99A000418

Research Report

A Computing Strategy for Applications Involving Offsets, Sweeps, and Minkowski Operations

E.E. Hartquist
Sibley School of Mechanical and Aero Engineering
Cornell University
Ithaca, NY 14853

J.P. Menon
IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598

K. Suresh
Research and Development Engineering
Kulicke and Soffa Industries
Willow Grove, PA 19090

H.B. Voelcker
Sibley School of Mechanical and Aero Engineering
Cornell University
Ithaca, NY 14853

J. Zagajac
Product and Development
Ford Motor Company
Dearborn, MI 48126

Research Division
Almaden - Austin - Beijing - Haifa - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g. payment of royalties).
Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home email reports@us.ibm.com
Copies may be requested from IBM T. J. Watson Research Center [Publications 16-220 ykt] P. O. Box 218, Yorktown Heights, NY 10598 USA

A computing strategy for applications involving offsets, sweeps, and Minkowski operations

E.E. Hartquist[a], J.P. Menon[b], K. Suresh[c], H.B. Voelcker[a,*], J. Zagajac[d]

[a]*The Sibley School of Mechanical and Aero Engineering, Cornell University, Ithaca, NY, USA*
[b]*T.J. Watson Research Center, IBM Corporation, Yorktown Heights, NY, USA*
[c]*Research and Development Engineering, Kulicke and Soffa Industries, Willow Grove, PA, USA*
[d]*Product Development Systems, Ford Motor Company, Dearborn, MI, USA*

Received 29 June 1998; accepted 24 September 1998

Abstract

Offsets, sweeps, and Minkowski operations (M-ops) are easy to define in the existential (representation-free) mathematics of point sets, but computing 'values' for offset, swept, and M-summed entities is thought to be difficult by many. This article argues that such computations may be easy if (1) they are cast in specific application contexts, and (2) relevant mathematical definitions are discretized and implemented directly. The argument is based on 10 years of research on a range of motional, process-modeling, and visualization problems that involved offsetting, sweeping, and M-ops; the solution paradigm common to all was direct approximation of mathematical definitions, using ray representations and parallel computation as primary media. This article presents no new results; it merely summarizes a body of well documented research that illustrates an approach to problem solving, whose primary tenets are: compute only what you need to solve the problem at hand, and do that as directly as possible. © 1999 Elsevier Science Ltd. All rights reserved.

Keywords: Offsets; Sweeps; Minkowski sums; Ray representations

1. Introduction

Offsets, sweeps, and Minkowski operations (M-ops) are easy to define in the existential (representation-free) mathematics of point sets. For example:

$$\text{Sweep}: \quad \text{Sweep}(A \text{ on } B) = \bigcup_{b \in B} A@M(b), \tag{1}$$

where A is usually a three-dimensional (3D) 'solid' set (an r-set), B a set of dimension ≤ 6, and $M(b)$ a rigid motion applied to A.

$$\text{M-sum}: \quad A \oplus B = \{\mathbf{a} + \mathbf{b}|_{a \in A, b \in B}\}, \tag{2a}$$

$$\text{M-sum}: \quad A \oplus B = \text{Sweep}(A \text{ on } B), \tag{2b}$$

$$\text{M-sum}: \quad A \oplus B = \text{Sweep}(B \text{ on } A), \tag{2c}$$

where A and B typically are 3D 'solid' sets and $\mathbf{a} + \mathbf{b}$ denotes addition of vectors representing points a, b.

Some useful simplifications are also easy to establish. For example, when B is a singly connected set containing the origin $\mathbf{o}$, Eqs. (2a–c) can be rewritten as follows.

$$\text{M-sum}: \quad A \oplus B = A \cup (B \oplus \partial A), \tag{3}$$

where ∂A denotes the boundary of A. See [4] for refined versions of Eq. (3) that relax the restrictions on B.

While defining offsets, sweeps, and M-ops is easy, computing useful *representations* of offset, swept, and Minkowski-summed (or -differenced) entities is widely viewed as difficult and expensive – and indeed this can be so if one insists on representing explicitly the exact boundary geometry of such entities. Fortunately, explicit boundary geometry is not needed in a surprising variety of engineering applications. Ten years of research on a range of motional, process-modeling, and visualization problems that involved offsetting, sweeping, and M-ops convinced us that an application's intrinsic mathematical requirements should dominate the design of an implementation; when these are specified, representations and algorithms can be designed to satisfy them. We illustrate this approach later by reviewing an examplary application – NC-program verification.

