at a vertex, edge, or triangle outside the convex hull. Note, however, that this needs to be done only for convex hull vertices, edges, and triangles that belong to $\mathcal{K}$. For a triangle, the angle is $\frac{1}{2}$. For an edge, the angle is the full angle, 1, minus the dihedral angle at the edge inside the convex hull. For a vertex, i, the angle is $\frac{1}{2} - \frac{1}{4}\text{degree}(i) + \frac{1}{2}\sum \varphi_{ij}$, where φ_{ij} is the outside angle at the convex hull edge ij, and the sum extends over all degree(i) convex hull edges incident to i. This follows from Descartes' formula for solid angles, normalized to angles of revolution. Instead of computing the angle at a vertex, we distribute its contribution over the vertex itself (the $\frac{1}{2}$ part) and all incident convex hull edges ($\frac{1}{2}\varphi_{ij} - \frac{1}{4}$ per edge ij). All these computations are made somewhat more cumbersome by separate accumulations of area contributions due to individual balls.

There are two reasons for implementing both ways to measure the outside fringe. The first is that one tends to be fast when the other is slow; naturally, fringe_measurements_cx is slow when $\mathcal{K}$ contains many tetrahedra and fast when $\mathcal{K}$ contains few tetrahedra, and the opposite is true for fringe_measurements_ov. The second reason is that we can compare the results and if they match consider this as evidence that the computations are done correctly. These comparisons are done in volbl_checking_of.

III.9 Surface area contributions. The surface of the space filling diagram $\bigcup B$ consists of patches of spheres, each being part of the boundary of a ball in B. A single ball may contribute an arbitrary number of such patches, and its *area contribution* is the total area of all its patches. Each term in the inclusion-exclusion formula belongs to a group of 1, 2, 3, or 4 balls, and can be split into the same number of terms, each attributed to a single ball. The total contribution of a single ball is then a partial sum in the inclusion-exclusion formula. When the surface area of $\bigcup B$ is computed, we distribute the terms to the individual balls and keep track of partial sums in the array $Ac_{\mathsf{sf}}[1..n]$; its ith element accumulates the contributions of ball i. The same thing can be done for voids and for the outside fringe; the corresponding partial sums are accumulated in arrays $Ac_{\mathsf{tv}}[1..n]$ and $Ac_{\mathsf{of}}[1..n]$. The results are saved on a file by save_contributions, and an initialization to 0 of all contributions in all three arrays is done in initialize_contributions.

Similar to other measurements, we can cross-check the contributions of individual balls to gain confidence in the computed numbers. This is done in function volbl_checking and in volbl_checking_of. The former function tests the following relations:

$$\begin{aligned}
\max_{1\le i\le n}\{\text{abs}(Ac_{\mathsf{sf}}[i]-Ac_{\mathsf{tv}}[i]-Ac_{\mathsf{of}}[i])\} &= 0,\\
A_{\mathsf{sf}}-\sum_{i=1}^{n} Ac_{\mathsf{sf}}[i] &= 0,\\
A_{\mathsf{tv}}-\sum_{i=1}^{n} Ac_{\mathsf{tv}}[i] &= 0,\\
A_{\mathsf{of}}-\sum_{i=1}^{n} Ac_{\mathsf{of}}[i] &= 0.
\end{aligned}$$

In volbl_checking_of, the individual area contributions as computed by fringe_measurements_cx are compared with those computed by fringe_measurements_ov.

It is possible, in principle, to compute individual area contributions of balls per void, and also individual volume contributions, or individual length contributions of circles. All these measures seem to be of little interest though, and are thus not implemented.

IV.10 Finding voids. The voids of $\mathcal{K}$ are constructed using a union-find data structure for tetrahedra. We take advantage of a linear array, called ML for master-list, which stores the simplices of $\mathcal{R}$ in sorted order. It is part of the representation of $\mathcal{R}$ generated by mkalf. Given an index m, all tetrahedra up to position m in ML belong to the corresponding dual complex, $\mathcal{K}$, and all tetrahedra after position m belong to $\mathcal{R}-\mathcal{K}$. Similarly, for triangles, edges, and vertices, although there is a complication because they can stored up to three times in ML. The first occurrence marks the transition from not in $\mathcal{K}$ to singular in $\mathcal{K}$, the second is the transition from singular to regular in $\mathcal{K}$, and the third occurrence is the transition from regular to interior in $\mathcal{K}$, see [7]. Some of these occurrences might coincide depending on the local surrounding of the simplex. For computing voids, we are interested in first occurrences of triangles. Let n be the last index of ML and m the index corresponding to $\mathcal{K}$. The voids are computed in function find_voids as follows.

```
for $i := n$ downto $m+1$ do let $\sigma$ be the simplex stored in $ML[i]$;
        if $\sigma$ is a tetrahedron then uf_add($\sigma$)
        elseif $\sigma$ is a triangle then if alf_ml_is_first($\sigma$) then
                find incident tetrahedra $\tau_1, \tau_2$;
                uf_union($\tau_1, \tau_2$)
            endif
        endif
endfor.
```

After computing all voids, a priority queue is created for convenience. Each element of the queue has two fields: a pointer to the root of the corresponding void (set) in the union-find data structure, and the initial volume of the void, which is the sum of volumes of its tetrahedra computed in function initialize_volume. The priority queue is implemented as a linear array. The sorting is done by the functions sort_by_volume and heapify based on the heapsort algorithm, see [2]. Basic operations in the sorting algorithm, such as exchanging two elements, are conveniently defined as macros.

IV.11 Union-find. As explained in IV.10, the voids of the dual complex are determined using a union-find data structure for the tetrahedra in $\mathcal{R} - \mathcal{K}$. Since union-find data structures are fairly well understood and part of every standard algorithms text, see e.g. [2], we will be brief.

The basic data structure is a linear array. Each element represents a tetrahedron or it is blank. The tetrahedra are organized in sets. To represent a set, each of its elements has a pointer to its first element, the *root*, and a pointer to the next element. Initially, an array of blank elements is created by uf_create. A tetrahedron is added as a set by itself using uf_add. It initializes the root- and next-pointers. Given the array index of a tetrahedron, uf_find returns the root-pointer, which can be considered the name of the set that contains the tetrahedron. In our implementation, adding and finding takes constant time each.

Two sets are merged by uf_union. In general, this is done by changing the root-pointers of the elements in the smaller set to point to the root of the larger set. An exception is the set of tetrahedra that belong to the unbounded component of the complement. Its root is the array element with index 0, which does not represent any tetrahedron but rather the space outside $\mathcal{R}$. Whenever another set is merged with the outside set the root-pointers of the former set are changed to 0.

```
uf_union($i, j$);
  $i :=$ uf_find($i$);   $j :=$ uf_find($j$);
  if $i \neq j$ then if set $i$ is smaller than set $j$ or $j = 0$ then switch $i$ and $j$ endif;
                 change the root-pointers of all elements in set $j$ to $i$;
                 adjust next-pointers of the root of $i$ and the last element of set $j$
  endif.
```

An effective way to store the size of a set is to use the root-pointer of its root, which is otherwise unused. Note that the root-pointer of each element is changed at most $1 + \log_2 n$ times, where n is the number of tetrahedra. This is because each change either moves the element to a set at least twice as large as the old set, or it moves it to set 0, which happens only once. The total running time is therefore in $\mathrm{O}(n \log n)$. Although faster union-find structures are described in [2], the above method is used because of it is simple, it provides convenient access to the elements of a set.

