Determination of Mass Properties of Polygonal CSG Objects in Parallel

Chandrasekhar Narayanaswami*
William Randolph Franklin†
Electrical, Computer, and Systems Engineering Dept.
Rensselaer Polytechnic Institute
Troy, New York 12180, USA

Abstract

A parallel algorithm for determining the mass properties of objects represented in the Constructive Solid Geometry (CSG) scheme that uses polygons as primitives is presented. The algorithm exploits the fact that integration of local information around the vertices of the evaluated polygon is sufficient for the determination of its mass properties, i.e., determination of the edges and the complete topology of the evaluated polygon is not necessary. This reduces interprocessor communication and makes it suitable for parallel processing.

The algorithm uses *data parallelism, spatial partitioning* and *parallel sorting* for parallelization. Tuple-sets on which simple operations have to be performed are identified. The elements of tuple-sets are distributed among the processors for parallelization. The *uniform grid* spatial partitioning technique is used generate sub-problems that can be done in parallel and to reduce the cardinality of some of the tuple-sets generated in the algorithm. Parallel sorting is used to sort the tuple-sets between the data-parallel phases of the algorithm.

1 Introduction

The Constructive Solid Geometry (CSG) and Boundary Representation (B-rep) schemes are used in conjunction to represent solid models because of their complementary functionalities. The CSG scheme is an intuitive way of *synthesizing* complicated solids, while the B-rep scheme is more suitable for *analyzing* a solid's mass properties and for generating images. Due to the complementary nature of the two schemes it is often necessary to convert between the two. This process is commonly known as representation conversion.

Lee and Requicha [9] have presented sequential algorithms, based on regular and recursive cellular subdivision, for approximate evaluation of integral properties of solids defined in the CSG scheme. Lien and Kajiya [10] give a symbolic method for calculating the integral properties of arbitrary nonconvex polyhedra.

*Now at: IBM Austin, 11400 Burnet Road, Austin, Texas 78758
†Email: wrf@ecse.rpi.edu, Phone: (518) 276-6077

Since speed is important in these interactive applications, a fast and practical parallel algorithm for this problem will be useful. We present new sequential and parallel algorithms for answering some of the commonly encountered queries, such as the determination of area and perimeter, on polygonal objects stored in the CSG representation scheme. An extension of these ideas to the polyhedral domain is also addressed. Implementations of related algorithms on the Sequent Balance 21000 parallel machine have shown close to linear speedup and the above algorithms are expected to show similar performance.

Since the mass property of a null CSG object is zero, our results can be used to perform Null-Object-Detection (NOD) without complete representation conversion. This result can be extended to Same-Object-Detection (SOD) as the two CSG objects can be subtracted and the mass property of the result can be checked for nullity.

2 Techniques

Our algorithms use the combination of the following techniques:

1. *Data parallelism.*
2. *Uniform grid.*
3. *Local topology.*

2.1 Data Parallelism

Most graphics and geometric applications that justify parallelization deal with a large number of objects on which many common operations are performed. It is natural to attempt parallelization of these algorithms by distributing the vertices, edges, faces, or complete objects among the processors. In order to use this *data partitioning* technique, the *attributes* of the *tuple-sets* on which the algorithm can operate have to be identified. For example, a set of (*cell, edge*) tuples form a tuple-set. Each element of the tuple-set is defined by a set of attributes. The attributes for the above example are *cell* and *edge*. The number of attributes needed to describe a tuple is its *arity*. The number of tuples in the tuple-set is its *cardinality* or size. For parallelization, the elements of the tuple-sets are distributed among the processors.

For complex applications, vertices, edges, and faces are not sufficient attributes for the tuple-sets, and more complex tuple-sets which are necessary to solve the problem have to be identified. The arity, the attributes themselves, and the cardinality of the tuple-set are dependent on the problem and the cost of inter-processor communication on the parallel machine. Problems with robustness and the presence of special cases due to coincident geometric entities can increase the arity of the tuple-set [14]. The use of tuple-sets and data-parallelism in our algorithm is demonstrated in later sections.

From a practical point of view, this technique has the following advantages:

1. ease of implementation,
2. ease of load-balancing among the processors, and
3. regular memory accesses and good use of hardware caching-mechanisms.

2.2 The Uniform Grid Technique (UGT)

The *uniform grid* [5] spatial subdivision scheme divides the extent of a geometric scene uniformly into many smaller subregions (cells) of identical shape and volume. The idea is to exploit the limited spatial extent of individual geometric entities by inserting them into the subdivision and performing subregion-wise computations on them, thereby minimizing global computations. This reduces the cardinality of the tuple-sets in the average case and plays an important role in increasing the efficiency of the algorithm. Note that the geometric entities are not split at the cell boundaries and thus this technique is different from many similar techniques.

The technique is also a *divide-and-conquer* mechanism because problems of the same type but of smaller size are generated for each sub-region. The uniform grid is also useful while searching for geometric entities in particular regions of a scene and hence is an indexing scheme for locating objects. For example, it improves the average-case query time for the point location problem. Applications and algorithms developed with this technique are summarized in [1].

According to Asano [2], the popularity of bucketing schemes similar to the uniform grid is because they often outperform theoretically better algorithms. For example, Pullar [16] has experimentally shown that the sequential version of the uniform grid technique for determining the intersections between a set of segments lying in the plane is faster than the theoretically better plane-sweep technique for a variety of data sets. The overhead of maintaining the complicated data structure for the plane-sweep technique is cited as the reason for its inefficiency. As argued below, in the parallel domain the plane-sweep technique is even less attractive when compared to the uniform grid.