This short article offers no new results. It simply seeks to demonstrate that one can arrive at effective

* Corresponding author. Tel.: +1-607-255-9654; fax: +1-617-255-1222.
E-mail address: hbv1@cornell.edu (H.B. Voelcker)

Table 1
Mathematical operations intrinsic to NC verification

Application requirement	Mathematical operation(s)	Operand(s)	Result
Collision detection	(1) Regularized intersection	(1) Swept solid, solid	(1) Solid
	(2) Null-object detection	(2) Solid	(2) Boolean
Invasive machining detection	(1) Regularized intersection	(1) Swept solid, solid	(1) Solid
	(2) Null-object detection	(2) Solid	(2) Boolean
Material removal modeling	Regularized difference	Solid, swept solid	Solid
Feature-localized tolerancing	Dilation, contraction	Solid	Solid
Machining dynamics checking	d/dt Volume (solid)	Solid	Real number

Table 2
Mathematical operations associated with discretionary graphics

Graphic requirement	Mathematical operation(s)	Operand(s)	Result
Rendering	Surface gradient	Local boundary	Vector
Visibility	Sorting	Points (ray:bdry intercepts)	Visible bdry segment

implementations of applications involving offsets, sweeps, and M-ops by

- focusing on the computational requirements intrinsic to the application,
- using direct approximations of relevant mathematical definitions, and
- providing copious computing resources, so that implementations can be designed for simplicity, clarity, and robustness, rather than simply for resource conservation (i.e. classical efficiency).

2. Modeling and representation in NC verification

NC verification (our exemplary application) connotes an automated procedure that determines whether an NC program will produce a Finished Part from specified stock without damage to the machining environment or cutters. A verification procedure is summarized here in simplified form [5,6]. In essence, the procedure simulates the machining process specified by the program, checks each machining operation for undesirable conditions, and performs a final goal attainment test.[1]

Verify (NC_Program, Stock, Finished_Part, Fixtures, ... ⟨more⟩)
  Workpiece ← Stock
  For each cutter-motion instruction in NC_program *do*
    Decode the instruction
    {represent the spatial region traversed by the cutter on the current trajectory}
    V ← Sweep (Cutter on Trajectory)
      {check viability}
      if $V \cap_3$ Fixtures $\neq \phi$ *then* Collision_Exit
      if ($V \cap_3$ Contraction (Finished_Part) $\neq \phi$) *then* Invasive_Machining_Exit
      if (Rate ($V \cap_3$ Stock) > Limit) *then* Dynamic_Overload_Exit
        {...and more}
        {update the workpiece representation by 'subtracting' the cutter-swept region}
      Workpiece ← Workpiece $-_3$ V
    end_do
  if (Workpiece $\not\subset$ Dilation (Finished_Part)) *then* Out_of_Tolerance_Exit
end_Verify.

The mathematical operations intrinsic to NC verification, as defined by the simplified procedure given before, are summarized in Table 1.

Graphic displays provide helpful diagnostics, but are not intrinsic to automatic NC-program verification. Table 2 caters for discretionary graphics.

The critical implementation decisions reside in selecting representations for the Operands and Results that enable the mathematical operations to be implemented with efficacious (simple, clear, robust, ... acceptably efficient)

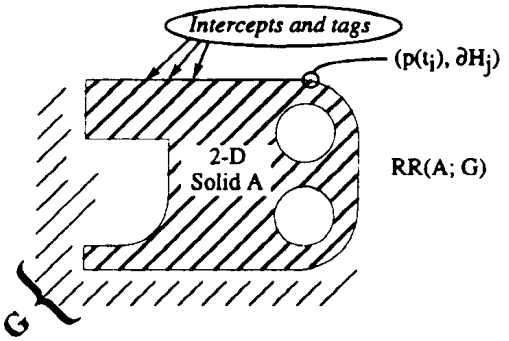

Fig. 1. A ray-rep of a 2D solid 'A'.

[1] Subscripts on operators e.g. $\cap_n$, denote regularization in an appropriate n-dimensional topology.

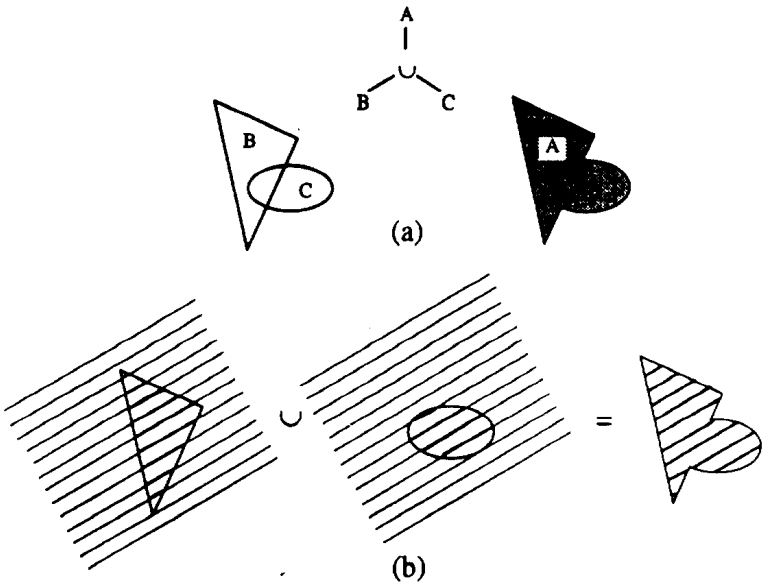

Fig. 2. Boolean simplification in E^2.

algorithms. If the use of approximations of adjustable accuracy is admitted at the outset, nearly all the intrinsic requirements can be met by providing representations and algorithms that support the following

- Set operations on solids: repeated unions for generating discrete sweep approximations, and for implementing by discrete approximation the M-ops needed to dilate and contract the desired Part; intersections to generate the test solids (e.g. $V \cap_3$ Fixtures) used to detect error conditions; set-differences to model material removal and to test for containment;
- Null-object detection: to determine whether the test solids are empty. Null-object detection is trivial in representation schemes that provide a unique representation (typically the empty representation) for the empty set, but not trivial in other schemes, such as Constructive Solid Geometry (CSG).