IV.12 Library functions. Most of the computations described above need access to the data structures provided by programs regtri and mkalf of the alpha shape software. We list and briefly describe the functions that facilitate this access.

Function alf_is_in_complex decides whether or not a vertex, edge, triangle, or tetrahedron $\sigma \in \mathcal{R}$ belongs to $\mathcal{K}$. A similar test is function is_tetra_in_voids, which decides whether or not a tetrahedron σ belongs to a void of $\mathcal{K}$. As indicated by the missing alf prefix, the latter function is not yet part of the alpha shape library because it accesses the union-find data structure constructed solely for measuring purposes. A vertex, edge, or triangle of $\mathcal{K}$ can be part of the boundary or lie inside $\bigcup\mathcal{K}$. These two cases are distinguished by function alf_is_interior. A vertex, edge, or triangle can have up to three occurrences in the master-list, ML. The function alf_ml_is_first distinguishes the first occurrence from the others. The alpha shape software ignores duplicate and redundant input points without eliminating them physically. Function alf_is_redundant can be used to distinguish points that have been ignored in the construction of $\mathcal{R}$ from others. The function alf_is_on_hull can be used to distinguish vertices and edges that lie on the boundary of the convex hull, $\bigcup\mathcal{R}$, from others. The library also provides two functions that find incident simplices: alf_tetra_index returns the tetrahedron on the positive side of a triangle or edge-facet, and alf_tetra_vertices returns the vertices of a tetrahedron. For some computations it is necessary to test whether or not a vertex is hidden by an edge or an edge is hidden by a triangle. Technically, this means the hidden face appears in the complex not before the simplex that hides it. This is related to the definition of attachedness, see [5, 7], namely a face is attached if it is hidden by at least one simplex. These tests are done by functions is_hidden0, is_hidden1, and is_hidden2. Finally, there are two functions that translate between the two ways of specifying $\mathcal{K}$; for a given rank, alf_interval returns the interval of corresponding alpha values, and for a given alpha value, alf_rank returns the corresponding rank.

V.13 Metric functions. The terms of the sums in III.5 through III.9 are translated into metric sizes by means of a network of 54 functions. These functions are clustered in six groups presented in V.14 through V.19. The first two groups collect functions representing terms of the sums. To give an overview of the network, we begin by listing the groups and functions. The parenthesized number behind each function is its height within the network. The acyclicity of this network can be checked by verifying that the parenthesized numbers decrease from a function to all functions it calls.

14 Functions that correspond to terms of the straight inclusion-exclusion formulas.
ball_volume (2), -_area (1), -_radius (0); ball2_volume (7), -_area (4), -_length (5); ball3_volume (8), -_area (5), -_length (6); ball4_volume (8), -_area (5), -_length (6).

15 Functions that correspond to terms of the decomposable formulas.
sector_volume (4), -_area (4); wedge_volume (8), -_area (5), -_length (5); pawn_volume (9), -_area (6), -_length (7); tetrahedron_volume (1).

16 Functions that measure three-dimensional objects, such as caps and intersections of caps.
cap_volume (6), -_area (3), -_height (2); cap2_volume (7), -_area (4); cap3_volume (7), -_area (4).

17 Functions that measure two-dimensional objects, such as disk, segments, and intersections of segments.
disk_area (5), -_length (4), -_radius (3); segment_area (6), -_angle(3), -_length (5), -_height (4); segment2_area (6), -_angle(3), -_length (5).

18 Functions that compute orthogonal centers.
center4 (1), center3 (1), center2 (1); triangle_dual (2).

19 Functions for three-dimensional vector computations.
angle_solid (3), -_dihedral (2); vector_diff (0), -_sum (0), -_scaling (0); distance (1), scalar_product (0), cross_product (1); ccw (1); det4 (0), det3 (0), det2 (0).

All computations are done in double-precision floating-point, which is globally controlled by defining Vol_real equal to the type double. Precision errors are an inescapable evil that comes with the imperfect floating-point representation of real numbers. We use functions volbl_warning and volbl_correct to alert the user about incidences where the result is dangerously close to a limiting value, and to pull it back if it exceeds the limit. The warning messages are saved in a .warn file in the data directory.

V.14 Balls and intersections of balls. In this section we discuss functions that correspond to terms of the straight inclusion-exclusion formulas presented in III.5. They measure the intersection of one, two, three, and four balls.

V.14.1 Balls. A ball is given by its center, i, and its radius. These parameters uniquely determine its volume, and its surface area. The formulas require a real number, π, that is approximately 3.1415 92653 58979 32384 62643.

ball_volume(i) : Vol_real; return $\frac{1}{3} *$ ball_radius(i) $*$ ball_area(i).

ball_area(i) : Vol_real; return $4 * \pi *$ ball_radius^2(i).

In the unweighted case, the radius of a ball is α. In the weighted case, we use the more complicated formula that computes the radius as discussed in II.2. By choice of the complex, $\mathcal{K}$, all expressions under the root are (supposedly) automatically non-negative.

```
ball_radius($i$) : Vol_real;   return $\sqrt{B[i].w^2\text{sgn}(B[i].w) + \alpha^2\text{sgn}(\alpha)}$.
```

V.14.2 Intersection of two balls. The intersection of two balls is decomposed into two pieces, called *caps*, by the plane containing the circle where the bounding spheres intersect, see figure 3. By the *length* of the intersection we mean the length of this circle.

```
ball2_volume($i, j$) : Vol_real;  return cap_volume($i; j$) + cap_volume($j; i$).

ball2_area($i, j$) : Vol_real;  return cap_area($i; j$) + cap_area($j; i$).

ball2_length($i, j$) : Vol_real;  return disk_length($i, j$).
```

V.14.3 Intersection of three balls. The intersection of three balls is decomposed into three pieces, each being the intersection of two caps that belong to a common ball, see figure 3. By a *segment* we mean the part of a two-dimensional disk cut off by a line; its *length* is the length of the bounding circle arc.

```
ball3_volume($i, j, k$) : Vol_real;
  return cap2_volume($i; j, k$) + cap2_volume($j; i, k$) + cap2_volume($k; i, j$).

ball3_area($i, j, k$) : Vol_real;
  return cap2_area($i; j, k$) + cap2_area($j; i, k$) + cap2_area($k; i, j$).

ball3_length($i, j, k$) : Vol_real;
  return segment_length($i, j; k$) + segment_length($i, k; j$) + segment_length($j, k; i$).
```

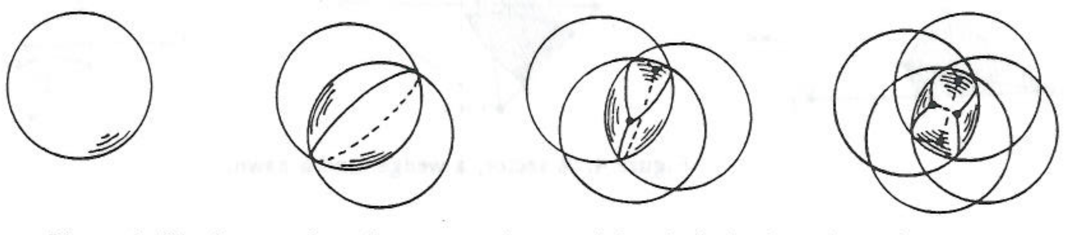
Figure 3: The intersection of one, two, three, and four balls in three dimensions.