2.2.1 Uniform Grid and Parallel Processing

It is the following properties of the UGT that make it a useful and practical technique for parallel processing:

1. Some optimal geometric techniques such as the plane-sweep technique, used for solving many geometric problems, impose a temporal ordering of computation on the objects, e.g., by scan line. The UGT does not impose one when none is strictly necessary. For example, computation of intersections within the grid cells can be done in any order. Therefore, the uniform grid technique is much simpler to parallelize.
2. The uniformity of the partitioning makes the reduction of the cardinality of tuple-sets simple and fast. It also makes the mapping of sub-tasks on to the processors easy and hardware implementations easier. When more sophisticated schemes such as quadtrees and octrees are used [11, 19], distribution of sub-tasks among the processors becomes more complicated. The overhead for maintaining the data structures would also be higher. However, such schemes may achieve a greater cardinality reduction.
3. Unlike hierarchical and adaptive spatial partitioning schemes, the UGT is a single-level spatial partitioning scheme which uses a flat data structure that exploits the large memories of modern machines and avoids tree data structures and indirect references through pointers to locate objects. This helps in avoiding log factors (due to tree traversals) in the complexity of the algorithm. In a parallel processing scenario, hierarchical schemes that use tree data structures can also introduce bus congestion if the leaf nodes have to be accessed through a common root.
4. The contiguous location of the tuples and the absence of pointers in the data structure for the UGT increase locality of memory references and exploit hardware caching-mechanisms. When objects in adjacent cells in the spatial partitioning interact, it is useful to transform adjacency in space to adjacency in memory to improve memory access patterns. The data structure for the UGT preserves such adjacency. This is important in the context of shared-memory parallel computers because improper use of caching-mechanisms causes severe degradation of performance [4].

2.2.2 Limitations of the Uniform Grid

Though the uniform grid has many advantages, it has some important limitations:

1. In worst-case situations, when all geometric entities are concentrated in a few grid cells, there is no appreciable reduction in the cardinality of the tuple-sets.
2. Uneven spatial densities of the objects reduce the efficiency of the uniform grid technique. However, experiments with the sequential segment intersection algorithm [15] have shown that this is not as serious a problem as it appears from a theoretical point of view because in such situations the added complexity of building an adaptive or finer sub-division for better object resolution competes with the benefit of such a sub-division. For better load-balancing in the context of parallel computation, contiguous cells from dense regions are allocated to different processors. In such situations, both load-balancing and locality of memory references for proper use of caching-mechanisms have to be considered.
3. In some applications where separate data structures are used for each grid cell, the memory requirements of the uniform grid technique may be greater than that of other schemes. However, with the decreasing cost of memory, this may not be a serious disadvantage.

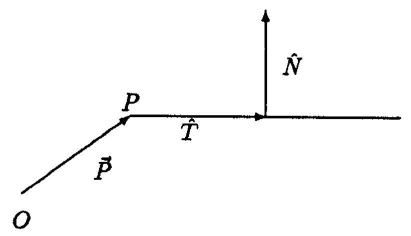


Figure 1: Local Topology Representation for a Vertex

2.3 Local Topology Scheme

The key idea of the *local topology* data structure for representing a polygon is that an integration of *local information* around its vertices is sufficient to determine its mass properties. A tuple of local information about a vertex includes its coordinates, the direction of an edge incident on it, and the direction of the inward normal to the edge. Formulae for mass properties and point classification for polygons and polyhedra have been given in [6]. In this paper we extend this scheme for the evaluation of mass properties of CSG objects.

In many CSG to B-Rep conversion algorithms, edges, faces, and other higher dimensional entities like shells are constructed in the later stages from lower dimensional entities. For example, while finding the union of two polygons, the tentative vertices of the resulting polygon can be determined first and the tentative edges can be determined next by splitting the edges of the primitives at these vertices. Since the local topology representation can be determined as soon as all the intersections among the primitives have been found, the later phases of the representation conversion process are not necessary when only mass properties are needed.

From the point of view of parallel processing, elimination of the unnecessary phases eliminates the corresponding inter-processor communication needed. Also, the data structure for the local topology representation scheme is simple and can be stored in an *unordered* fashion among the processors. The absence of any strict order in the data structure makes load balancing among the processors easier.

2.3.1 Theory

A polygon is represented as an unordered set of $(\vec{P}, \hat{T}, \hat{N})$ tuples. P is a vertex and $\vec{P} = (P_x, P_y)$ is the position vector of the vertex in a coordinate system of which O is the origin. $\hat{T}$ is a unit tangent vector from P along an edge incident on the vertex P and points to the other vertex of the edge. $\hat{N}$ is a unit vector normal to $\hat{T}$, in the plane of the polygon and points towards the face's interior. n denotes the number of tuples needed to describe the polygon.

Figure 1 shows an example of a vertex and its neighborhood. Figure 2 shows the representation of a polygon which has non-manifold conditions by the following set of *local topology* tuples.

$\{ (\vec{P}_1,\hat{T}_{12},\hat{N}_2), (\vec{P}_1,\hat{T}_{14},\hat{N}_1), (\vec{P}_2,\hat{T}_{21},\hat{N}_2), (\vec{P}_2,\hat{T}_{23},\hat{N}_3), (\vec{P}_3,\hat{T}_{32},\hat{N}_3), (\vec{P}_3,\hat{T}_{34},\hat{N}_4), (\vec{P}_4,\hat{T}_{41},\hat{N}_1), (\vec{P}_4,\hat{T}_{43},\hat{N}_4), (\vec{P}_4,\hat{T}_{45},\hat{N}_5), (\vec{P}_4,\hat{T}_{46},\hat{N}_7), (\vec{P}_5,\hat{T}_{54},\hat{N}_5), (\vec{P}_5,\hat{T}_{56},\hat{N}_6), (\vec{P}_6,\hat{T}_{64},\hat{N}_7), (\vec{P}_6,\hat{T}_{65},\hat{N}_6) \}$

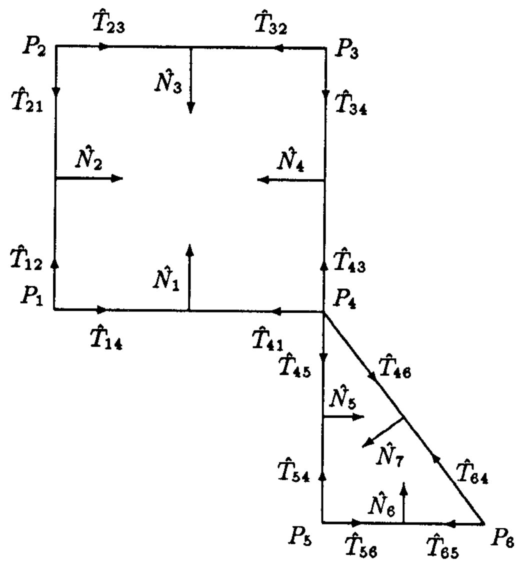


Figure 2: Local Topology Representation for a Polygon

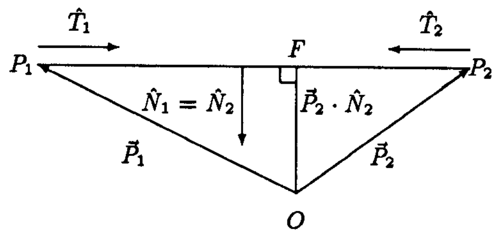


Figure 3: Proof for 2-D Mass Property Formulae

The following notation is used in the remainder of the paper. $\vec{T}_{ab}$ is a vector from P_a to P_b. $\vec{a} \cdot \vec{b}$ represents the dot product of the vectors $\vec{a}$ and $\vec{b}$. $\mid \vec{a} \mid$ represents the norm of the vector $\vec{a}$, and $\mid AB \mid$ the length of the line segment AB.

