Utilization of VLSI for Creating an Active Data Base of 3-D Geometric Models

J. Skyttä and T. Takala

Helsinki University of Technology, Laboratory of Information Processing Science
SF-02150 Espoo, Finland

INTRODUCTION

Parallelism of geometric computation can be achieved by distributing the computation efforts according to essentially three different strategies, based on functional, spatial and structural division, respectively (Mantyla 1983). The conventional and already commercialized way to introduce parallel computation for viewing 3-D geometric models is employing functional parallelism as a pipeline for performing different sequential transformation phases of the 3-D viewing operation (Clark 1981). This approach limits the number of parallel activities to the number of separable functional computational modules. A second approach for parallelism is the division of the modeling space into separable volume elements, which can be processed independently using a suitable data structure like an octree (Kronlof 1985). The logical component structure of a model gives a third distribution strategy. Then each processor answers only to the computational needs of its assigned objects.

The two latter approaches are utilized in this work. The motivation is to speed up processing of small entities within large geometric models. However, in conventional solutions a centralized data structure is used, which provides a serious communication problem. In VLSI environment the communication is usually more expensive than the actual processing. This is a totally controversial situation as compared to an ordinary computer architecture where memory access of an element is much cheaper than its actual processing. Thus also the data structure of a geometric model should be distributed among the processors, resulting in an active data base.

In a spatial or structural division the elements cannot usually be completely independent of each other, but they are interacting. Objects may range or move over boundaries of a space division, or logically different components may coincide spatially and then affect each other. Our aim is to combine the good features of both strategies. The smallest atomic logical elements of a geometric model (polygons in our case) are not destroyed, but are only roughly divided into partially overlapping spatial subareas. The processor structure is hierarchical, with small objects located in the leaf node processors and the larger objects in intermediate nodes. These nodes take care of a larger spatial area. The division strategy of objects among nodes is adaptive, such that each node contains about the same number of objects.

DATA STRUCTURES AND PROCESSOR ORGANIZATIONS

In order to be able to develop a feasible utilization schema for VLSI in the 3-D modelling environment, we must restrict ourselves to an essential subproblem of the model processing. As explained, the geometry engine (Clark 1982) solves the viewing problem efficiently. A general problem with 3-D models is the computation of a local operation between two model elements, for example an intersection of two surfaces. This can either be a part of the complete model formation (e.g. boundary evaluation or merging) or a way to access the model for providing useful exact information (e.g. a cross-section) of the object described. At this stage we make the assumption that our model is available in polygon format with desired accuracy for curvilinear objects (Takala 1986).

These polygons may be set to a central data structure, which makes possible an efficient search through the polygon space

(Tamminen 1982). This is a suitable solution for the calculation of intersections using a sequential computer, but the possibilities of parallelism offered by VLSI cannot be easily utilized with this strategy. What has been proposed previously as a suitable hardware solution has been an integrated circuit module which is able to compute some critical part in the calculation processing, for example determinant processing of triangular intersections (Yamaguchi 1985). This method introduces parallelism only into the computation, but not into the search which is performed sequentially through the data structure.

In order to solve this problem we must distribute the data structure into active elements. These elements should respond both to search and calculation commands and thus form an active data base of geometric information. In this way we can minimize the communication costs while the model information is statically stored in the processor hierarchy and only the parameters and results of computation are moved around in the network.

What is a suitable processor organization for localized 3-D model processing? We encounter the same problem in processor organization as in data organization in memory. The EXCELL data structure (Tamminen 1982) has a solution which partitions the model volume into a tree hierarchy according to the activity of the volume element, i.e. the more model entities are present in the cube, the more leaves in the data structure tree are used for representing them. However, this kind of dynamic tree building, although an efficient data structure, cannot be efficiently mapped to processor space with fixed interprocessor communication links. Of course these fixed communication channels can be utilized in a dynamic way by using for example a hexagonal processor array and packet routing for data messages, but this leads to extensive consumption of physical resources for pure wiring and inefficient worst case communication routes (long paths of information packets if the physical and logical data architectures are not compatible).