V.14.4 Intersection of four balls. The intersection of four balls is decomposed into four pieces, each being the intersection of three caps that belong to a common ball, see figure 3.

```
ball4_volume($i, j, k, \ell$) : Vol_real;
  return  cap3_volume($i; j, k, \ell$) + cap3_volume($j; i, k, \ell$)
          +cap3_volume($k; i, j, \ell$) + cap3_volume($\ell; i, j, k$).

ball4_area($i, j, k, \ell$) : Vol_real;
  return  cap3_area($i; j, k, \ell$) + cap3_area($j; i, k, \ell$)
          +cap3_area($k; i, j, \ell$) + cap3_area($\ell; i, j, k$).
```

The total length of the six circle arcs can be computed by considering the disks in the six planes used to decompose the intersection of four balls. Each disk carries two segments, and we need the length of the arc bounding their intersection.

```
ball4_length($i, j, k, \ell$) : Vol_real;
  return  segment2_length($i, j; k, \ell$) + segment2_length($i, k; \ell, j$)
          +segment2_length($i, \ell; j, k$) + segment2_length($j, k; i, \ell$)
          +segment2_length($j, \ell; k, i$) + segment2_length($k, \ell; i, j$).
```

V.15 Pieces of balls and of intersections of balls. Except for the volume of a tetrahedron, all top level functions needed for the decomposable formulas in III.6 measure angular pieces of the intersection of one, two, and three balls.

V.15.1 Sectors. A *sector* is the intersection of a ball with a triangular cone with apex at the center of the ball, see figure 4. It is given by four points, i, j, k, ℓ, where i is the center of the ball and the half-lines starting at i through j, k, and ℓ are the edges of the cone. The volume of the sector and its area (the spherical part not including the cone sides) are the fractions determined by the solid angle at i of the volume and area of the ball.

```
sector_volume($i; j, k, \ell$) : Vol_real;
  return angle_solid($i; j, k, \ell$) * ball_volume($i$).

sector_area($i; j, k, \ell$) : Vol_real;
  return angle_solid($i; j, k, \ell$) * ball_area($i$).
```

V.15.2 Wedges. A *wedge* is the intersection of two balls and two half-spaces, see figure 4. The planes bounding the half-spaces meet in a line through the centers of the two balls. The centers are i and j, and the planes are spanned by the triangles ijk and $ij\ell$. The points i, j, k, ℓ are converted to vectors s, t, u, v.

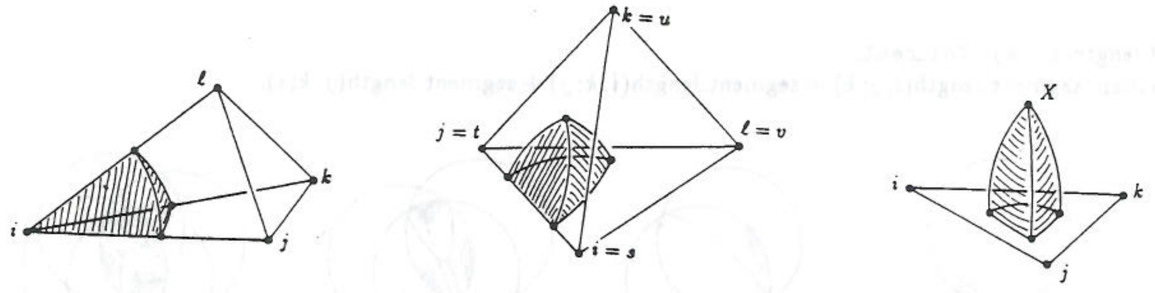

Figure 4: A sector, a wedge, and a pawn.

```
wedge_volume($i, j; k, \ell$) : Vol_real;
  $s$ := vector($i$);  $t$ := vector($j$);  $u$ := vector($k$);  $v$ := vector($\ell$);
  return angle_dihedral($s, t; u, v$) * ball2_volume($i, j$).

wedge_area($i, j; k, \ell$) : Vol_real;
  $s$ := vector($i$);  $t$ := vector($j$);  $u$ := vector($k$);  $v$ := vector($\ell$);
  return angle_dihedral($s, t; u, v$) * ball2_area($i, j$).

wedge_length($i, j; k, \ell$) : Vol_real;
  $s$ := vector($i$);  $t$ := vector($j$);  $u$ := vector($k$);  $v$ := vector($\ell$);
  return angle_dihedral($s, t; u, v$) * disk_length($i, j$).
```

V.15.3 Pawns. A *pawn* is half of the intersection of three balls, i, j, k, see figure 4. The half is on one side of the plane through i, j, k.

```
pawn_volume($i, j, k$) : Vol_real;  return $\frac{1}{2}$ * ball3_volume($i, j, k$).

pawn_area($i, j, k$) : Vol_real;  return $\frac{1}{2}$ * ball3_area($i, j, k$).

pawn_length($i, j, k$) : Vol_real;  return $\frac{1}{2}$ * ball3_length($i, j, k$).
```

V.15.4 Tetrahedra. A tetrahedron is given by four vertices, i, j, k, ℓ. Its volume can be computed using the determinant of the point coordinates. Specifically, the volume is one sixth the absolute value of the four-by-four determinant whose row entries are the three coordinates and 1. This function uses floating-point arithmetic and is not to be used for any positional tests related to constructing $\mathcal{R}$ and $\mathcal{K}$.

```
tetrahedron_volume($i, j, k, \ell$) : Vol_real;
  return abs($\frac{1}{6} * \det \begin{pmatrix} B[i].x & B[i].y & B[i].z & 1 \\ B[j].x & B[j].y & B[j].z & 1 \\ B[k].x & B[k].y & B[k].z & 1 \\ B[\ell].x & B[\ell].y & B[\ell].z & 1 \end{pmatrix}$).
```

V.16 Caps and intersections of caps. The objects computed in this group are intersections of one, two, and three caps that belong to a common ball.

V.16.1 Caps. A cap is defined by two balls, i and j. It is the part of the intersection of the two balls that lies on i's side of the plane through the intersection of the two bounding spheres. The volume of the cap is obtained by subtracting the cone from the sector.

```
cap_volume($i; j$) : Vol_real;
  $S := \frac{1}{3}$ * ball_radius($i$) * cap_area($i; j$);
  $C := \frac{1}{3}$ * [ball_radius($i$) − cap_height($i; j$)] * disk_area($i, j$);
  return $S - C$.
```

For the cap area there is a neat formula that says it is the area of the entire sphere times the cap height over the diameter of the ball, see [8]. This simplifies to

```
cap_area($i; j$) : Vol_real;  return $2 * \pi$ * ball_radius($i$) * cap_height($i; j$).
```

The height of the cap is either the radius of ball i minus the distance between its center and the orthogonal center of i and j, or it is the radius plus the distance. The former case occurs when i is not hidden by j, and the other case occurs when i is hidden by j. See V.18 for a description of orthogonal centers.

```
cap_height($i; j$) : Vol_real;
  $s$ := vector($i$);  $Y$ := center2($i, j$);
  if alf_is_hidden0($i; j$) then return ball_radius($i$) + distance($s, Y$)
                         else ball_radius($i$) − distance($s, Y$)
  endif.
```