2.3.2 Mass Property Formulae

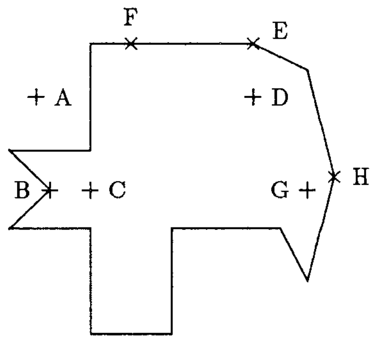
The formulae for determining some of the mass properties of the polygon are as follows.

$$Perimeter = -\sum_{i=1}^{i=n} \vec{P}_i \cdot \hat{T}_i \quad (1)$$

$$Area = \frac{1}{2}\sum_{i=1}^{i=n} \vec{P}_i \cdot \hat{T}_i \; \vec{P}_i \cdot \hat{N}_i \quad (2)$$

Consider the local topology tuples for the edge shown in Figure 3. Here, $\hat{T}_1 = \hat{P_1P_2}$ and $\hat{T}_2 = \hat{P_2P_1} = -\hat{T}_1$. As shown below, the contribution of these tuples to the formula for the perimeter is the length of the edge. Therefore the sum over all the tuples gives the perimeter of the polygon.

$$\begin{aligned} contribution &= -\sum_{i=1}^{i=2} \vec{P}_i \cdot \hat{T}_i \\ &= -\vec{P}_1 \cdot \hat{T}_1 - \vec{P}_2 \cdot \hat{T}_2 \end{aligned}$$



P	s_O
A	0
B	4
C	4
D	4
E	2
F	2
G	4
H	0

Figure 4: Examples Point Classifications

$$\begin{aligned} &= \mid P_1F \mid + \mid P_2F \mid \\ &= \mid \vec{P_1P_2} \mid \end{aligned}$$

For the area, the tuples for the edge shown in Figure 3 contribute the signed area of ΔOP_1P_2 because their

$$\begin{aligned} \text{contribution} &= -\frac{1}{2}\sum_{i=1}^{i=2} \vec{P_i}\cdot\hat{T_i}\ \vec{P_i}\cdot\hat{N_i} \\ &= \frac{1}{2}(\mid P_1F \mid \times \mid OF \mid + \mid P_2F \mid \times \mid OF \mid) \\ &= \frac{1}{2} \mid \vec{P_1P_2} \mid \times \mid OF \mid \end{aligned}$$

The sum over all local topology tuples is the sum of the signed areas of all such triangles. This sum gives the area of the polygon [10].

The following quantity has to be computed to classify the point O with respect to (w.r.t.) the polygon.

$$s_O = \sum_{i=1}^{i=n} \sigma(P_{i_x}T_{i_x})\sigma(\vec{P_i}\cdot\vec{N_i}) \qquad (3)$$

where σ is the sign function:

$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

$$\text{If point O is} = \begin{cases} \text{outside polygon} & \text{then } s_O = 0 \\ \text{inside polygon} & \text{then } s_O = 4 \\ \text{on boundary} & \text{then } s_O = 0, 2, or, 4 \end{cases}$$

The above result is a direct application of the Jordan Curve theorem in which the ray is a vertical one. Consider the tuples for the edge P_1P_2 in Figure 3. These tuples contribute a value of 2 if O is contained within the horizontal projection of the edge and is on the material side of the edge, a value of -2 if O is contained within the horizontal projection of the edge and is not on the material side of the edge, and a value of 0 if O is not contained within the horizontal projection of the edge. If O is vertically below an endpoint of the horizontal projection of the edge, the contribution is 1 instead of 2 and -1 instead of -2. The sum of these contributions is the quantity indicated in the formula.

Figure 4 shows some examples. The case where the origin is on the boundary of the polygon needs elaboration. If the origin is on an edge, then $s_O = 2$ (case F in Figure 4). If the origin is on a vertex, s_O can be 0, 2, or 4 (cases H, E and B) . The ambiguity between these cases and those of the points being inside or outside the polygon is resolved by noting that $\vec{P}$ will be zero for at least one $(\vec{P}, \hat{T}, \hat{N})$ tuple in the polygon if the origin is on a vertex.

Degenerate cases, such as when the origin is on an edge, can cause problems. To handle degeneracies, we recommend the Simulation of Simplicity technique of Edelsbrunner and Mucke [3]. Conceptually, this adds (ϵ, ϵ^2) to the origin so that it cannot be exactly on any vertex, edge, or extended edge. Epsilon is an infinitesimal, smaller than any positive real. For an explanation of infinitesimals, see Knuth [8]. We cannot actually add epsilon since it is not a representable floating point number, so we calculate the effect that it would have on any decisions.

Most of the formulae presented in this section can also be derived by using Green's theorem in the plane, which we recall here.

$$\oint_C Pdx + Qdy = \iint_{\mathcal{R}} \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) dxdy$$

P, Q, $\frac{\partial P}{\partial y}$, and $\frac{\partial Q}{\partial x}$ are single-valued continuous functions in $\mathcal{R}$ bounded by a closed curve C.

The evaluation of $\iint_{\mathcal{R}} F(x,y)dxdy$ can therefore be transformed into a contour integral after finding functions $P(x,y)$ and $Q(x,y)$ such that

$$\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} = F(x,y)$$

The contour integral can be evaluated piecewise over the edges of the polygon. Since $y = f(x)$ for each edge P_1P_2, the contour integral simplifies to $G(P_2) - G(P_1)$, where $G(P)$ can be computed with information in the local topology tuples for vertex P. In essence, the information needed to evaluate $G(P)$ must be captured in local topology representation for the vertex P.