Thus, the key decisions devolve to a search for representations and algorithms that support these two central capabilities.[2]

The literature on NC verification abounds with articles describing verifiers based on boundary representations (B-reps), CSG representations, spatial enumerations, graphic z-buffers, and some hybrid schemes. Our experience indicates that *ray representations* (ray-reps) provide the most efficacious hosts for NC verification, and for several other demanding applications cited briefly later.

[2] These requirements are intrinsic to the particular model we used for verification, rather than to verification per se. Model-building is an exercise in engineering judgement. The rationale for our current model is reported in [6,10]. This model reflects 15 years of research on verification, during which we built three generations of verification systems.

Thus, we shall devote the remaining sections of this article to summarizing (1) the properties that make ray-reps so powerful, and (2) some computing strategies for handling ray-reps.

3. Ray representations

Fig. 1 shows a ray-rep RR(A;G) of a two-dimensional (2D) solid 'A'. It is a collection of line segments generated by 'clipping' (i.e. *classifying*) a regular grid of lines, G, against A. In 3D, the ray-rep may be defined as follows:

$$\mathrm{RR}(A;G) = \bigcup_{m,n} (L_{m,n} \cap_1 A), \tag{4}$$

where $L_{m,n}$ is a line in a 3D grid G indexed by m and n. RR(A;G) is represented as a collection of parametric endpoints, each with one or more associated tags, plus seven numbers that define the position, direction, and spacings of the grid G. In Fig. 1, the tag ∂H_j denotes a pointer to the halfspace boundary that generated $p(t_i)$; tags can carry other information as well, such as material and surface properties. The data in RR(A;G) are triply indexed – by the m and n indices of the grid, and by parametric t-values along each line.

Ray-reps are relatively verbose. For example, G may contain $10^3 \times 10^3$ lines, and each active line may be decomposed into several line segments by the classification process; each segment requires several bytes to define its endpoints, and more bytes to carry its tag(s). However, the very simple and regular structure of ray-reps makes them easy to store, retrieve, and address. The major properties of ray-reps may be summarized as follows:

1. Spatial addressability and spatial hashing. As noted earlier, ray-reps are triply indexed, and hence are

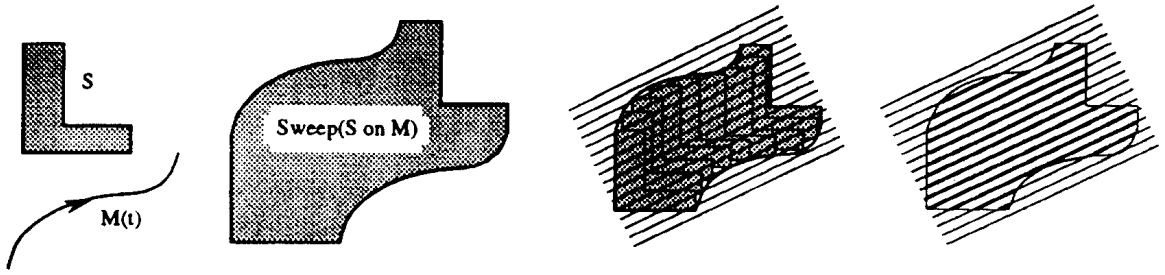

Fig. 3. Spatial sweeping in E^2.

bidirectionally addressable. In other words, the mappings

$$(m, n, i) \rightarrow p(x, y, z), \tag{5a}$$

$$p(x, y, z) \rightarrow (m, n, i) \tag{5b}$$

are computable in essentially constant time.

2. Directionality. Ray-reps may be regarded merely as sampled boundary representations, but ray-rep sample sets have a special property – directionality – that makes them almost unique amongst practical representations for solids. A complete directional representation at a point (usually within the solid) would represent the solid with ordered sets of boundary intercepts along all rays emanating from the point. Directional reps are optimal for certain applications, e.g. those involving ray*tracing*, but they are usually impractical because they are essentially existential. A ray-rep may be regarded as a *section* of a complete directional rep, and several such sections – three, say, in orthogonal directions – provide practical means for calculating approximations to complete directional reps. Ray-reps are the only 'practical' representation we know that exhibit directionality.