V.16.2 Intersection of two caps. The two caps are part of the same ball, i, and are determined by i, j and by i, k, see figure 5. We compute the volume by subtracting fractions of two cones from the intersection of the two sectors. The volume of the sector intersection is one third of the ball radius times its area. The fraction of one cone is defined by the area of the base segment.

```
cap2_volume($i; j, k$) : Vol_real;
    $S2 := \frac{1}{3} * \text{ball\_radius}(i) * \text{cap2\_area}(i; j, k)$;
    $C_j := \frac{1}{3} * [\text{ball\_radius}(i) - \text{cap\_height}(i; j)] * \text{segment\_area}(i, j; k)$;
    $C_k := \frac{1}{3} * [\text{ball\_radius}(i) - \text{cap\_height}(i; k)] * \text{segment\_area}(i, k; j)$;
    return $S2 - C_j - C_k$.
```

We compute the area of the intersection of two caps step by step. The implicit formula can be proved using a limit process approximating the intersection by a spherical polygon, see appendix A. The formula has also been used in [3]. We compute points p_{jk} and p_{kj} so that p_{jk} sees (i, j, k) in a counterclockwise (ccw) order, and p_{kj} sees (i, k, j) in a ccw order, see figure 5. Recall that the dihedral angle between two triangles sharing an edge is the smaller of the two possible angles.

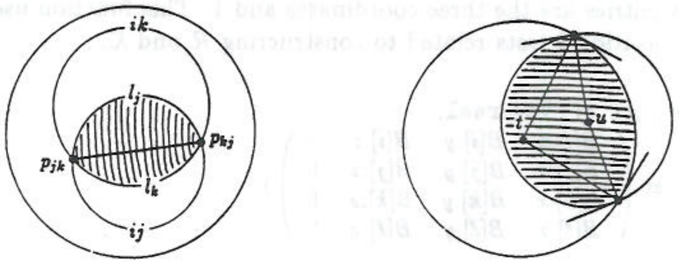

Figure 5: The intersection of two caps.

```
cap2_area($i; j, k$) : Vol_real;
    $p_{jk} := \text{triangle\_dual}(i; j; k)$;  $p_{kj} := \text{triangle\_dual}(i; k; j)$;
    $l_j := \text{segment\_angle}(i, j; k)$;  $l_k := \text{segment\_angle}(i, k; j)$;
    $s := \text{vector}(i)$;  $t := \text{vector}(j)$;  $u := \text{vector}(k)$;
    $\varphi_{jk} := \frac{1}{2} - \text{angle\_dihedral}(s, p_{jk}; t, u)$;  $\varphi_{kj} := \frac{1}{2} - \text{angle\_dihedral}(s, p_{kj}; u, t)$;
    $A_1 := \frac{1}{2} * \text{ball\_area}(i) * (\varphi_{jk} + \varphi_{kj})$;
    $A_2 := 2 * \pi * \text{ball\_radius}(i) * l_j * [\text{ball\_radius}(i) - \text{cap\_height}(i; j)]$;
    $A_3 := 2 * \pi * \text{ball\_radius}(i) * l_k * [\text{ball\_radius}(i) - \text{cap\_height}(i; k)]$;
    return $A_1 - A_2 - A_3$.
```

Note that in the absence of round-off errors we have $\varphi_{jk} = \varphi_{kj}$.

V.16.3 Intersection of three caps. We consider intersections of three caps common to a ball i, see figure 6. The cone is the intersection of three half-spaces defined by i, j and i, k and i, ℓ. It is assumed that the apex of the thus defined triangular cone lies inside i. That this is indeed always the case is a property of the dual complex of $\bigcup B$. The volume is the difference of the intersection of three sectors and pieces of three cones.

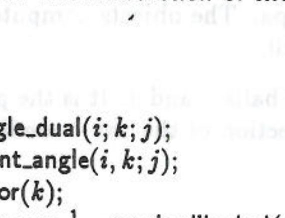

Figure 6: The intersection of three caps.

```
cap3_volume($i; j, k, \ell$) : Vol_real;
  $S3 := \frac{1}{3} * \mathsf{ball\_radius}(i) * \mathsf{cap3\_area}(i; j, k, \ell)$;
  $C_j := \frac{1}{3} * [\mathsf{ball\_radius}(i) - \mathsf{cap\_height}(i; j)] * \mathsf{segment2\_area}(i, j; k, \ell)$;
  $C_k := \frac{1}{3} * [\mathsf{ball\_radius}(i) - \mathsf{cap\_height}(i; k)] * \mathsf{segment2\_area}(i, k; j, \ell)$;
  $C_\ell := \frac{1}{3} * [\mathsf{ball\_radius}(i) - \mathsf{cap\_height}(i; \ell)] * \mathsf{segment2\_area}(i, \ell; j, k)$;
  return $S3 - C_j - C_k - C_\ell$.
```

The formula for the area is a minor generalization of the one for the intersection of two caps, see appendix A. We first make sure that point i sees (j, k, ℓ) is a ccw order.

```
cap3_area($i; j, k, \ell$) : Vol_real;
  if not ccw($i; j; k; \ell$) then exchange $k$ and $\ell$ endif;
  $p_{kj} := \mathsf{triangle\_dual}(i; k; j)$;  $p_{\ell k} := \mathsf{triangle\_dual}(i; \ell; k)$;
                                    $p_{j\ell} := \mathsf{triangle\_dual}(i; j; \ell)$;
  $l_j := \mathsf{segment2\_angle}(i, j; k, \ell)$;  $l_k := \mathsf{segment2\_angle}(i, k; \ell, j)$;
                                    $l_\ell := \mathsf{segment2\_angle}(i, \ell; j, k)$;
  $s := \mathsf{vector}(i)$;  $t := \mathsf{vector}(j)$;  $u := \mathsf{vector}(k)$;  $v := \mathsf{vector}(\ell)$;
  $\varphi_{kj} := \frac{1}{2} - \mathsf{angle\_dihedral}(s, p_{kj}; u, t)$;  $\varphi_{\ell k} := \frac{1}{2} - \mathsf{angle\_dihedral}(s, p_{\ell k}; v, u)$;
                                    $\varphi_{j\ell} := \frac{1}{2} - \mathsf{angle\_dihedral}(s, p_{j\ell}; t, v)$;
  $A_1 := \frac{1}{2} * \mathsf{ball\_area}(i) * (\varphi_{kj} + \varphi_{\ell k} + \varphi_{j\ell} - \frac{1}{2})$;
  $A_2 := 2 * \pi * \mathsf{ball\_radius}(i) * l_j * [\mathsf{ball\_radius}(i) - \mathsf{cap\_height}(i; j)]$;
  $A_3 := 2 * \pi * \mathsf{ball\_radius}(i) * l_k * [\mathsf{ball\_radius}(i) - \mathsf{cap\_height}(i; k)]$;
  $A_4 := 2 * \pi * \mathsf{ball\_radius}(i) * l_\ell * [\mathsf{ball\_radius}(i) - \mathsf{cap\_height}(i; \ell)]$;
  return $A_1 - A_2 - A_3 - A_4$.
```

V.17 Disks, segments, and intersections of segments. We now discuss a group of functions that measure two-dimensional objects, such as a disk and a segment of a disk.