The above result can be used to derive local topology based formulae for determining the center of gravity and the moments of the polygon. The point to note is that though these formulae are just a simple reorganization of standard formulae, it is useful in the parallelization of the evaluation of the mass properties of CSG trees and in eliminating the need for a complete evaluation of the boundary.

3 Mass Properties of CSG Trees

3.1 Input Specifications

The input to our algorithm includes the boundary representations of all the polygonal primitives of the CSG tree and any convenient representation of the CSG tree. The edges of the primitives of the CSG tree can be given in any *random* order, but they are oriented such that the interior of the primitive is to their right. The orientation of the edges is used in the point in polygon test described later. The primitives can themselves have multiple components and holes. Non-manifold conditions do not pose any representational problems in this scheme.

3.2 Outline of Algorithm

The outline for the algorithm is as follows:

1. Use the uniform grid technique to determine all the intersections between the edges of the primitives of the CSG tree.
2. Determine the local topology representation of the evaluated object.
3. Use the local topology representation to determine the mass properties of the CSG object.

3.3 The Detailed Algorithm

The algorithm resulting from the strategy outlined above is described in greater detail in this section.

3.3.1 Parallel Computation Intersection Points

The algorithm used for computing the points of intersection in parallel is described below.

```
Procedure: Get_pts_of_intersection (Polygons in Tree)
begin
  Partition the scene into $G \times G$ uniform square cells;

  For each polygon in CSG tree in parallel do
    For each edge in polygon do
      Determine (cell, edge, polygon) tuples;
    end for
    Add (cell, edge, polygon) tuples to global list of tuples;
  end parallel for

  /* sort by cell */
  Parallel_bucket_sort ((cell, edge, polygon) tuples);

  For each cell in grid in parallel do
    For all pairs of tuples $(e_i, p_k)$, $(e_j, p_l)$ in cell do
      If $k \neq l$ and $e_i$ and $e_j$ intersect then
      begin
        Get point of intersection, $P$, of $e_i$ and $e_j$;
        If $P$ is inside cell then
        begin
          If $P$ is a vertex of $e_i$ or $e_j$ then
          t_x ← end point intersection;
          else t_x ← mid point intersection;
          If $P$ is close to cell boundary then
          on_g ← true;
          else on_g ← false;
          Add tuple $(e_i, p_k, e_j, p_l, P_x, P_y,$ t_x, on_g);
        end if
      end if
    end for
  end parallel for
end
```

Parallelization is achieved by distributing the polygons and the cells among the processors. For the task of determining the *(cell, edge, polygon)* tuples, good load balancing can be achieved by distributing the polygons among the processors in a random fashion. To achieve better load balancing for the parallel computation of the points of intersection in cases where the density of edges varies considerably over the cells and when certain regions contain a lot of edges, cells that are spatially adjacent are given to different processors.

Table 1: Combining Classifications for $A \bigcup B$

A \ B	*in*	*out*	*on*
in	in	in	in
out	in	out	on
on	in	on	?

The fineness G, of the uniform grid is a function of the length of the edges in the primitives of the CSG tree. Usually $G = cL^{-1}$ is a good heuristic for minimizing the computation time, where L is the average length of the edges of all the primitives and c is a tuning constant.

The cells which an edge passes through are determined by using a variant of Bresenham's raster line drawing algorithm. For this purpose, the lower and left boundaries of a cell are included in the cell. For improving numerical robustness, if an edge passes through a grid corner it is entered in all four cells adjacent to the corner.

In order to handle cases of collinearity and coincidence in the input, overlapping collinear edges and edges which touch each other are considered to intersect.

In order to avoid multiple reporting of the same intersection between a pair of edges going through many adjacent cells, the intersection is reported only if the point of intersection lies inside the cell in which the pair is tested. Since a point-in-cell test can be devised such that the point belongs to only one cell, when the same pair of edges is checked for intersection in many cells, only one of them will report the intersection.

Suppose two edges intersect at a grid corner. For robust intersection detection, they are tested for intersection in all four grids cells incident at the corner. Otherwise, there is a possibility of missing the intersection because the computed point of intersection may lie in one of the other three cells, due to limited numerical accuracies.

Cases where numerical problems could lead to incorrect results are identified in the intersection phase and appropriate information is stored in the intersection tuples for use in the determination of the local topology representation.

3.3.2 Determination of Local Topology

At this point, the locations of the tentative vertices in the evaluated polygon have been determined. To determine the local topology representation, it is necessary to determine the vertices that lie on the evaluated object and the corresponding edges incident on them. To determine whether a vertex lies on the evaluated object, the classification of the vertex w.r.t. the CSG tree has to be obtained. This requires the classification of the vertex w.r.t. all the primitives in the CSG tree. These classifications then have to be combined at the internal nodes of the CSG tree. As shown in Table 1, one of the problems with combining the classifications is that combining ON-ON classifications needs explicit neighborhood information. Moreover, determination of the vertices of the evaluated object alone is not sufficient to determine its local topology representation.

The following simple observations (also see Figures 5 and 6) can be used to solve the above problems.

Observation 1 *A vertex can lie ON the boundary of only primitives which intersected to form it. Its classification*