3. Boolean simplification. Fig. 2a shows a simple boolean composition of (2D) solids. A ray-rep of the composition can be obtained by composing the rays-reps of each solid (Fig. 2b), viz.

$$\mathrm{RR}(A\langle \mathrm{op}\rangle_n B; G) = \mathrm{RR}(A; G)\langle \mathrm{op}\rangle_1 \mathrm{RR}(B; G), \tag{6}$$

where $\langle \mathrm{op}\rangle_n$ denotes a regularized boolean operation ($\cap$ or $\cup$ or $-$) on n-dimensional solids regularized in the topology of E^n, and $\langle \mathrm{op}\rangle_1$ denotes $\langle \mathrm{op}\rangle$ applied to segments in each line and regularized in the one-dimensional (1D) topology of the line. This property reduces an n-dimensional (usually 3D) problem into a series of independent 1D problems.

4. Rigid motion. It is easy to show that, in general,

$$\mathrm{RR}(A@M; G) \neq \mathrm{RR}(A; G)@M, \tag{7}$$

i.e. motion transformation and ray-rep are not commutative. However, it is also easy to show that for any rigid motion,

$$\mathrm{RR}(A@M; G) = \mathrm{RR}(A; G@M^{-1})@M, \tag{8}$$

and this is useful, especially when the representation of A has a large number of primitives requiring transformation by M; see [4] or [1].

5. Discrete translations. These are an important class of rigid motions – specifically, translations by integer distances that correspond to the line spacings in the grid G. Eq. (7) becomes an equality when M is a discrete translation, and $\mathrm{RR}(A@M;G)$ may be computed simply by shifting indices and adding a constant to every t-value in $\mathrm{RR}(A;G)$.

6. Null-set representation. The empty ray-rep represents the empty set.

7. Completeness. Approximate representations of solids can be constructed easily from ray-reps; columnar decompositions, faceted b-reps (via 'tiling' of ray end-points), and octrees are examples. As the spacing between rays increases, the quality of the approximation decreases and small features are lost. Less obviously, under suitable conditions ray-reps with boundary tags can be *complete (unambiguous) and exact* representations. To see what this means, let $\mathrm{CSG}_i(A)$ and $\mathrm{CSG}_k(A)$ be two possibly different CSG representations of a solid A. (CSG_i and CSG_k are complete and unambiguous but generally not unique.) A ray-rep of A is produced from $\mathrm{CSG}_i(A)$, viz.

$$\mathrm{RR}(A; G) = f(\mathrm{CSG}_i(A)), \tag{9a}$$

and if the ray-rep is complete,

$$\mathrm{CSG}_k(A) = g(\mathrm{RR}(A; G)). \tag{9b}$$

The conversion f (ray-rep generation) is discussed briefly later. See [8] for an explanation of the conversion g, which is new and unusual, and the conditions governing ray-rep completeness.

4. Using ray representations

The properties of ray-reps summarized before can be used effectively in a variety of applications. We survey briefly those listed in Tables 1 and 2 that are relevant to NC verification. For details and examples of other uses of ray-reps see [7].

- Regularized set operations on solids for collision and invasive machining detection, as well as material removal modeling, can be processed effectively by using the Boolean simplification property.
- Ray-reps provide a direct and simple, albeit approximate and computationally intense, solution to

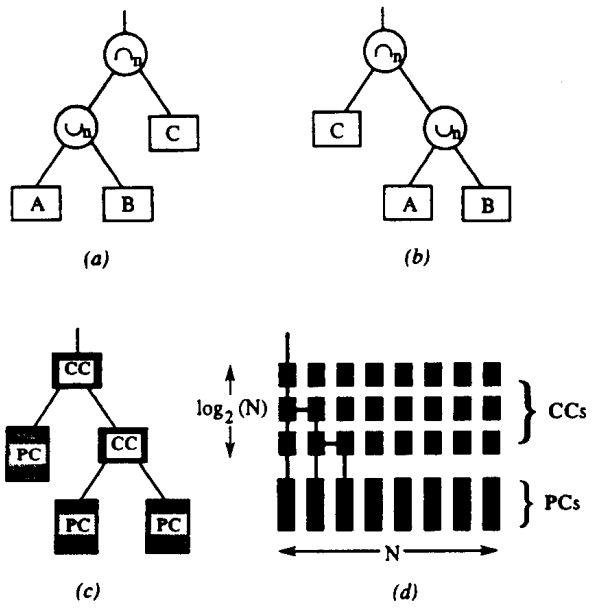

Fig. 4. Mapping a CSG classification algorithm into hardware.

computing swept solids: simply instantiate ray-reps of A in Eq. (1) at closely spaced points on M, viz.

$$\text{Sweep}(A \text{ on } M(t)) \cong \bigcup_i A@M(t_i), \tag{10}$$

and then use the boolean simplification property to calculate the union of all instances. Fig. 3 illustrates this approach. Many examples can be found in [1,6,7] and other Cornell publications.