As we are tied to processor architectures being efficiently implementable by a large number of moderately simple similar elements, the alternatives that are left for us as architecture candidates are the pipeline, tree organization and a two-dimensional grid. A pipelined structure can be computationally very efficient, but it cannot by any means be a way to implement a data base approach because the ordering of the data access is not fixed in such a way as computational functions are. Two-dimensional grid architecture is very near to the real hardware world, and even some existing microprocessor designs support this approach. However, in order to support hierarchical search mechanisms in our data base we must include hierarchical ordering in the processor grid. A balanced tree maps nicely on the grid, and the extra communication channels existing on the grid can be used for read only access to the data base. Making all the updates to the data base using the tree hierarchy guarantees the consistency, but using the strict hierarchical structure for pure information consumption access is ineffective. The hierarchy tree can be utilized for searching the location of the data, but transfers of large volumes of detailed image formation data can be handled using grid channels not included in the tree organization.

THE TREE APPROACH FOR PROCESSORS

The severe problem is how to map a static processor structure into the dynamic world of 3-D models. A hierarchical tree structure is a natural way to organize a large geometric model. This hierarchy is usually based either on the logical composition of the object like in a CSG model, or on the geometric distribution of the objects in space, like in a polytree (Carlbom 1984). We need to have a free choice of the depth of hierarchy: we must either be able to produce a completely flat model by connecting all polygons to the root or we must be able to produce a deep hierarchy by creating intermediate objects. Each node of the processor hierarchy should be aware of what volume information is stored and processed below, and can then give service or reject inquiries which are broadcast from the upper level.

Both the CSG and the polytree methods often lead to very imbalanced tree structures. On the other hand the shortest interprocessor communication paths can be provided within a balanced tree structure assuming that the number of communication channels for one processor is limited to a few. A balanced polytree can be achieved by dividing the space adaptively instead of a fixed regular way. In the polytree structure geometric elements occupying more than one volume element are split by pseudoboundaries. However, we want to avoid the creation of extraneous geometric pseudoelements due to the division boundaries. This can be achieved through not splitting natural geometric entities, but placing them one level higher in the hierarchy. Thus all the geometric information involved is not stored in the leaves of the tree, but throughout the hierarchical structure including the intermediate nodes. This means that the processor nodes containing objects overlapping in space must communicate with each other. In tree hierarchy of processors this means communication load to the previous level, or if the overlap is large in volume, to several upper layers. If such a case can be predicted, the communication congestion of the upper levels can be relieved by creating data paths between the nodes of the same level. We can take for example a planar quadratic architecture with four links to neighbouring processors or even a hexagonal architecture with six links. In this situation each processor can contain in its local memory a routing table of those processors which can be reached more directly through bypassing the hierarchical tree order, which still forms the backbone of the information distribution in the network.

As we distribute the objects of our model to the processor space, i.e. we map the physical space to our architecture, we have to choose between two fundamental principles: we can either form tree hierarchy statically according to data structure of the model, or we can split the data structure dynamically according to the activity of different objects. If we choose statical division, we have the problem of dividing sets partially ordered in three dimensions into a planar hierarchy. If we take the dynamic alternative (Dippe 1984), we can possibly balance the load better, but have increased communication problems instead. Nothing can be guaranteed about the communication needs, but simulations can give useful information of correct choices. The static method lends itself to mathematical analysis, and if a clever enough algorithm for the distribution of the objects is developed, consistent analysis of the performance is possible.

DATA DISTRIBUTION IN PROCESSOR SPACE

Our approach is the latter one: to use a static processor structure of the form of a balanced binary tree (B-tree) storage for geometric objects like e.g. individual polygons of a faceted boundary model. The objects are distributed in the network in such a way that the extents (bounding boxes) of objects in different processors overlap as little as possible. Then operations with geometric locality, for example finding of intersections, can mostly be performed in parallel, and interprocessor communication is only needed between processors with overlapping extents. Basically the idea of utilizing bounding boxes is similar to that presented by Roth (1982) in the context of CSG trees. However, the possibilities of balancing the tree are better with small polygonal faces than with whole half-space primitives.