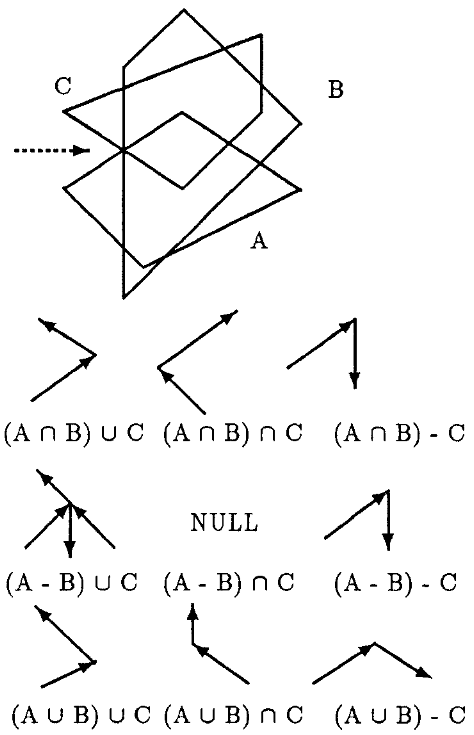
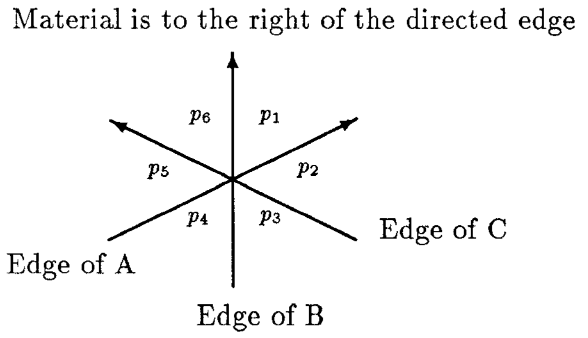


Figure 5: Some Possible Wedges Around a Vertex



Point	A	B	C
p1	OUT	IN	IN
p2	IN	IN	IN
p3	IN	IN	OUT
p4	IN	OUT	OUT
p5	OUT	OUT	OUT
p5	OUT	OUT	IN

Figure 6: Point Classification for Wedge Points

w.r.t. all other primitives is either IN or OUT.

Observation 2 *Points which are infinitesimally close to the vertex and lie in the wedges around the vertex (wedge points) cannot lie ON any primitive in the CSG tree.*

Observation 3 *The classification of wedge points which lie in consecutive wedges around the vertex changes only w.r.t. the primitive(s) to which the common edge(s) between the consecutive wedges belong.*

Observation 4 *A vertex will be ON the evaluated object if the classification w.r.t. the CSG tree of the points in the wedges around the vertex are not all identical.*

Observation 5 *If the classifications of points w.r.t. the CSG tree in the radially adjacent wedges around a vertex are different, the edge that is common to the two wedges lies on the boundary of the evaluated object.*

The use of the above observations removes the ON-ON ambiguity (see Table 1) and also allows the determination of the local topology representation of the evaluated polygon. Figure 5 shows an example where primitives A, B, and C intersect. The wedges at this intersection in the final object corresponding to different Boolean operations on these primitives are also shown. Figure 6 shows the wedge points for the intersection in Figure 5. The classifications of the wedge points w.r.t. the primitives A, B, and C are also shown. Note that no wedge point lies on any primitive.

Before the following procedure to determine the local topology representation is invoked, tuples corresponding to the vertices of the primitives are also added to the appropriates cell's list of $(e_i, p_k, e_j, p_l, x, y, t\_x, on\_g)$ tuples. The computations in the procedure are done in parallel on a cell-wise basis after the following adjustment. Consider scenarios in which more than two edges intersect at a grid line or corner (detected by checking the on_g variable for the intersection). In these cases, due to limited numerical accuracies, all the intersections at these vertices may not be reported in the same cell. This implies that tuples corresponding to the same intersection may not be handled together since computations are done independently in the cells. This can cause incorrect wedges in the result and therefore tuples whose on_g variable is *true* are gathered together and the core of the procedure is invoked on them separately.

```
Procedure: Get_local_topology (Intersection tuples)
begin
  For each cell in grid in parallel do
  begin
    Sort $(e_i, p_k, e_j, p_l, x, y, t\_x, false)$ tuples by the $x$ and $y$;
    For each set of tuples whose intersection points $(x, y)$
    are closer than $\epsilon$ do
      Let $V$ be the vertex at location $(x, y)$;
      Sort $(e_i, p_k, e_j, p_l, t\_x, false)$ tuples radially around $V$;
      Classify $V$ with all primitives not intersecting at $V$;
      Create wedges around $V$;
      For each wedge around $V$ do
        Classify wedge pt. w.r.t. primitives intersecting at
        $V$;
        Deduce wedge pt. classification w.r.t. primitives not
        intersecting at $V$;
        Evaluate wedge pt. classification with tree;
      end for
      For each wedge around $V$ do
```

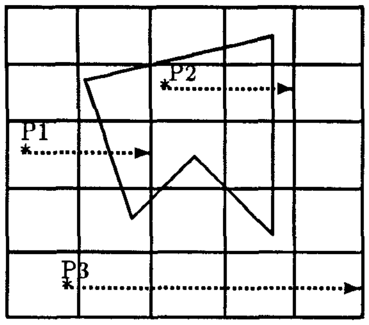


Figure 7: Point Classification with Polygons

```
        If classification of wedge point and classification of
        next wedge point are different then
        begin
          Get direction of edge between wedges;
          Add local topology tuple for $V$;
        end if
      end for
    end for
  end parallel for
end
```

Vertices that are separated by a distance less than ϵ are treated as identical vertices. ϵ is a small constant that is determined experimentally. As shown in Figure 6, the classification of the wedge points w.r.t. the primitives intersecting at the vertex is determined from the circular ordering of the edges around the vertex. From observation 3 it follows that the wedge points need not be classified individually with a point in polygon test w.r.t. all the primitives that did not intersect at the vertex; instead classification of the vertex is sufficient.

The classification of wedge points w.r.t. the CSG tree is done by combining the classifications recursively as shown in Table 1. Figure 7 shows the point w.r.t. primitive classification procedure. The pseudo-code for the test is shown below.

```
Procedure: Classify_pt_with_polygons ($P$, Polygons)
begin

  Determine cells intersected by ray from $P$ in $+x$ direction;
  For each cell in ray's path do
    For each edge in cell do
      If ray passes through vertex of edge then
        Perturb ray in $-y$ direction;
      If ray intersects interior of edge then
      begin
        Determine x coordinate of point of intersection $I_x$;
        Add (polygon, edge, $I_x$) to list of such tuples;
      end if
    end for
  end for

  /* major key polygon, minor key $I_x$ */
  Sort (polygon, edge, $I_x$) tuples;

  For each polygon in (polygon, edge, $I_x$) tuples do
    Get first (polygon, edge, $I_x$) tuple for polygon;
    If $P$ is to left of the oriented edge then
      Mark $P$ as outside the polygon;
    else Mark $P$ as inside polygon;
  end
  For each polygon not in (polygon, edge, $I_x$) tuples do
    Mark $P$ as outside polygon;
  end for
end
```

The ray will intersect only a few cells and in the average case the uniform grid ensures that it intersects only a small fraction of the primitives. If the ray intersects a primitive, it can be classified with the orientation of the first edge of the primitive to intersect it. If the ray does not intersect an edge of a primitive, the point is outside that primitive. So a *single* ray is used to classify the point with all the primitives.