- M-ops are needed in feature-localized tolerancing to model offset (dilated, contracted) solids. They may be implemented as sweeps – specifically, through unions of repeated instantiations of one solid on the boundary of the other, per Eq. (3). (This is a very loose description; see [4,6,7] for more rigorous definitions and a variety of examples.)
- Mass-properties can be obtained from a columnar decomposition that is naturally induced from a ray-rep, and variations of material properties can be tracked via tags.
- Simple graphic rendering, e.g. Phong shading, can be performed by using approximated surface normals in tiled b-reps inferred from ray-reps. For more refined graphics, such as in ray-tracing, rays cast in arbitrary directions must be classified against solids. This can be done approximately, but very cheaply, against ray-reps by using a '2.5-D DDA (Digital Differential Analyzer)' that 'hashes' the random ray against the ray-rep, i.e. exploits the addressability of the ray-rep. Details, examples, and applications are discussed in [7,11].
- Visibility determination for viewing directions along rays in the ray-rep grid is easy because ray-reps contain all 'in' segments along the ray, and thus the problem can be solved by table look-up.

5. Processing ray representations in parallel

Ray-reps are simple but relatively verbose, and thus ray-rep processing involves straightforward but intense computation. For example, generating the ray-rep of a sphere using a grid with $10^3 \times 10^3$ lines requires $O(10^8)$ arithmetic operations. Contemporary serial computers can sustain this level of effort but, in general, ray-rep processing requires more power than contemporary or near-term foreseeable serial computers can provide. To see this, consider evaluating the Minkowski sum of the sphere with itself using the simple-minded approach described earlier, i.e. instantiate one sphere at every ray intercept of the other. At the given grid resolution, the computational requirements now increase by orders of magnitude, rendering the task difficult, if not hopeless, in contemporary workstations.

Sometimes the computational intensity associated with ray-rep processing can be reduced by better algorithms. There are obvious simplifications in the foregoing example but, in general, algorithmic improvements are hard to contrive and have limited scope. A more powerful strategy exploits the properties of ray-representations to create simple algorithms that utilize copious but easily parallelizable calculations, and then deploy overwhelming computing power. The key to effective use of this 'brute-force' approach lies in (1) exploiting suitable decompositions of ray-rep computations, and (2) understanding the data-accessibility requirements that each such decomposition imposes. The former determines what can be processed in parallel, whereas the latter establishes processing precedence (order of calculations) and the character of the 'data trafficking' that is necessary.

We shall examine these issues briefly and review some parallel computer architectures that we studied and found suitable. The discussion is organized in terms of two major families of decompositions: *structural*, which exploit intrinsic parallelism in the syntactic structure of the representation of solid A, and *spatial*, which achieve parallelism by subdividing the defining grid G.

5.1. Structural decomposition

Useful structural decompositions derive from algorithms for generating ray-reps. The fundamental computation in such algorithms is line:solid classification [9]: to generate $R(A;G)$, each line in the grid G must be classified against the solid A and divided into "in" and "out" segments. The design of line:solid classification algorithms, and hence the opportunities for decomposition, are dominated by the character of the representation of the solid A [1,3]. (Ref. [1] summarizes the logic used to avoid handling "on" segments in the classification process.)

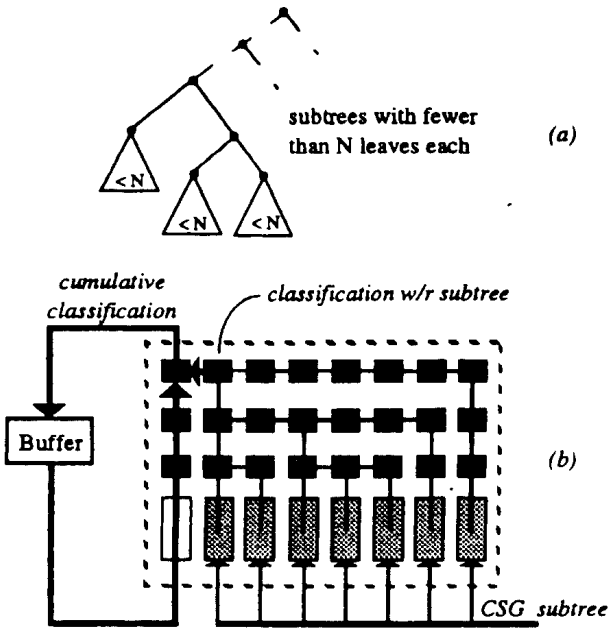

Fig. 5. Processing large ray-reps via recirculation.

The simplest case arises when A is given as a boolean union or intersection of multiple ray-reps. This occurs when computing spatial sweeps or processing M-ops by the instantiation method described earlier. As discrete versions of these operations involve sequences of commutative operators, the computation breaks naturally into a (large) set of independent tasks that can be carried out in any order. As a result, the data trafficking requirements are simple and can be accommodated easily. (An example is shown in Fig. 5 and explained later.)

When the solid is represented in CSG, the following recursive classification procedure defines a natural structural decomposition. (When the solid is represented in other schemes, the opportunities for decomposition and their consequences must be examined on a case-by-case basis. This is usually a non-trivial exercise.)