An outline of the behavior of the system is as follows: In the first phase a stream of polygons is brought into the top of the processor tree, from which they are evenly distributed down to the lowest level leaf processors. Each processor roughly divides the coordinate space into two halves, and each polygon is forwarded to either left or right subprocessor according to its extents. In order to map a binary subdivision into the 3-D space, each of the three coordinate axes is alternatively selected at successive levels for divisioning. The approach is

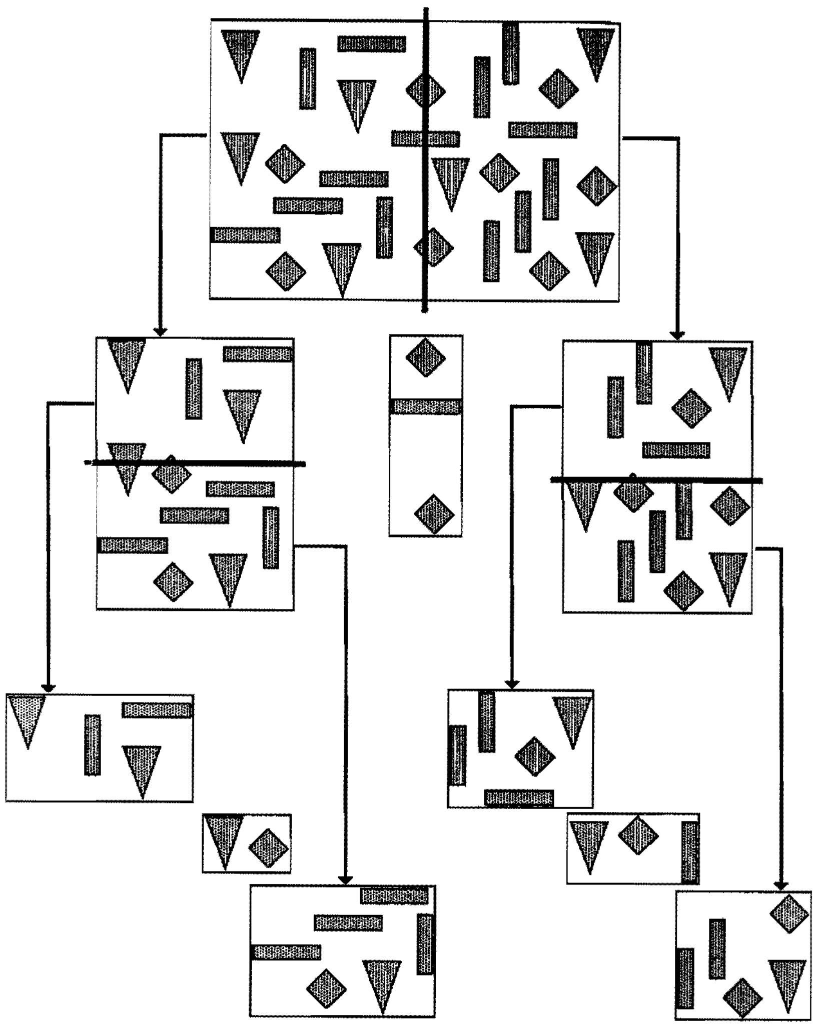
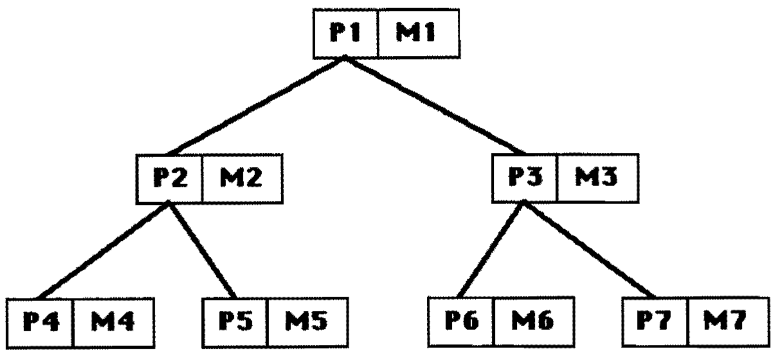


Fig. 1 : Polygon space divided into composite groups. Composite extents are shown by boxes and the subdivision boundaries by solid lines. Polygons crossing a boundary are stored in the local memory of the corresponding processor.