3.3.3 Parallel Evaluation of Formulae

Once the local topology representation of the polygon is available, the evaluation of the mass properties using the formulae in Section 2.3.2 can be easily parallelized because they involve a sum of simple functions. In a tightly coupled multiprocessor (sharing common memory through a bus) each processor can compute some of the necessary calculations which can then be summed up using the *parallel prefix* algorithm. In a Single-Instruction-Multiple-Data (SIMD) machine with many *node* processors, the local topology tuple can be stored in the node processors which compute the desired information w.r.t. that vertex. The *host* processor then gathers the results of the computations performed by the nodes by using the broadcast graph used to dispatch the data to the node processors. The regular communication pattern between the nodes and the host, and the absence of any inter-node communication are some of the advantages of this scheme.

3.4 Accuracy

Some methods to increase the robustness of computation were presented in the algorithm. The criterion for determining special cases, such as an intersection being close to a cell boundary is less strict than for checking whether two intersections are coincident, i.e., a larger ϵ is used to check closeness to a cell boundary than the one used to group intersections together. This is done to be on the conservative side so that incorrect topologies in the output are avoided, since incorrect topologies can reduce the accuracy of the results drastically.

The error introduced by grouping vertices closer than ϵ is now analyzed. Let $\vec{P}_i' = \vec{P}_i + d\vec{P}$, be the local topology tuples due to errors in computation. Note that the $\vec{N}$ and $\vec{T}$ are not affected by this process as these vectors are derived from the input primitives. $| d\vec{P} | < 2\epsilon$, where ϵ is the tolerance to check whether two vertices are coincident. The effect on perimeter is $dPerimeter = \sum_{i=1}^{i=n} d\vec{P}_i \cdot \vec{T}_i$. Thus the error in the perimeter is linear in the error introduced in the computation. Similarly, the error in the area is also linear.

3.5 Algorithm Complexity and Implementation

Let p be the number of processors used, N_{in} the number of non-leaf nodes in the CSG tree, N_p the number of primitives or leaf nodes in the CSG tree, n the total number of edges in all the primitives, and k the number of intersections between the edges of the primitives. Usually in the CSG domain

$n = \theta(N_p)$. Also when every interior node has only two children, $N_p = N_{in} + 1$.

We derive the worst-case sequential complexity first in the absence of the uniform grid. The intersection process can take $O(n^2)$ time as k can be n^2. Sorting the k intersections takes $O(k \log k) = O(n^2 \log n)$ time. The $\theta(k)$ point classifications against all the primitives takes $O(N_p)$ time. Combining the $\theta(k)$ classifications at the interior nodes of the CSG tree takes $\theta(N_{in})$ time as each combination can be done in constant time. The complexities of the local topology representation and the related formulae are linear in the size of the result and is thus $O(n^2)$. Using the above relations, the worst-case sequential complexity of the algorithm is thus $O(n^3)$.

Experience with practical implementations of geometric algorithms shows that worst case complexity results provide litte information on the expected performance. In the average case, the uniform grid will determine the intersections in $O(n)$ time [15] and will also ensure that only a constant number of edges of the primitives intersect the ray for the point classification test. Thus a more reasonable estimate of the expected complexity of the complete algorithm is $O(n^2)$.

The practicality of the techniques for parallelization discussed in Section 2 has been shown in [14]. An implementation of the polygon intersection algorithm on the Sequent Balance 21000 shows linear speedup [14]. Thus the intersection phase of the mass property determination algorithm will also show linear speedup. The classification of wedge points and the evaluation of the formulae are also expected to show linear speedup because good load balancing can be achieved for these phases.

4 Extension to 3-D Polyhedra

4.1 Formulae

In this section we show how the ideas presented so far can be extended to the polyhedral domain. P is a vertex and $\vec{P} = (P_x, P_y, P_z)$ is the position vector of the vertex in a coordinate system of which O is the origin. $\hat{T}$ is a unit tangent vector from P along an edge. $\hat{N}$ is a unit vector normal to $\hat{T}$, in the plane of a face adjacent to both P and the edge, and pointing towards the face's interior. $\hat{B}$ is a unit vector normal to both $\hat{T}$ and $\hat{N}$, which makes it normal to the face plane, and pointing towards the interior of the polyhedron. n is the total number of tuples needed to represent the polyhedron. Figure 8 shows the representation of a polyhedron vertex.

The polyhedron is represented by a set of $\{(\vec{P}, \hat{T}, \hat{N}, \hat{B})\}$ tuples, where there is one element for each occurrence of an adjacency of a vertex, edge, and face. For the cube in Figure 9 there are six elements per vertex, or 48 in all. The six $(\vec{P}, \hat{T}, \hat{N}, \hat{B})$ tuples for vertex $\vec{A}(0,0,0)$ are
((0,0,0), (1,0,0), (0,1,0), (0,0,1)), ((0,0,0), (1,0,0), (0,0,1), (0,1,0)),
((0,0,0), (0,1,0), (1,0,0), (0,0,1)), ((0,0,0), (0,1,0), (0,0,1), (0,1,0)),
((0,0,0), (0,0,1), (1,0,0), (0,1,0)), ((0,0,0), (0,0,1), (0,1,0), (1,0,0)).
For example, the first $(\vec{P}, \hat{T}, \hat{N}, \hat{B})$ tuple is $(\vec{A}, \hat{AB}$, a normal to $\hat{AB}$ in the plane ABE pointing into the face, a normal to the plane ABE pointing into the cube).