Classify (Grid, Solid)
if (Solid is a Primitive_Solid) *then* return *PrimClass* (Grid, Solid)
{*PrimClass* classifies the Grid against a primitive solid}
else return *Combine* (*Classify* (Grid, Left_Subtree (Solid), *Classify* (Grid, Right_Subtree (Solid), Operator (Solid))
{*Combine* implements the boolean composition of the subtree classifications}
end_Classify.

Observe that there are just two generic calculations, *PrimClass* and *Combine*. An 'unwinding' of this recursive algorithm reveals that invocations of PrimClass are independent of one another. Hence, all PrimClass calculations may be done in parallel. In contrast, invocations of Combine require the results of two subtree combinations and must be properly sequenced. Supplying the operands to a succession of Combine operations can be effectively handled by 'pipelining'. These observations can guide the design of algorithms to control general purpose parallel computers, or can guide the design of specialized hardware – highly parallel computers organized specifically for (Grid, CSG (Solid)) classification. We shall summarize an extant hardware solution – the RayCasting Engine (RCE) –because it is novel and effective [1,3].

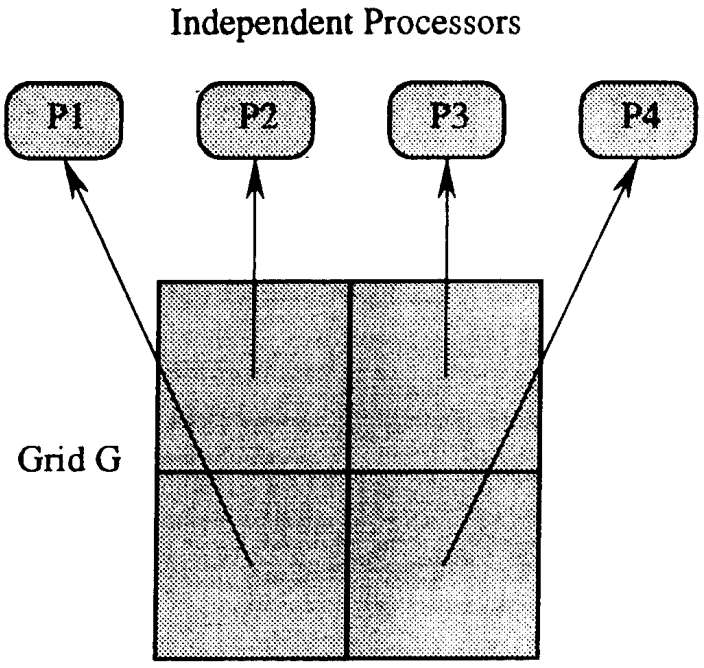

Fig. 6. Simple spatial decomposition with static load allocation.

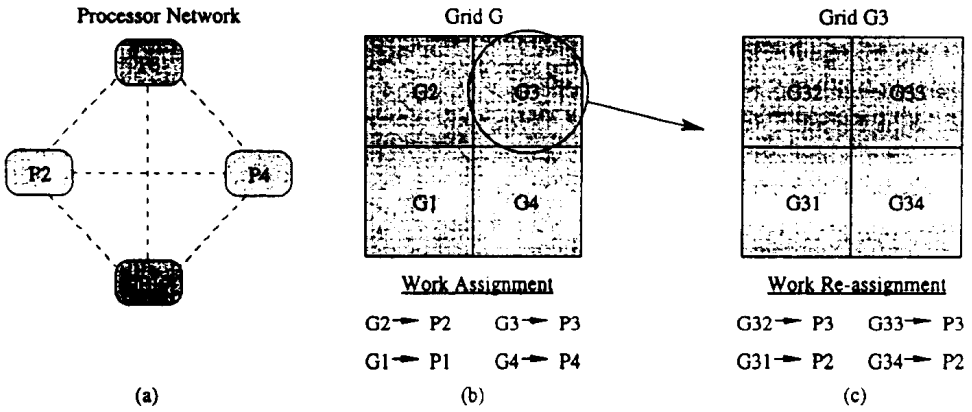

Fig. 7. Dynamic load balancing.

The RCE provides thousands of specialized processors that implement the PrimClass and Combine procedures, and can be configured (i.e. interconnected, via programming) to mimic any CSG representation of a solid. For practical reasons dictated by VLSI technology, the specialized processors are embedded in a 2D grid, with all PrimClass processors on the bottom of the grid. Two key facts motivated the architecture: (1) any CSG tree may be reorganized as a right-heavy tree, and (2) the nodes of any N-leaf right-heavy tree may be mapped into an $N \times \log_2 N$ grid.