V.17.1 Disks. A *disk* is given by two balls, i and j. It is the intersection of the two balls with the plane that contains the circle where the bounding spheres intersect.

```
disk_area($i, j$) : Vol_real;  return $\frac{1}{2} * \mathsf{disk\_radius}(i, j) * \mathsf{disk\_length}(i, j)$.
```

```
disk_length($i, j$) : Vol_real;  return $2 * \pi * \mathsf{disk\_radius}(i, j)$.
```

The radius of the disk can be computed from the two ball radii and the distance between the centers. Specifically, the disk radius forms a right angled triangle with one ball radius and the same ball radius minus the cap height. This leads to the following formula.

```
disk_radius($i, j$) : Vol_real;
  return $\sqrt{\mathsf{cap\_height}(i; j) * [2 * \mathsf{ball\_radius}(i) - \mathsf{cap\_height}(i; j)]}$.
```

V.17.2 Segments. A *segment* is the intersection of a disk with a half-plane. Given i, j, k in this order, the segment is the part of the disk defined by i and j cut off by the line through the intersection of the bounding circle with the sphere bounding k. The area is computed as the difference between a disk sector and a triangle.

```
segment_area($i, j; k$) : Vol_real;
  $S := \frac{1}{2} * \text{disk\_radius}(i, j) * \text{segment\_length}(i, j; k)$;
  $p_{jk} := \text{triangle\_dual}(i; j; k)$;  $p_{kj} := \text{triangle\_dual}(i; k; j)$;
  $H := \text{disk\_radius}(i, j) - \text{segment\_height}(i, j; k)$;
  $T := \frac{1}{2} * H * \text{distance}(p_{jk}, p_{kj})$;
  return $S - T$.
```

The length of the arc bounding the segment is computed by taking the appropriate fraction of the circle. This is done in two steps, first computing the normalized length and then the actual length of the segment.

```
segment_angle($i, j; k$) : Vol_real;
  $p_{jk} := \text{triangle\_dual}(i; j; k)$;  $p_{kj} := \text{triangle\_dual}(i; k; j)$;
  $s := \text{vector}(i)$;  $t := \text{vector}(j)$;  $u := \text{vector}(k)$;
  return $\text{angle\_dihedral}(s, t; u, p_{jk}) + \text{angle\_dihedral}(s, t; u, p_{kj})$.

segment_length($i, j; k$) : Vol_real;
  return $\text{segment\_angle}(i, j; k) * \text{disk\_length}(i, j)$.
```

The height of the segment is computed from the radius of the disk defined by i and j and the distance of the orthogonal center of ijk from the line through i and j. This is the same as the distance to the orthogonal center of ij. The cases where the segment covers more or less than half the disk are distinguished by checking whether or not ij is attached to ijk, see figure 7. This test, alf_is_hidden, is also used in the construction of $\mathcal{K}$; it is based on integer arithmetic.

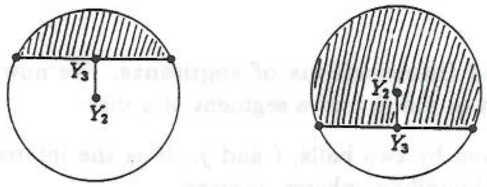

Figure 7: The segment covering less and more than half of the disk.

```
segment_height($i, j; k$) : Vol_real;
  $Y_3 := \text{center3}(i, j, k)$;  $Y_2 := \text{center2}(i, j)$;
  if alf_is_hidden1($i, j; k$) then return $\text{disk\_radius}(i, j) + \text{distance}(Y_2, Y_3)$
                                else return $\text{disk\_radius}(i, j) - \text{distance}(Y_2, Y_3)$
  endif.
```

The height of the segment is necessarily between 0 and twice the disk radius.

V.17.3 Intersection of two segments. We consider two segments, both part of the disk defined by i and j. One is defined by k and one by ℓ. The intersection is computed as the difference between the disk sector and two triangles. The bases of the two triangles connect dual points of the triangles ijk and $ij\ell$ with the orthogonal center of i, j, k, ℓ. First we make sure that point i sees (j, k, ℓ) is a ccw order.

```
segment2_area($i, j; k, \ell$) : Vol_real;
  if not ccw($i; j; k; \ell$) then exchange $k$ and $\ell$ endif;
  $p_{jk} := \text{triangle\_dual}(i; j; k)$;  $p_{kj} := \text{triangle\_dual}(i; k; j)$;
  $p_{j\ell} := \text{triangle\_dual}(i; j; \ell)$;  $p_{\ell j} := \text{triangle\_dual}(i; \ell; j)$;
```

```
    $Y$ := center4($i, j, k, \ell$);
    $h_k$ := segment_height($i, j; k$);  $h_\ell$ := segment_height($i, j; \ell$);
    $r_{ij}$ := disk_radius($i, j$);
    $S := \frac{1}{2} * r_{ij} *$ segment2_length($i, j; k, \ell$);
    $T_k := \frac{1}{2} * (r_{ij} - h_k) *$ distance($p_{kj}, Y$);  $T_\ell := \frac{1}{2} * (r_{ij} - h_\ell) *$ distance($p_{j\ell}, Y$);
    return $S - T_k - T_\ell$.
```

For computing the length of the intersection of two segments defined by i, j, k, ℓ, we assume that the two segments intersect but are not nested. By the choice of i, j, k, ℓ, as guided by the dual complex of $\bigcup B$, this is indeed always the case. The length is obtained by subtracting the overlap from the sum of the two half-segments. This is done in two steps, first computing the normalized and then the actual length.

```
segment2_angle($i, j; k, \ell$) : Vol_real;
  $p_{j\ell}$ := triangle_dual($i; j; \ell$);  $p_{kj}$ := triangle_dual($i; k; j$);
  $s$ := vector($i$);  $t$ := vector($j$);  $u$ := vector($k$);  $v$ := vector($\ell$);
  return angle_dihedral($s, t; u, p_{kj}$) + angle_dihedral($s, t; v, p_{j\ell}$) − angle_dihedral($s, t; u, v$).

segment2_length($i, j; k, \ell$) : Vol_real;
  return segment2_angle($i, j; k, \ell$) * disk_length($i, j$).
```

V.18 Orthogonal centers. Two balls are *orthogonal* if the square of the distance between their centers equals the sum of their weights. The *orthogonal center* of a collection of balls is the center of the smallest ball orthogonal to all balls in the collection. In the special case where all balls in the collection have radius zero, the orthogonal center is the center of the smallest sphere through the points. We need orthogonal centers of four, three, and two balls.

V.18.1 Orthogonal center of four balls. In the case of four balls, i, j, k, ℓ, there is a unique ball orthogonal to all four balls; its center, Y, is the orthogonal center of i, j, k, ℓ. Y is computed from the following system of four linear equations. Each equation constrains the variable ball be orthogonal to one of the original balls. We use indices 1 through 4 for coordinates x, y, z, w and define $l_0 = \frac{1}{2}(l_4^2 \text{sgn}(l_4) - l_1^2 - l_2^2 - l_3^2)$, for $l = x, i, j, k, \ell$.

$$\begin{array}{ccccccccc} i_0 x_1 & + & i_2 x_2 & + & i_3 x_3 & + & x_0 & = & -i_0 \\ j_0 x_1 & + & j_2 x_2 & + & j_3 x_3 & + & x_0 & = & -j_0 \\ k_0 x_1 & + & k_2 x_2 & + & k_3 x_3 & + & x_0 & = & -k_0 \\ \ell_0 x_1 & + & \ell_2 x_2 & + & \ell_3 x_3 & + & x_0 & = & -\ell_0 \end{array}$$