Some of the mass property formulae for a polyhedron are

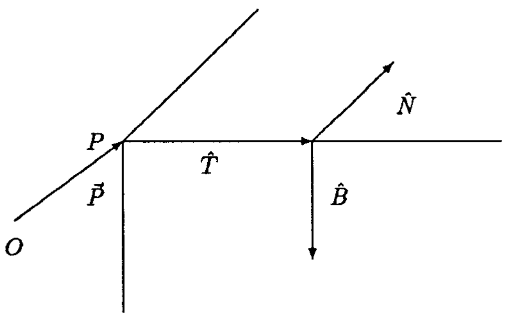


Figure 8: Representation for a Polyhedron Vertex

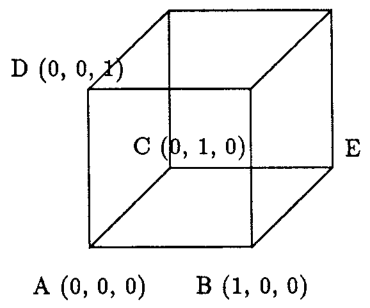


Figure 9: Local Topology Representation for a Cube

as follows.

$$\sum_{\text{all faces}} \textit{Perimeter} = -\sum_{i=1}^{i=n} \vec{P}_i \cdot \hat{T}_i \tag{4}$$

$$\textit{Area} = \frac{1}{2}\sum_{i=1}^{i=n} \vec{P}_i \cdot \hat{T}_i \; \vec{P}_i \cdot \hat{N}_i \tag{5}$$

$$\textit{Volume} = -\frac{1}{6}\sum_{i=1}^{i=n} \vec{P}_i \cdot \hat{T}_i \; \vec{P}_i \cdot \hat{N}_i \; \vec{P}_i \cdot \hat{B}_i \tag{6}$$

The proofs for the formulae for length and area are similar to the ones for polygons. The proof for the volume makes use of the fact that it can be evaluated by taking a central projection and adding appropriate contributions of the cones defined by the faces of the object with respect to the center of projection O. Consider the $(\vec{P}, \hat{T}, \hat{N}, \hat{B})$ tuples corresponding to a face of the polyhedron. For all these tuples $\vec{P} \cdot \hat{B}$ is the same and $\mid \vec{P} \cdot \hat{B} \mid$ is equal to the height of the cone. Hence

$$\begin{aligned} \text{contribution} &= -\frac{1}{6}\sum_{\text{face}} \vec{P} \cdot \hat{T} \; \vec{P} \cdot \hat{N} \; \vec{P} \cdot \hat{B} \\ &= \frac{1}{3}\text{height of cone}\frac{1}{2}\sum_{\text{face}} \vec{P} \cdot \hat{T} \; \vec{P} \cdot \hat{N} \\ &= \text{volume of cone} \end{aligned}$$

4.2 Algorithm

We now briefly present an algorithm for evaluating mass properties of polyhedral CSG objects. The first task is to intersect the faces of the polyhedra and to determine the lines of intersection (cut-lines) between the faces. The intersection determination phase of our 3-D uniform grid based parallel algorithm for Boolean operations on polyhedra [14] can be used for this purpose.

As in the 2-D case, once the cut-lines have been determined, the local topology representation for the evaluated CSG tree is created. For this purpose, the vertices, edges incident at these vertices, and the faces around the edges have to be determined. The set of endpoints of the cut-lines and vertices in the primitives of the CSG tree is the tentative set of vertices in the evaluated object. The set of cut-lines and unintersected edges of the primitives is the set of tentative edges in the evaluated object. The cut-lines are sorted so that all faces intersecting at a particular cut-line are available together for consideration. All the faces intersecting at a tentative edge are sorted radially around the edge. To determine the normals of the faces around the edges of the resulting object, points in the dihedral wedges around the tentative edges in the evaluated object are classified w.r.t. the CSG tree. To determine which faces to include, observations similar to the ones for 2-D apply. Point classification with primitives is done by a 3-D version of the 2-D algorithm. The local topology representation can now be created and the mass properties evaluated. Thus, evaluation of the complete topology of the evaluated polyhedron is avoided for mass property calculation. A face in the result need not even know all the edges on it.

Parallelization is achieved by distributing the faces, cut-lines, cells, etc. among the processors. An implementation of our parallel algorithm on the Sequent Balance 21000 for determining Boolean combinations of polyhedra shows close to linear speedup. Therefore the first phase of the mass property algorithm will show good speedup. For the speedup of the remaining phases of the algorithm, arguments similar to those for the 2-D analysis given in Section 3.5 apply.

5 Discussion

5.1 Acceleration Techniques

In an implementation many optimizations are possible. For example, at a *union* node it is not necessary to classify the point w.r.t. the right sub-tree of the node if the point lies inside the object represented by the left sub-tree. Similar optimizations can be done at other types of nodes.

Furthermore, it may not be necessary to evaluate the complete tree for each of the wedge points around a vertex. The classification of all the wedge points around a vertex w.r.t. a subtree will be identical if the leaves of the subtree do not include the primitives intersecting at the vertex. Consider the vertex shown in Figure 5 where the primitives A, B, and C intersect. Figure 10 shows subtrees (enclosed in the boxes) whose leaves do not include the primitives A, B, and C. In an implementation, this can be done by storing the list of a node's children at the node.

It remains to be investigated whether our ideas can be combined with the active-zone technique proposed by Rossignac and Voelcker [18] for accelerating CSG computations.

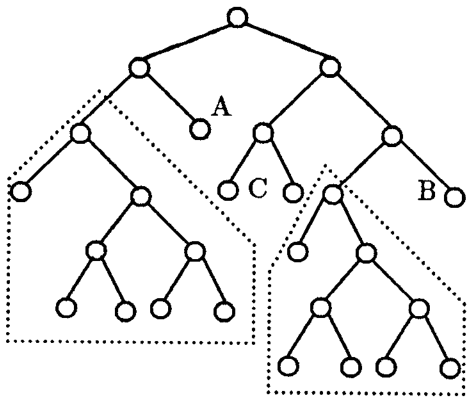


Figure 10: Optimization of CSG Point Classification

At a more theoretical level, as indicated in [13], the techniques for parallel tree contraction (evaluating an arithmetic expression tree) given by Miller and Reif in [12] can be used for the parallel evaluation of a CSG tree whose primitives store Boolean values of 0 or 1. Goodrich has also independently explored the application of the above ideas to boundary evaluation of CSG trees in [7]. This approach would be practical when a large number of processors is available.