Fig. 4 shows such a mapping. The CSG tree in Fig. 4a is re-represented as a right-heavy tree in Fig. 4b; this is associated with the programmably configured hardware tree in Fig. 4c, whose leaves are primitive classifiers (PCs) and whose nodes are classification combiners (CCs). The hardware tree is then mapped into the hardware array shown in Fig. 4d. Each PC classifies the lines in the grid G against a particular primitive solid in the CSG representation, with results pipelined bottom-up through the array of CCs. The final result is 'returned' (one ray at a time) by the top-left (root) processor. As the PC and CC calculations are simple, the data traffic through the dedicated processor interconnections can be accurately predicted, and the predictions can be used to optimize the hardware design. See [3] and references cited therein for a fuller explanation of the design strategy, hardware specifics, and performance benchmarks.

While the architecture in Fig. 4d has many interesting properties, we emphasize only the property that is most relevant to our discussion. The capacity of the RCE (as characterized by N, the number of PCs) limits the complexity of the mechanical parts that can be processed in a single pass through the machine. However, because a CSG tree may be partitioned into subtrees, ray-reps of 'large' objects may be accumulated by combining 'old' results with the current subtree classifications. This strategy, called recirculation, requires buffer storage and a direct-entry path into the CC array via interconnection busses, see Fig. 5. Recirculation also provides an ideal mechanism for processing boolean unions or intersections of multiple ray-reps – for example, to produce representations of sweeps and M-sums.

5.2. *Spatial decomposition*

Spatial decomposition entails partitioning the defining grid G of a ray-rep RR(A;G) into 'manageable clumps' of rays that can be processed separately. Such partitioning is facilitated by the very regular, indexable, and spatially addressable structure of the grid G. An example of this approach is shown in Fig. 6, where a grid is split into four equal parts and processing is performed on distinct processors. The only significant data traffic involved in this simple computational model is collecting ray-rep segments from remote locations as they are produced in order to merge them into a single data stream.

Spatial decomposition is the most obvious source of parallelism in ray-rep processing, but harnessing spatial parallelism in practice is harder than the previous example indicates. Simple static partitioning produces coarse, uneven, and unpredictable partitions of the work to be done, and engenders highly non-uniform loading of the computational resources.

One effective approach to solving this problem is to use dynamic load balancing to control the operation of a network of peer processors capable of asynchronous all-to-all communication. The main idea is illustrated in Fig. 7. Initial task assignments are generated by partitioning the grid into parts that are expected to yield an equal distribution of work among processors (Fig. 7b). Any processor that "finishes early" may request additional work from overloaded peers. Subsequent task parceling is done by subdividing overloaded processors' grids, e.g. in the quadtree fashion illustrated in Fig. 7c, and assigning new (sub)tasks to underloaded processors. This approach is suitable for a variety of Fig. 7a-type architectures ranging from clusters of workstations to the IBM SP Power Parallel machines.

The foregoing discussion illustrates two primary approaches – structural and spatial – to computational partitioning, which is central to efficient parallel processing of ray-reps. Each approach has several variants, and effective ray-rep systems use both approaches.

Our interest in parallel processing dates back to the 1970s. We became practically involved in 1982, when the RCE collaboration with Gershon Kedem was launched [10]. The pivotal period was 1989–1993; in those five years, Kedem's 2000-processor RCE-1.0 ran flawlessly, our application R&D burgeoned, and most of the views expressed here were formed. In the mid-1990s we gradually shifted our computing resource base from the hardware RCE to RCE emulators running on workstations and on SP-1 and SP-2 Power Parallel computers, and then we moved from RCE emulation to more general distributed processing, in which structural partitioning was exploited within a 'global' spatial partitioning strategy. Our most recent work was focused on load balancing in distributed ray-rep processors [2].

6. Remarks

The message set forth at the outset is simple: focus on the mathematical requirements that are intrinsic to a particular application, and exploit the continuing proliferation of computing power to construct solutions that are mathematically direct, clear, and robust.

NC verification provides a textbook example for illustrating the application of these principles. We identified as central requirements null-object detection and set operations on stationary, swept, and Minkowski-summed and -differenced (i.e. dilated and contracted) solids. We elected to implement solution facilities through ray-reps, because ray-reps enable swept and M-summed solids to be approximated, composed, and null-tested directly and robustly, and at affordable cost if the natural decompositions associated with ray-reps and ray-rep procedures are exploited to design effective distributed algorithms or hardware.

We do not claim that ray-reps are the optimal implementational medium for NC verification or the other applications we have studied (rendering, boundary-value problem solving, and more, see [1,7,11] and references cited therein). They are simply the most effective representations we found for the cited applications, and they make adherence to our problem-solving principles very easy.

A final reiteration, "Essentially unlimited (really cheap) computing power, if used sensibly, can yield excellent application (problem-solving) systems – systems that are robust, responsive, easy to use, easy to build, and easy to understand. Copious computing allows one to reformulate problems from first principles, which often means implement or approximate the mathematical definitions as directly as possible [3]."

Acknowledgements

The research summarized here was supported or abetted by a sequence of grants from the National Science Foundation, by contracts from Sandia National Laboratories, by software contributed by the Unigraphics Division of Electronic Data Systems Inc., and by a graduate fellowship provided by the IBM Corporation.