The solutions for x_1, x_2, x_3 are computed using Cramer's rule. The solution for x_0 is not needed.

```
center4($i, j, k, \ell$) : Vector.
  $i_0 := \frac{1}{2}(B[i].w^2 \text{sgn}(B[i].w) - B[i].x^2 - B[i].y^2 - B[i].z^2)$;
  $j_0 := \frac{1}{2}(B[j].w^2 \text{sgn}(B[j].w) - B[j].x^2 - B[j].y^2 - B[j].z^2)$;
  $k_0 := \frac{1}{2}(B[k].w^2 \text{sgn}(B[k].w) - B[k].x^2 - B[k].y^2 - B[k].z^2)$;
  $\ell_0 := \frac{1}{2}(B[l].w^2 \text{sgn}(B[l].w) - B[\ell].x^2 - B[\ell].y^2 - B[\ell].z^2)$;
  $D_0 := \det \begin{pmatrix} B[i].x & B[i].y & B[i].z & 1 \\ B[j].x & B[j].y & B[j].z & 1 \\ B[k].x & B[k].y & B[k].z & 1 \\ B[\ell].x & B[\ell].y & B[\ell].z & 1 \end{pmatrix}$;  $D_x := \det \begin{pmatrix} -i_0 & B[i].y & B[i].z & 1 \\ -j_0 & B[j].y & B[j].z & 1 \\ -k_0 & B[k].y & B[k].z & 1 \\ -\ell_0 & B[\ell].y & B[\ell].z & 1 \end{pmatrix}$;
  $D_y := \det \begin{pmatrix} B[i].x & -i_0 & B[i].z & 1 \\ B[j].x & -j_0 & B[j].z & 1 \\ B[k].x & -k_0 & B[k].z & 1 \\ B[\ell].x & -\ell_0 & B[\ell].z & 1 \end{pmatrix}$;  $D_z := \det \begin{pmatrix} B[i].x & B[i].y & -i_0 & 1 \\ B[j].x & B[j].y & -j_0 & 1 \\ B[k].x & B[k].y & -k_0 & 1 \\ B[\ell].x & B[\ell].y & -\ell_0 & 1 \end{pmatrix}$;
  $Y.x := \frac{D_x}{D_0}$;  $Y.y := \frac{D_y}{D_0}$;  $Y.z := \frac{D_z}{D_0}$;
  return $Y$.
```

V.18.2 Orthogonal center of three balls. In the case of three balls, i, j, k, the orthogonal center, Y, of i, j, k is the center of the smallest ball orthogonal to i, j, and k. Again, it is computed from a system of four linear equations. We reuse the first three equations from V.18.1 and replace the fourth by the constraint that Y be coplanar with i, j, k:

$$A_1x_1 + A_2x_2 + A_3x_3 = A_4,$$

where

$$A_1 = \det\begin{pmatrix} i_2 & i_3 & 1 \\ j_2 & j_3 & 1 \\ k_2 & k_3 & 1 \end{pmatrix},\ A_2 = \det\begin{pmatrix} i_3 & i_1 & 1 \\ j_3 & j_1 & 1 \\ k_3 & k_1 & 1 \end{pmatrix},$$

$$A_3 = \det\begin{pmatrix} i_1 & i_2 & 1 \\ j_1 & j_2 & 1 \\ k_1 & k_2 & 1 \end{pmatrix},\ A_4 = \det\begin{pmatrix} i_1 & i_2 & i_3 \\ j_1 & j_2 & j_3 \\ k_1 & k_2 & k_3 \end{pmatrix},$$

```
center3(i, j, k) : Vector.
  $A_1 := \det\begin{pmatrix} B[i].y & B[i].z & 1 \\ B[j].y & B[j].z & 1 \\ B[k].y & B[k].z & 1 \end{pmatrix};\ A_2 := \det\begin{pmatrix} B[i].z & B[i].x & 1 \\ B[j].z & B[j].x & 1 \\ B[k].z & B[k].x & 1 \end{pmatrix};$
  $A_3 := \det\begin{pmatrix} B[i].x & B[i].y & 1 \\ B[j].x & B[j].y & 1 \\ B[k].x & B[k].y & 1 \end{pmatrix};\ A_4 := \det\begin{pmatrix} B[i].x & B[i].y & B[i].z \\ B[j].x & B[j].y & B[j].z \\ B[k].x & B[k].y & B[k].z \end{pmatrix};$
  $i_0 := \frac{1}{2}(B[i].w^2\mathrm{sgn}(B[i].w) - B[i].x^2 - B[i].y^2 - B[i].z^2);$
  $j_0 := \frac{1}{2}(B[j].w^2\mathrm{sgn}(B[j].w) - B[j].x^2 - B[j].y^2 - B[j].z^2);$
  $k_0 := \frac{1}{2}(B[k].w^2\mathrm{sgn}(B[k].w) - B[k].x^2 - B[k].y^2 - B[k].z^2);$
  $D_0 := \det\begin{pmatrix} B[i].x & B[i].y & B[i].z & 1 \\ B[j].x & B[j].y & B[j].z & 1 \\ B[k].x & B[k].y & B[k].z & 1 \\ A_1 & A_2 & A_3 & 0 \end{pmatrix};\ D_x := \det\begin{pmatrix} -i_0 & B[i].y & B[i].z & 1 \\ -j_0 & B[j].y & B[j].z & 1 \\ -k_0 & B[k].y & B[k].z & 1 \\ A_4 & A_2 & A_3 & 0 \end{pmatrix};$
  $D_y := \det\begin{pmatrix} B[i].x & -i_0 & B[i].z & 1 \\ B[j].x & -j_0 & B[j].z & 1 \\ B[k].x & -k_0 & B[k].z & 1 \\ A_1 & A_4 & A_3 & 0 \end{pmatrix};\ D_z := \det\begin{pmatrix} B[i].x & B[i].y & -i_0 & 1 \\ B[j].x & B[j].y & -j_0 & 1 \\ B[k].x & B[k].y & -k_0 & 1 \\ A_1 & A_2 & A_4 & 0 \end{pmatrix};$
  $Y.x := \frac{D_x}{D_0};\ Y.y := \frac{D_y}{D_0};\ Y.z := \frac{D_z}{D_0};$
  return $(Y.x, Y.y, Y.z)$.
```

V.18.3 Orthogonal center of two balls. In the unweighted case this is just the midpoint of the two centers. In the weighted case we set $Y = \lambda i + (1-\lambda)j$ and solve the linear relation obtained by setting the power distances to i and j equal:

$$|Yi|^2 - i_4 = |Yj|^2 - j_4.$$

This implies $\lambda = \frac{1}{2} - \frac{i_4 - j_4}{2\langle i-j, i-j\rangle}$.

```
center2(i, j) : Vector;
  $s :=$ vector$(i)$; $t :=$ vector$(j)$;
  $aux :=$ scalar_product$(s - t, s - t)$;
  $i_4 := B[i].w^2\mathrm{sgn}(B[i].w)$;
  $j_4 := B[j].w^2\mathrm{sgn}(B[j].w)$;
  $\lambda := \frac{1}{2} - \frac{i_4 - j_4}{2 * aux}$;
  return $\lambda * s + (1 - \lambda) * t$.
```

V.18.4 The dual point of a triangle. The *dual point*, X, is the intersection point of the three spheres bounding i, j, k that lies on the ccw side of the oriented triangle, ijk, see the pawn in figure 4. First, we compute the orthogonal center of i, j, k. Second, we compute a normal to the plane spanned by the points i, j, k that points into the ccw side of the oriented triangle. Finally, we get X by solving a polynomial of degree 2.