M_i : C_i (division value)

D_i (local data)

S_i (size of D_i)

E_i (extents of D_i)

P_i : $E'_{2i} < C_i$, $E'_{2i+1} > C_i$

$$S'_i = S_i + S'_{2i} + S'_{2i+1}$$

$$E'_i = E_i \cup E'_{2i} \cup E'_{2i+1}$$

Fig. 2 : The processor tree organization. Each processor (P) takes care that the extents at lower levels do not cross the division boundary (C). Local data (D) together with parameters (S,E,C) are stored in local memory (M). The total size (S') and extents (E') of data in a subtree are reported upwards.

similar to that in Excell (Tamminen 1982), except that if imbalance occurs, the place of space division is changed and some polygons are moved from one subtree to the other. These principles are applied recursively at each level of the tree. Each processor keeps track of the number of polygons and of their total extents in its both subtrees. The sum of their numbers and the union of their extents is recursively given to the next higher level.

As all polygons are brought into the system, it behaves in the second phase like an active geometric data base, where inquiries can be answered. The specific problem of finding intersection of two or more faceted boundary models essentially means calculating mutual intersections of individual faces. This can be done in each leaf processor for those polygons, whose extents don't overlap with any other processor's contents. For the overlapping parts each polygon is sent back up to the level where all its potential intersectors are within one processor's extents. The final results can be collected at the top level and brought out from there as a stream of polygons.

To formalize the problem for a more detailed analysis it is simplified to the one-dimensional case. Then the first phase can be presented as the problem of constructing a B-tree of one-dimensional coordinate intervals as key values. It differs from the construction of a usual ordered binary tree in that the intervals are only partially ordered keys. In order to get full ordering we give the additional optimization criterion: the sum of path lengths between overlapping intervals should be minimized in the tree. This means that the interprocessor communication needed in the second phase is also minimized.

The solution and is analysis is next to be worked out in 1-D. We are seeking for a recursive algorithm which would be the same in all processors. After solving the 1-D case, its extension to 3-D is straightforward, because the subdivision of space is done by one coordinate at a time. Variations to the solution may be caused, if lateral interprocessor communication paths are arranged within the bottom levels.

The surface intersection algorithm basically consists of finding intersection lines of polygons. Generally the complexity of calculating mutual interferences of N polygons is N-squared. However, if the number of potentially intersecting polygons can be reduced by a factor C, then the complexity will be (N/C)-squared. With geometrically localized distribution to C parallel processors this is exactly achieved, if the N polygons can be equally divided to C mutually non-overlapping spatial areas. In practice there will be partial overlaps, which are handled in the higher levels of processor hierarchy. Thus the overall reduction factor of the problem using C processors will be somewhat less than C-squared.

The idea of rough localisation of objects can be generalized to other kinds of problems also. Any characteristic parameter of the problem space can be used as a basis for grouping of objects, provided that the subproblems localized in parameter space do not interfere each other but can be solved independently in parallel. The processor organization presented in this paper implements for such problems the well-known "divide-and-conquer" strategy.

5. References

Carlbom I, Chakravarty I, Vanderschel D (1984) A Hierarchical Data Structure for Representing the Spatial Decomposition of 3D Objects. In: Computer Graphics Tokyo 84, Tokyo, Japan.

Clark J (1980) A VLSI Geometry Processor for Graphics. IEEE Computer 13: 59-68

Clark J (1980) The Geometry Engine: A VLSI Geometry System for Graphics. Computer Graphics 16: 127-133

Dippe M, Swensen J (1984) An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis. Computer Graphics 18: 149-158

Kronlof K, Tamminen M (1985) A Viewing Pipeline for Discrete Solid Modeling. The Visual Computer 1: 24-36

Mantyla M (1983) Hardware Structures for Computer Graphics and Geometric Modeling. In: NICOGRAPH '83, Tokyo, Japan, p. 68-79.

Roth S (1982) Ray Casting for Modeling Solids. Computer Graphics and Image Processing 18: 109-144

Takala T (1986) Geometric Boundary Modelling without Topological Data Structures. In: EUROGRAPHICS '86, Lisbon, Portugal, p. 115-128

Tamminen M, Sulonen R (1982) The EXCELL Method for Efficient Geometric Access to Data. In: 19th Design Automation Conference, Las Vegas, p. 345

Yamaguchi F, Tokieda T (1985) A Solid Modeler with a 4x4 Determinant Processor. IEEE Computer Graphics and Applications 5: 51-59