5.2 Analysis of Our Approach

Lee and Requicha [9] combined the use of spatial subdivision schemes and the Monte Carlo technique for determining integral properties of CSG objects approximately. The contributions from the individual cells that were inside the CSG object were summed to yield the integral properties. In contrast, the formulae for mass properties that were presented in this paper are symbolic and hence the accuracy of the result is dependent on the arithmetic precision available on the computer. Greater accuracy is available in their approach at the expense of both computer time and space. Their technique is likely to become progressively more susceptible to numerical errors in point classification as the resolution of the cellular approximation is increased to obtain greater accuracy. The complexities of their algorithms depend on the square and cube of the resolution. In contrast, the complexity of our algorithm depends on the number of intersections between the primitives. For coarse resolutions, their algorithm may be faster if parallelized, but when accuracy requirements are increased, our algorithm is likely to perform better. Our algorithm can be used to evaluate surface properties of the CSG object such as the surface area, which would be difficult in their approach. However, their techniques are easier to modify for handling curved polyhedra.

As with some of the algorithms in [9], an advantage of our method over traditional complete boundary evaluation schemes is that it does not need an edge w.r.t. polygon classifier with explicit neighborhood information and can manage with a simpler point w.r.t. polygon or polyhedron classifier without explicit neighborhood information.

The algorithm presented can be used to handle dynamic insertion of objects into the CSG tree. The points of intersections of the new object with the other primitives, and the subtrees which get modified as a result of insertion are the only factors to affect the mass property of the new evaluated object. The uniform grid and the local topology scheme are well suited for this computation.

The advantage of flat non-incremental evaluation [17] of the CSG tree and the consequential parallelism derived from it has to be compared with the additional work performed in computing *unnecessary* intersections which an incremental evaluation algorithm would avoid. In addition to applicability of the local topology technique, an advantage of non-incremental evaluation is that if the interior nodes of the CSG tree are modified, the mass properties of the new tree can be evaluated by simply re-evaluating the classifications of the wedge points.

6 Conclusion

We demonstrated that the uniform grid and local topology techniques can be combined to determine the mass properties of objects stored in the CSG representation scheme with polygonal primitives efficiently in parallel. An extension to the polyhedral problem was also discussed. The technique can be extended to handle curved polygons. Whether and how extensions to handle curved polyhedra can be made is yet to be investigated.

Our objective was to demonstrate that a synergy of previously known techniques is useful to design parallel geometric algorithms. We hope that this paper will motivate researchers to think of new object representation schemes that are suitable for parallel processing.

References

[1] V. Akman, W. R. Franklin, M. Kankanhalli, and C. Narayanaswami. Geometric Computing and the Uniform Grid Data Structure. *Computer Aided Design*, 21(7):410–420, September 1989.

[2] T. Asano, M. Edahiro, H. Imai, M. Iri, and K. Murota. *Practical Use of Bucketing Techniques in Computational Geometry*, volume 2 of *Machine Intelligence and Pattern Recognition*, pages 153–195. Elsevier Science Publishers, 1985.

[3] H. Edelsbrunner and E. P. Mucke. Simulation of Simplicity : A Technique to Cope Degenerate Cases in Geometric Algorithms. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 118–133, Champaign-Urbana, Illinois, June 1988.

[4] S. J. Eggers and R. H. Katz. The Effect of Sharing on the Cache and Bus Performance of Parallel Programs. In *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 257–270, Boston, April 1989.

[5] W. R. Franklin. *Combinatorics of Hidden Surface Algorithms.* PhD thesis, Center for Research in Computing Technology, Harvard University, June 1978.

[6] W. R. Franklin. Vertex Based Polyhedron Formulae. (submitted for publication), November 1988.

[7] M. T. Goodrich. Applying Parallel Processing Techniques to Classification Problems in Constructive Solid Geometry. In *Proceedings of the First ACM-SIAM Symposium on Discrete Algorithms*, pages 118–128, San Francisco, January 1990.

[8] D. E. Knuth. *Surreal Numbers: How Two Ex-students Turned on to Pure Mathematics and Found Total Happiness: A Mathematical Novelette.* Addison-Wesley, 1974.

[9] Y. T. Lee and A. A. G. Requicha. Algorithms for Computing the Volume and Other Integral Properties of Solids. II. A Family of Algorithms Based on Representation Conversion and Cellular Approximation. *Communications of the ACM*, 25(9):642–650, September 1982.

[10] S. L. Lien and J. T. Kajiya. A Symbolic Method for Calculating the Integral Properties of Arbitrary Nonconvex Polyhedra. *IEEE Computer Graphics and Applications*, pages 43–51, October 1984.

[11] D. J. Meagher. Geometric Modelling Using Octree Encoding. *Computer Graphics and Image Processing*, 19:129–147, June 1982.

[12] G. L. Miller and J. H. Reif. Parallel Tree Contraction and its Application. In *Proceedings of the 26th IEEE Symposium of Foundations of Computer Science*, pages 478–489, 1985.

[13] C. Narayanaswami. Parallel Processing for Geometric Applications. Doctoral Thesis Proposal, Rensselaer Polytechnic Institute, July 1989.

[14] C. Narayanaswami. *Parallel Processing for Geometric Applications.* PhD thesis, Rensselaer Polytechnic Institute, Troy, New York 12180, December 1990.

[15] C. Narayanaswami and M. Seshan. The Efficiency of the Uniform Grid for Computing Intersections. Master's thesis, Electrical, Computer, and Systems Engineering Dept., Rensselaer Polytechnic Institute, Troy, NY, December 1987.

[16] D. Pullar. Comparative Study of Algorithms for Reporting Geometrical Intersections. In *Proceedings of the International Symposium on Spatial Data Handling*, pages 66–75, Zurich, July 1990.

[17] A. A. G. Requicha and H. B. Voelcker. Boolean Operation in Solid Modeling: Boundary Evaluation and Merging Algorithms. In *Proceedings of the IEEE*, volume 73, pages 30–44, January 1985.

[18] J. R. Rossignac and H. B. Voelcker. Active Zones in CSG for Accelerating Boundary Evaluation, Redundancy Elimination, Interference Detection, and Shading Algorithms. *ACM Transactions on Graphics*, 8(1):51–87, January 1989.

[19] H. Samet and R. E. Webber. Hierarchical Data Structures and Algorithms for Computer Graphics. *IEEE Computer Graphics and Applications*, 8:48–68, 1988.