Let $Y \in \mathbb{R}^3$ be the orthogonal center of i, j, k and let N be the required normal vector. We compute X by intersecting the half-line $Y + \xi N$, $\xi > 0$, with the sphere bounding i. Let r be the radius and s the center of i. The condition that $Y + \xi N$ lie on the sphere gives

$$\langle (Y + \xi N) - s, (Y + \xi N) - s \rangle = \xi^2 \langle N, N \rangle + 2\xi \langle Y - s, N \rangle + \langle Y - s, Y - s \rangle = r^2.$$

By definition of X, we need the positive root of this polynomial, which is

$$\xi = \frac{-\langle Y - s, N \rangle + \sqrt{\langle Y - s, N \rangle^2 - \langle Y - s, Y - s \rangle \langle N, N \rangle + r^2 \langle N, N \rangle}}{\langle N, N \rangle}.$$

```
triangle_dual($i, j, k$) : Vector;
  $Y$ := center3($i, j, k$);
  $s$ := vector($i$);  $t$ := vector($j$);  $u$ := vector($k$);
  $N$ := cross_product($t - s, u - s$);
  $S_1$ := scalar_product($Y - s, N$);  $S_2$ := scalar_product($N, N$);
                                        $S_3$ := scalar_product($Y - s, Y - s$);
  $r$ := ball_radius($i$);
  $\xi := \frac{-S_1 + \sqrt{S_1^2 - S_3 * S_2 + r^2 * S_2}}{S_2}$;
  return $Y + \xi * N$.
```

By construction, ξ is necessarily positive.

V.19 Vector computations. In this group we collect functions that compute angles, distances, the orientation of four points in $\mathbb{R}^3$, and other things.

V.19.1 Angles. The solid angle at a vertex i within a tetrahedron $ijk\ell$ can be reduced to computing dihedral angles. Recall that we measure angles in revolution, that is, they are scaled between 0 and 1.

```
angle_solid($i; j, k, \ell$) : Vol_real;
  $s$ := vector($i$);  $t$ := vector($j$);  $u$ := vector($k$);  $v$ := vector($\ell$);
  $\varphi_t$ := angle_dihedral($s, t; v, u$);  $\varphi_u$ := angle_dihedral($s, u; t, v$);
                                              $\varphi_v$ := angle_dihedral($s, v; u, t$);
  return $\frac{1}{2} * (\varphi_t + \varphi_u + \varphi_v) - \frac{1}{4}$.
```

The dihedral angle between two triangles meeting at an edge st is the smaller of the two angles. It is computed by constructing unit normal vectors of the two triangles, both on the ccw or both on the cw side. The scalar product of the two vectors is the cosine of the angle.

```
angle_dihedral($s, t; u, v$) : Vol_real;
  $M_u$ := cross_product($u - s, u - t$);  $N_u := \frac{1}{\sqrt{\text{scalar\_product}(M_u, M_u)}} M_u$;
  $M_v$ := cross_product($v - s, v - t$);  $N_v := \frac{1}{\sqrt{\text{scalar\_product}(M_v, M_v)}} M_v$;
  return $\frac{1}{\pi}$ * arccos[scalar_product($N_u, N_v$)].
```

V.19.2 Trivial vector manipulations. The difference or sum of two vectors and the product of a scalar with a vector are trivial operations that nonetheless need to be computed. We do this using the following three functions. Calls to these functions are not explicitly marked.

```
vector_diff($s, t$) : Vector;
  $u.x := s.x - t.x$;  $u.y := s.y - t.y$;  $u.z := s.z - t.z$;
  return $u$.
```

```
vector_sum(s, t) : Vector;
  $u.x := s.x + t.x$;  $u.y := s.y + t.y$;  $u.z := s.z + t.z$;
  return $u$.
```

```
vector_scaling(f, s) : Vector;
  $u.x := f * s.x$;  $u.y := f * s.y$;  $u.z := f * s.z$;
  return $u$.
```

V.19.3 Distance and products. The distance between two points (vectors) is reduced to the scalar product of their difference vector with itself.

```
distance(s, t) : Vol_real;  return $\sqrt{\text{scalar\_product}(s - t, s - t)}$.
```

The scalar product is defined for two vectors. It is the sum of the component-wise products.

```
scalar_product(u, v) : Vol_real;  return $u.x * v.x + u.y * v.y + u.z * v.z$.
```

Given two vectors, u and v, their cross product is another vector, V, normal to both. We design the function so that (u, v, V) is a right-handed system, that is, the endpoint of V sees a, b, c is a ccw order, where $u = b - a$ and $v = c - a$. The coordinates of V are computed using two-by-two determinants.

```
cross_product(u, v) : Vector;
  $V.x := \det\begin{pmatrix} u.y & u.z \\ v.y & v.z \end{pmatrix}$;  $V.y := \det\begin{pmatrix} u.z & u.x \\ v.z & v.x \end{pmatrix}$;  $V.z := \det\begin{pmatrix} u.x & u.y \\ v.x & v.y \end{pmatrix}$;
  return $V$.
```

V.19.4 Ccw. Given four points in $\mathbb{R}^3$, we test whether or not the first point, i, sees the sequence of other points, (j, k, ℓ), in a counterclockwise (ccw) order. The decision is based on computing a four-by-four determinant as shown below. In actual fact, the ccw test is done in integer arithmetic and it uses a simulated perturbation in cases where the determinant evaluates to zero. This is done by calling **sos_positive3** in the SoS-package used to construct $\mathcal{R}$ and $\mathcal{K}$.

```
ccw(i; j; k; ℓ) : boolean;
  return $\det\begin{pmatrix} B[i].x & B[i].y & B[i].z & 1 \\ B[j].x & B[j].y & B[j].z & 1 \\ B[k].x & B[k].y & B[k].z & 1 \\ B[\ell].x & B[\ell].y & B[\ell].z & 1 \end{pmatrix} > 0$.
```

V.19.5 Determinants. We need determinants of two-by-two, three-by-three, and four-by-four matrices. In all cases, the elements of the matrix are of type Vol_real. Since determinants are evaluated at the bottom of the computation graph, they consume the biggest fraction of the running time. For this reason the three matrix sizes are covered by three separate functions and all arithmetic is done without any loop construction.

```
det2(A) : Vol_real;
  return $A_{00}A_{11} - A_{10}A_{01}$.
```

```
det3(A) : Vol_real;
  return $A_{00}(A_{11}A_{22} - A_{21}A_{12}) - A_{10}(A_{01}A_{22} - A_{21}A_{02}) + A_{20}(A_{01}A_{12} - A_{11}A_{02})$.
```

```
det4(A) : Vol_real;
    $t_0 := A_{00}[A_{11}(A_{22}A_{33} - A_{32}A_{23}) - A_{21}(A_{12}A_{33} - A_{32}A_{13}) + A_{31}(A_{12}A_{23} - A_{22}A_{13})]$;
    $t_1 := A_{10}[A_{01}(A_{22}A_{33} - A_{32}A_{23}) - A_{21}(A_{02}A_{33} - A_{32}A_{03}) + A_{31}(A_{02}A_{23} - A_{22}A_{03})]$;
    $t_2 := A_{20}[A_{01}(A_{12}A_{33} - A_{32}A_{13}) - A_{11}(A_{02}A_{33} - A_{32}A_{03}) + A_{31}(A_{02}A_{13} - A_{12}A_{03})]$;
    $t_3 := A_{30}[A_{01}(A_{12}A_{23} - A_{22}A_{13}) - A_{11}(A_{02}A_{23} - A_{22}A_{03}) + A_{21}(A_{02}A_{13} - A_{12}A_{03})]$;
    return $t_0 - t_1 + t_2 - t_3$.
```

Acknowledgements. We thank Ernst Mücke and Michael Facello for their implementations of three-dimensional alpha complexes, which form the basis of our software. Without their dedication to create an always consistent simplicial complex this project could not succeed. We also thank Hai-Ping Cheng and Shankar Subramaniam for their encouragement and for convincing us that computing metric properties of a space filling diagram does have applications in chemistry, physics, and biology.