References

[1] Ellis J, Kedem G, Lyerly T, Thielman D, Marisa R, Menon J, Voelcker H. The ray-casting engine and ray representations: a technical summary. International Journal of Computational Geometry and Applications, 1991;1(4): 347–80, see also Proc. ACM /SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications, pp. 255–67; Austin, TX; 1991.

[2] Hartquist EE, et al. Processing ray-representations in parallel: options, algorithms and results, Technical Report CPA98-4, Sibley School of Mechanical Engineering, Cornell University, 1998.

[3] Kedem G, Voelcker HB. The ray casting project: massively parallel computation for mechanical design and manufacturing. In: Snyder L, editor. Proc. NSF Conference on Experimental Research in Computer Systems, Arlington, VA: National Science Foundation, 1996 pp. 69–88.

[4] Menon J, Voelcker H. Mathematical foundations 1: set theoretic properties of ray representations and M-ops on solids, Technical Report CPA91-9, Sibley School of Mechanical Engineering, Cornell University, 1991; rev. 1993.

[5] Menon JP, Voelcker HB. Toward a comprehensive formulation of NC verification as a mathematical and computational problem, Proc. ASME 1992 Winter Annual Meeting, Anaheim CA. In: Dutta D, et al., editors. American Society of Mechanical Engineers, New York, 1992, Vol. 59, pp. 147–164;. Also Journal of Design and Manufacturing 1993;3:263–77.

[6] Menon JP, Robinson DM. Advanced NC verification via massively parallel raycasting: extensions to new phenomena and geometric domains. ASME Manufacturing Rev. 1993;6(2):141–54.

[7] Menon J, Marisa R, Zagajac J. More powerful solid modeling through ray representations. IEEE Computer Graphics and Applications 1994;14(3):22–35.

[8] Menon J, Voelcker H. On the completeness and conversion of ray representations of arbitrary solids, Proc. ACM/IEEE Third Symposium on Solid Modeling and Applications, Salt Lake City, UT, 1995, pp. 175–86.

[9] Tilove RB. Set membership classification: a unified approach to geometric intersection problems. IEEE Trans. on Computers 1980;C-29(20):874–83.

[10] Voelcker HB, Requicha AAG. Research in solid modeling at the University of Rochester: 1972–1987, Fundamental Developments of Computer-Aided Geometric Modeling In: Piegl L, editor. London: Academic Press Ltd., 1993, pp. 203–54.

[11] Zagajac J. A fast method for estimating discrete field values in early engineering design. IEEE Transactions on Visualization and Computer Graphics 1996; 2(1) 35–43; see also Proc. ACM/IEEE Third Symposium on Solid Modeling and Applications, Salt Lake City, UT, 1995, pp. 420–30.

Eugene Hartquist is the Computer Operations Manager for the Sibley School of Mechanical and Aerospace Engineering at Cornell University. He worked as a senior research engineer in the field of solid modeling from 1973 to 1998, first at the University of Rochester (the PADL-1 and -2 Projects) and then at Cornell (the RayCasting and BCSG Projects). He earned a BS/EE degree from the university of Rochester in 1969, and a MS/EE degree at Ohio State University in 1972.

Jai Menon is a Program Director of Internet Media in the IBM Internet Division, where he leads research, development, and marketing on media technologies to facilitate electronic commerce, and an Adjunct Professor at SUNY/Stony-Brook. He earned degrees in mechanical engineering at IIT/Delhi (BS, 1986) and Cornell University (MS, 1989; PhD, 1992), the last for research on Constructive Shell Representations for representing freeform surfaces and solids in CSG environments. His current interest range over geometric modeling, graphics and visualization, parallel computation, and manufacturing.

K. Suresh is a senior R&D engineer at Kulicke & Soffa Industries, Inc., where he is designing semiconductor assembly equipment. He earned BS, MS and PhD degrees in mechanical engineering at IIT/Madras, UCLA, and Cornell Universities, respectively, the last in 1998 for research on interior (dual) formulations of boundary value problems. His current primary interest is in developing new representations, methods, and tools for conceptual design.

Herbert Voelcker has held the Charles Lake Chair in mechanical engineering at Cornell University since 1986. Before that he was a member of the electrical engineering faculty at the University of Rochester for 25 years. He holds degrees in mechanical and electrical engineering from MIT and the Imperial College of Science and Technology (London), and has worked in fields ranging from aural perception and ionospheric propagation to CNC systems and mechanical design.

Jovan Zagajac is a Principal Computer Applications Engineer in the CAD/CAM & CAE Systems Department of the Ford Motor Company, where he is involved with automotive modeling, shape optimization, and structural analysis. In the later 1980s he was a primary designer of the EDVIS (European) structural modeling system. He earned BS and MS degrees in nuclear engineering from the Polytechnic Institute of New York and the University of Michigan in the early 1980s, and a PhD in mechanical engineering from Cornell University in 1997 for research on statistical methods for solving boundary value problems.