References

[1] S. Arizzi, P. H. Mott and U. W. Suter. Space available to small diffusants in polymeric glasses: analysis of unoccupied space and its connectivity. *J. Polymer Science* 30 (1992), 415–426.

[2] T. H. Cormen, Ch. E. Leiserson and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, Mass., 1990.

[3] T. H. Connolly. Analytical molecular surface calculation. *J. Appl. Cryst.* 16 (1983), 548–558.

[4] L. R. Dodd and D. N. Theodorou. Analytic treatment of the volume and surface area of molecules formed by an arbitrary collection of unequal spheres intersected by planes. *Molecular Physics* 72 (1991), 1313–1345.

[5] H. Edelsbrunner. Weighted alpha shapes. Rept. UIUCDCS-R-92-1760, Comput. Sci. Dept., Univ. Illinois, Urbana, Illinois, 1992.

[6] H. Edelsbrunner. The union of balls and its dual shape. *In* "Proc. 9th Ann. Sympos. Comput. Geom., 1993", 218–231.

[7] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. To appear in *ACM Trans. Graphics* (1993).

[8] W. Gellert, S. Gottwald, M. Hellwich, H. Kästner and H. Küstner. *The VNR Concise Enzyclopedia of Mathematics.* Second edition, van Nostrand Reinhold, New York, 1989.

[9] B. Lee and F. M. Richards. The interpretation of protein structures: estimation of static accessibility. *J. Mol. Biol.* 55 (1971), 379-400.

[10] G. Perrot, B. Cheng, K. D. Gibson, J. Vila, A. Palmer, A. Nayeem, B. Maigret and H. A. Scheraga. MSEED: a program for rapid determination of accessible surface areas and their derivatives. *J. Comput. Chem.* 13 (1992), 1-11.

[11] B. W. Kernighan and D. M. Ritchie. *The C Programming Language.* Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[12] F. M. Richards. Areas, volumes, packing, and protein structures. *Ann. Rev. Biophys. Bioeng.* 6 (1977), 151-176.

[13] F. M. Richards. Calculation of molecular volumes and areas for structures of known geometries. *Methods in Enzymology* 115 (1985), 440–464.

Appendix A. This appendix presents a proof of the formula for the area of the intersection of two caps used in V.16.2. The formula readily generalizes to the intersection of three or more caps. Indeed, the version that deals with the intersection of three caps is used in V.16.3. Similar formulas have also been used by Connolly [3].

We begin by recalling that p_{jk} and p_{kj} are the intersection points of the two circles on the bounding sphere of i. These circles are the intersections of this sphere with the bounding spheres of j and k. We write l_j for the fraction of the circle for j that bounds the intersection of caps, and symmetrically we write l_k for the corresponding fraction of the circle for k.

We approximate the sphere part of the intersection of caps by a spherical m-gon. For $A_i = \text{ball\_area}(i)$, the area of this m-gon is $\frac{A_i}{2}(\sum_{\lambda=1}^{m} \phi_\lambda - \frac{m-2}{2})$, where ϕ_λ is the angle at the λth vertex. This is because a triangulation produces $m-2$ spherical triangles each contributing $A_i(\frac{1}{2}(\psi_1+\psi_2+\psi_3)-\frac{1}{4})$ to the area, where ψ_1, ψ_2, ψ_3 are the angles of the generic spherical triangle. We approximate each of the two circles by a regular spherical n-gon. The points are placed slightly outside the circle so that the enclosed areas coincide. Assuming that l_j and l_k are rationals, we can find n so that the two n-gons share two vertices approximately at p_{jk} and p_{kj}. We get $m = l_j n + l_k n$. The angles at the two shared vertices approximate φ_{jk} and φ_{kj} at p_{jk} and p_{kj}. Furthermore, there are $l_j n - 1$ vertices with angle φ_j and $l_k n - 1$ vertices with angle φ_k. Next we compute φ_j and φ_k.

Consider the cap defined by the first circle. Its area is $2\pi r_i h_j$, where $r_i = \text{ball\_radius}(i)$ and $h_j = \text{cap\_height}(i;j)$. By construction, the approximating n-gon bounds the same area, which is

$$\frac{A_i}{2}(n\varphi_j - \frac{n-2}{2}) = 2\pi r_i h_j.$$

Therefore,

$$\varphi_j = \frac{4\pi r_i h_j}{A_i n} + \frac{n-2}{2n} = \frac{1}{2} - \frac{1}{n} + \frac{h_j/r_i}{n}.$$

We plug the values for φ_j and φ_k into the formula for the area of the intersection of caps. The $l_j n - 1$ angles of size φ_j add up to $(l_j n - 1)(\frac{1}{2} - \frac{1}{n} + \frac{h_j/r_i}{n})$. Similarly, the $l_k n - 1$ angles of size φ_k add up to $(l_k n - 1)(\frac{1}{2} - \frac{1}{n} + \frac{h_k/r_i}{n})$. Furthermore, we have φ_{jk} and φ_{kj}. From the sum of angles we subtract $\frac{m-2}{2} = \frac{l_j n + l_k n - 2}{2}$. This gives

$$\varphi_{jk} + \varphi_{kj} - l_j(1 - \frac{h_j}{r_i}) - l_k(1 - \frac{h_k}{r_i}) + (\frac{1}{n} - \frac{h_j}{r_i n}) + (\frac{1}{n} - \frac{h_k}{r_i n}).$$

The last two terms vanish as n goes to infinity. By multiplying this with $\frac{A_i}{2}$ we get the final result:

$$\frac{A_i}{2}(\sum_{\lambda=1}^{m} \phi_\lambda - \frac{m-2}{2}) \xrightarrow{}_{n\to\infty} \frac{A_i}{2}(\varphi_{jk} + \varphi_{kj}) - 2\pi r_i l_j (r_i - h_j) - 2\pi r_i l_k (r_i - h_k).$$

This is the same as the formula used to compute the area in V.16.2.

It is straightforward to generalize this formula so it can be used to compute the area of the intersection of more than two caps. We have use only for the case of three caps with "triangular" intersection. Let p_{kj}, $p_{\ell k}$, $p_{j\ell}$ be the corners of this "triangle", and let φ_{kj}, $\varphi_{\ell k}$, $\varphi_{j\ell}$ be the angles at these vertices. Again with a limiting process approximating the triangle by a spherical polygon we get the following formula for the area:

$$\frac{A_i}{2}(\varphi_{kj} + \varphi_{\ell k} + \varphi_{j\ell} - \frac{1}{2}) - 2\pi r_i [l_j(r_i - h_j) + l_k(r_i - h_k) + l_\ell(r_i - h_\ell)].$$

This is used in V.16.3.